
Toward Certified Robustness Against Real-World Distribution Shifts

Haoze Wu^{*1} Teruhiro Tagomori^{*12} Alexandar Robey^{*3} Fengjun Yang^{*3} Nikolai Matni³ George Pappas³
Hamed Hassani³ Corina Păsăreanu⁴⁵ Clark Barrett¹

Abstract

We consider the problem of certifying the robustness of deep neural networks against real-world distribution shifts. To do so, we bridge the gap between hand-crafted specifications and realistic deployment settings by proposing a novel neural-symbolic verification framework, in which we train a generative model to learn perturbations from data and define specifications with respect to the output of the learned model. A unique challenge arising from this setting is that existing verifiers cannot tightly approximate sigmoid activations, which are fundamental to many state-of-the-art generative models. To address this challenge, we propose a general meta-algorithm for handling sigmoid activations which leverages classical notions of counter-example-guided abstraction refinement. The key idea is to “lazily” refine the abstraction of sigmoid functions to exclude spurious counter-examples found in the previous abstraction, thus guaranteeing progress in the verification process while keeping the state-space small. Experiments on the MNIST and CIFAR-10 datasets show that our framework significantly outperforms existing methods on a range of challenging distribution shifts.

1. Introduction

Despite remarkable performance in various domains, it is well-known that deep neural networks (DNNs) are susceptible to seemingly innocuous variation in their input data. Indeed, recent studies have conclusively shown that DNNs are vulnerable to a diverse array of changes ranging from norm-bounded perturbations (Goodfellow et al., 2014; Madry et al., 2017; Wong & Kolter, 2018; Zhang

et al., 2019; Kannan et al., 2018; Moosavi-Dezfooli et al., 2016; Robey et al., 2021a) to distribution shifts in weather conditions in perception tasks (Robey et al., 2020; Wong & Kolter, 2020; Hendrycks & Dietterich, 2019; Hendrycks et al., 2020; Koh et al., 2020). To address these concerns, there has been growing interest in using formal methods to obtain rigorous verification guarantees for neural networks with respect to particular specifications (Katz et al., 2017; 2019; Tjeng et al., 2017; De Palma et al., 2021; Bunel et al., 2020b; Xu et al., 2020b; Ehlers, 2017; Botoeva et al., 2020; Anderson et al., 2019; Khedr et al., 2020; Fischetti & Jo, 2017; Bunel et al., 2020a; Dvijotham et al., 2018; 2020; Wu et al., 2022; Ferrari et al., 2022; Tran et al., 2020; Huang et al., 2017; Singh et al., 2019b;a; Lyu et al., 2020; Zhang et al., 2018; Wang et al., 2018b;a; Dutta et al., 2018; Weng et al., 2018; Salman et al., 2019; Tjandraatmadja et al., 2020; Raghunathan et al., 2018; Xiang et al., 2018; Gehr et al., 2018; Wang et al., 2021; Singh et al., 2019c; Boopathy et al., 2019; Singh et al., 2018; Müller et al., 2021; El-boher et al., 2020; Ryou et al., 2021). A key component of verification is devising specifications that accurately characterize the expected behavior of a DNN in *realistic deployment settings*. Designing such specifications is crucial for ensuring that the corresponding formal guarantees are meaningful and practically relevant.

By and large, the DNN verification community has focused on specifications described by simple analytical expressions. This line of work has resulted in a set of tools which cover specifications such as certifying the robustness of DNNs against norm-bounded perturbations (Singh et al., 2019b; Katz et al., 2019; Henriksen & Lomuscio, 2020; Wang et al., 2021). However, the practical implications of these specifications are unclear beyond the realm of malicious security threats (Biggio et al., 2013), as distribution shifts often cannot be described via a set of simple equations. While progress has been made toward broadening the range of specifications (Balunović et al., 2019; Pateron et al., 2021; Mohapatra et al., 2019; Katz et al., 2021), it remains a crucial open challenge to narrow the gap between formal specifications and distribution shifts.

Outside of the verification community, a simultaneous line of work has shown that deep generative models can be trained to provably capture real-world distribu-

^{*}Equal contribution ¹Stanford University ²NRI Secure ³University of Pennsylvania ⁴Carnegie Mellon University ⁵Carnegie Mellon University. Correspondence to: Haoze Wu <haozewu@stanford.edu>.

^{1st} Workshop on Formal Verification of Machine Learning, Baltimore, Maryland, USA. Colocated with ICML 2022. Copyright 2022 by the author(s).

tion shifts (Robey et al., 2020; Wong & Kolter, 2020; Goyal et al., 2020; Robey et al., 2021b). In particular, this body of work has shown that latent variables in generative adversarial networks (Robey et al., 2020; Goyal et al., 2020) and variational autoencoders (Wong & Kolter, 2020) can capture relevant shifts in applications such as graph contrastive learning (You et al., 2021), medical imaging (Robey et al., 2021b; Bashyam et al., 2022), and autonomous perception (Robey et al., 2020; Wong & Kolter, 2020). And while this progress has resulted in several robust training algorithms, the few verification schemes that leverage these tools assume that the generative models are piecewise-linear functions (Katz et al., 2021; Mirman et al., 2021), thereby excluding state-of-the-art architectures which heavily rely on transcendental activations (e.g., sigmoid) (Huang et al., 2018).

To bridge the gap between formal verification tools and application-driven deployment settings, in this paper we introduce a new framework for verifying the robustness of DNNs against real-world distribution shifts. Distribution shifts often cannot be described by simple analytical expressions; in the absence of such expressions, our key insight is to use deep generative models—trained to provably capture distribution shifts—to define specifications in a neural-symbolic verification framework (Xie et al., 2022). To support state-of-the-art generative models in this framework, we propose a novel abstraction-refinement algorithm for handling transcendental activation functions. We show that this innovation significantly boosts verification precision when compared to existing approaches.

Contributions. Our contributions are as follows:

- We introduce a new framework for verifying DNNs against real-world distribution shifts.
- We are the first to incorporate deep generative models that capture distribution shifts—e.g. changes in weather conditions or lighting in perception tasks—into verification specifications.
- We propose a novel abstraction-refinement strategy for transcendental activation functions.
- We show that our verification techniques are significantly more precise than existing techniques on a range of challenging real-world distribution shifts on MNIST and CIFAR-10.

2. Problem formulation

In this section, we formally define the problem of verifying the robustness of DNN-based classifiers against real-world distribution shifts. The key step in our problem formulation is to propose a unification of logical specifications with deep generative models which capture distribution shifts.

Neural network classification. We consider classification

tasks where the data consists of instances $x \in \mathbb{X} \subseteq \mathbb{R}^{d_0}$ and corresponding labels $y \in [k] := \{1, \dots, k\}$. The goal of this task is to obtain a classifier $C_f : \mathbb{R}^d \rightarrow [k]$ such that C_f can correctly predict the label y of each instance x for each (x, y) pair. In this work, we consider classifiers $C_f(x)$ defined by $C_f(x) = \arg \max_{j \in [k]} f_j(x)$, where we take $f : \mathbb{R}^{n_0} \rightarrow \mathbb{Y} \subseteq \mathbb{R}^{d_L}$ (with $d_L = k$) to be an L -layer feed-forward neural network with weights and biases $\mathbf{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{d_i}$ for each $i \in \{1, \dots, L\}$ respectively. More specifically, we let $f(x) = \mathbf{n}^{(L)}(x)$ and recursively define

$$\begin{aligned} \mathbf{n}^{(i)}(x) &= \mathbf{W}^{(i)} \left(\hat{\mathbf{n}}^{(i-1)}(x) \right) + \mathbf{b}^{(i)} \\ \hat{\mathbf{n}}^{(i)}(x) &= \rho \left(\mathbf{n}^{(i)}(x) \right) \quad \text{and} \quad \hat{\mathbf{n}}^{(0)}(x) = x \end{aligned}$$

Here, ρ is a given activation function (e.g., ReLU, sigmoid, etc.) and $\mathbf{n}^{(i)}$ and $\hat{\mathbf{n}}^{(i)}$ represent the pre- and post-activation values of the i^{th} layer of f respectively.

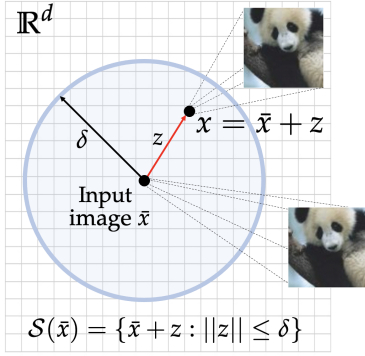
Perturbation sets and logical specifications. The goal of DNN verification is to determine whether or not a given *logical specification* regarding the behavior of a DNN holds in the classification setting described above. Throughout this work, we use the symbol Φ to denote such logical specifications, which define relations between the input and output of a DNN, i.e., $\Phi := (\Phi_{\text{in}}(x) \Rightarrow \Phi_{\text{out}}(y))$. For example, given a fixed instance-label pair (\bar{x}, \bar{y}) , the specification

$$\Phi := (\|\bar{x} - x\|_p \leq \epsilon \implies C_f(x) = \bar{y}) \quad (1)$$

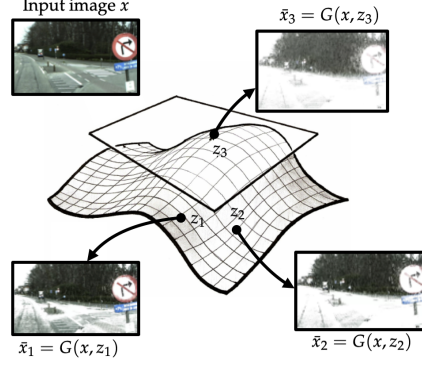
captures the property of robustness against norm-bounded perturbations by checking whether all points in an ℓ_p -norm ball centered at \bar{x} are classified by C_f as having the label \bar{y} .

Although the study of specifications such as (1) has resulted in numerous verification tools, there are many problems which cannot be described by this simple analytical model, including settings where data varies due to distribution shifts. For this reason, it is of fundamental interest to generalize such specifications to capture more general forms of variation in data. To do so, we consider abstract *perturbation sets* $\mathcal{S}(x)$, which following (Wong & Kolter, 2020) are defined as “a set of instances that are considered to be equivalent to [a fixed instance] x .” An example of an abstract perturbation set is illustrated in Figure 1b, wherein each instance in $\mathcal{S}(x)$ shows the same street sign with varying levels of snow. Ultimately, as in the case of norm-bounded robustness, the literature surrounding abstract perturbation sets has sought to train classifiers to predict the same output for each instance in $\mathcal{S}(x)$ (Robey et al., 2020; Wong & Kolter, 2020; Goyal et al., 2020).

Learning perturbation sets from data. Designing abstract perturbation sets $\mathcal{S}(x)$ which accurately capture real-



(a) **Norm-bounded perturbation sets.** The majority of the verification literature has focused on a limited set of specifications, such as ℓ_p -norm bounded perturbations, wherein perturbations can be defined by simple analytical expressions.



(b) **Real-world perturbation sets.** Most real-world perturbations cannot be described by simple analytical expressions. For example, obtaining a simple expression for a perturbation set $\mathcal{S}(x)$ that describes variation in snow would be very challenging.

Figure 1: Perturbation sets. We illustrate two examples of perturbation sets $\mathcal{S}(x)$.

istic deployment settings is critical for providing meaningful guarantees. Recent advances in the generative modeling community have shown that distribution shifts can be *provably* captured by deep generative models. The key idea in this line of work is to parameterize perturbation sets $\mathcal{S}(x)$ in the latent space \mathcal{Z} of a generative model $G(x, z)$, where G takes as input an instance x and a latent variable $z \in \mathcal{Z}$. Prominent among such works is (Wong & Kolter, 2020), wherein the authors study the ability of conditional variational autoencoders (CVAEs) to capture shifts such as variation in lighting and weather conditions in images. In this work, given a CVAE parameterized by $G(x, \mu(x) + z\sigma(x))$, where $\mu(x)$ and $\sigma(x)$ are neural networks, the authors consider abstract perturbation sets of the form

$$\mathcal{S}(x) := \{G(x, \mu(x) + z\sigma(x)) : \|z\| \leq \delta\}. \quad (2)$$

Under favorable optimization conditions, the authors of (Wong & Kolter, 2020) prove that CVAEs satisfy two statistical properties which guarantee that the data belonging to learned perturbation sets in the form of (2) produce realistic approximations of the true distribution shift (c.f. Assumption 1 and Thms. 1 and 2 in (Wong & Kolter, 2020)). To further verify this theoretical evidence, we show that this framework successfully captures real-world shifts on MNIST and CIFAR-10 in Figure 2.

Verifying robustness against learned distribution shifts. To bridge the gap between formal verification methods and perturbation sets which accurately capture real-world distribution shifts, our approach in this paper is to incorporate perturbation sets parameterized by deep generative models into verification routines. We summarize this setting in the following problem statement.

Problem 2.1. Given a DNN-based classifier $C_f(x)$, a fixed instance-label pair (\bar{x}, \bar{y}) , and an abstract perturbation set

$\mathcal{S}(\bar{x})$ in the form of (2) that captures a real-world distribution shift, our goal is to determine whether the following neural-symbolic specification holds:

$$\Phi := (x \in \mathcal{S}(\bar{x}) \implies C_f(x) = \bar{y}) \quad (3)$$

In other words, our goal is to devise methods which verify whether a given classifier C_f outputs the correct label y for each instance in a perturbation set $\mathcal{S}(x)$ parameterized by a generative model G .

3. Technical approach and challenges

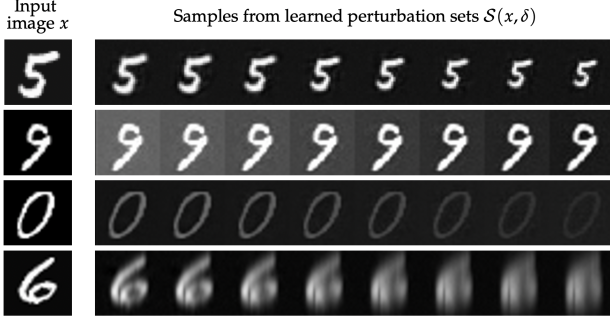
The high-level idea of our approach is to consider the following equivalent specification to (3), wherein we absorb the generative model G into the classifier C :

$$\Phi := (\|z\| \leq \delta \implies C_{Q_z}(\bar{x}) = \bar{y}) \quad (4)$$

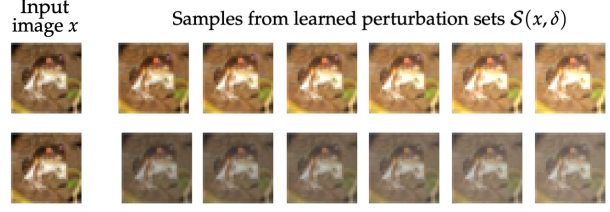
Here, $Q_z(x) = (f \circ G)(x, \mu(x) + z\sigma(x))$ is the concatenation of the deep generative model G with the DNN f . While this approach has clear parallels with verification schemes within the norm-bounded robustness literature, there is a *fundamental technical challenge*: state-of-the-art generative models heavily rely on sigmoid activations to produce realistic data; however, the vast majority of the literature concerning DNN verification considers DNNs that are piece-wise linear functions. For completeness, below we provide a general definition of sigmoid activations.

Definition 3.1 (Inflection point). A function $f : \mathbb{R} \rightarrow \mathbb{R}$ has an inflection point at η iff it is twice differentiable at η , $f''(\eta) = 0$, and f' changes sign as its argument increases through η .

Definition 3.2 (Sigmoid). A sigmoid function $\rho : \mathbb{R} \rightarrow \mathbb{R}$ is a bounded, twice differentiable function which has a non-



(a) **MNIST samples.** From top to bottom, the distribution shifts are scale, brightness, contrast, and Gaussian blur.



(b) **CIFAR-10 samples.** The distribution shifts for these sets are brightness (top) and fog (bottom).

Figure 2: Samples from learned perturbation sets. We show samples from two learned perturbation sets $\mathcal{S}(x, \delta)$ on the MNIST and CIFAR-10 datasets. Samples were generated by gridding the latent space of $\mathcal{S}(x, \delta)$.

negative derivative at each point and has exactly one inflection point.

Verification with sigmoid activations. There are a few verification techniques that can handle sigmoid functions (Singh et al., 2019b; Zhang et al., 2018; Henriksen & Lomuscio, 2020; Müller et al., 2021; Xu et al., 2020b;a). They rely on abstraction, which builds an over-approximation of the network behavior, but suffer from imprecision, especially when verifying a neural network on large input domains. Fig. 3 shows an abstraction of the popular logistic activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. If the specification holds on the abstracted network (where the sigmoid activation is abstracted with the convex region), then it holds on the original network. However, if a counter-example to the specification is found when using the abstraction, it may be spurious (as shown in the figure), and the specification might still hold on the precise, un-abstracted network.

This spurious behavior of existing verifiers demonstrates that there is a need for *refinement* of abstraction-based methods to improve the precision. For piecewise-linear activations, there is a natural refinement step:

case splitting on the activation phases. However, in the case of sigmoid activations, it is less clear how to perform this refinement, because if the refinement is performed too aggressively, the state space may explode and exceed the capacity of current verifiers. To address this technical chal-

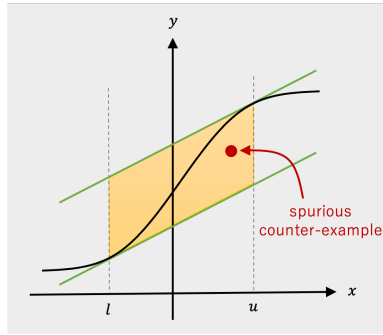


Figure 3: An abstraction of the sigmoid activation function.

Algorithm 1 VNN-CEGAR($M := \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \phi_{\text{aff}}, \phi_{\rho} \rangle, \Phi$)

```

1:  $M' \leftarrow \text{Abstract}(M)$ 
2: while true do
3:    $\langle \alpha, \text{proven} \rangle \leftarrow \text{Prove}(M', \Phi)$ 
4:   if proven then
5:     return true
6:    $\langle M', \text{refined} \rangle \leftarrow \text{Refine}(M', \Phi, \alpha)$ 
7:   if  $\neg \text{refined}$  then
8:     return false
    
```

lenge, we propose a counter-example guided refinement strategy for sigmoid activation functions which is based on the CEGAR approach (Clarke et al., 2000). Our main idea is to limit the scope of the refinement to the region around a specific counter-example. In the next section, we formally describe our proposed framework and we show that it can be extended to other transcendental activation functions (e.g., softmax).

4. A CEGAR framework for sigmoid activations

In this section, we formalize our meta-algorithm for precisely reasoning about DNNs with sigmoid activations, which is based on the CEGAR framework (Mann et al., 2021). We first present the general framework and then discuss concrete instantiations of the sub-procedures.

Verification preliminaries. Our procedure operates on tuples of the form $M := \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \phi_{\text{aff}}, \phi_{\rho} \rangle$. Here, \mathcal{V} is a set of real variables with $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{V}$ and ϕ_{aff} and ϕ_{ρ} are sets of formulas over \mathcal{V} (when the context is clear, we also use ϕ_{aff} and ϕ_{ρ} to mean the conjunctions of the formulas in those sets). A *variable assignment* $\alpha : \mathcal{V} \mapsto \mathbb{R}$ maps variables in \mathcal{V} to real values. We consider properties of the form $\Phi := (\Phi_{\text{in}}(\mathcal{X}) \Rightarrow \Phi_{\text{out}}(\mathcal{Y}))$, where $\Phi_{\text{in}}(\mathcal{X})$ and $\Phi_{\text{out}}(\mathcal{Y})$ are linear arithmetic formulas over \mathcal{X} and \mathcal{Y} ,

and we say that Φ holds on M if and only if the formula $\psi := \phi_{\text{aff}} \wedge \phi_{\rho} \wedge \Phi_{\text{in}}(\mathcal{X}) \wedge \neg\Phi_{\text{out}}(\mathcal{Y})$ is unsatisfiable. We use $M \models \Phi$ to denote that Φ holds (ψ is unsatisfiable), $M[\alpha] \models \neg\Phi$ to denote that Φ does not hold and is falsified by α (ψ can be satisfied with assignment α), and $M[\alpha] \models \Phi$ to denote that Φ is not falsified by α (α does not satisfy ψ). Given this notation, we define a sound abstraction as follows:

Definition 4.1 (Sound abstraction). *Given a tuple $M := \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \phi_{\text{aff}}, \phi_{\rho} \rangle$ and a property $\Phi = (\Phi_{\text{in}}(X) \Rightarrow \Phi_{\text{out}}(Y))$, we say the tuple $M' := \langle \mathcal{V}' \supseteq \mathcal{V}, \mathcal{X}, \mathcal{Y}, \phi'_{\text{aff}}, \phi'_{\rho} \rangle$ is a sound abstraction of M if $M' \models \Phi$ implies that $M \models \Phi$.*

Verifying DNNs. Given a DNN f , we construct a tuple M_f as follows: for each layer i in f , we let $\mathbf{v}^{(i)}$ be a vector of d_i variables representing the pre-activation values in layer i , and let $\hat{\mathbf{v}}^{(i)}$ be a similar vector representing the post-activation values in layer i . Let $\hat{\mathbf{v}}^{(0)}$ be a vector of n_0 variables from \mathcal{V} representing the inputs. Then, let \mathcal{V} be the union of all these variables, and let \mathcal{X} and \mathcal{Y} be the input and output variables, respectively; that is, \mathcal{X} consists of the variables in $\hat{\mathbf{v}}^{(0)}$, and \mathcal{Y} contains the variables in $\mathbf{v}^{(L)}$. ϕ_{aff} and ϕ_{ρ} capture the affine and non-linear (i.e., activation) transformations in the neural network, respectively. In particular, for each layer i , ϕ_{aff} contains the formulas $\mathbf{v}^{(i)} = \mathbf{W}^{(i)}\hat{\mathbf{v}}^{(i-1)} + \mathbf{b}^{(i)}$, and ϕ_{ρ} contains the formulas $\hat{\mathbf{v}}^{(i)} = \rho(\mathbf{v}^{(i)})$.

Algorithm 1 presents a high-level CEGAR loop for checking whether $M \models \Phi$. It is parameterized by three functions. The **Abstract** function produces an initial *sound abstraction* of M . The **Prove** function checks whether $M' \models \Phi$. If so (i.e., the property Φ holds for M'), it returns with *proven* set to true. Otherwise, it returns an assignment α which constitutes a counter-example. The final function is **Refine**, which takes M and M' , the property P , and the counterexample α for M' as inputs. Its job is to refine the abstraction until α is no longer a counter-example. If it succeeds, it returns a new sound abstraction M' . It fails if α is an actual counter-example for the original M . In this case, it sets the return value *refined* to false. Throughout its execution, the algorithm maintains a sound abstraction of M and checks whether the property Φ holds on the abstraction. If a counter-example α is found such that $M'[\alpha] \models \neg\Phi$, the algorithm uses it to refine the abstraction so that α is no longer a counter-example. The following theorem follows directly from Def. 4.1:

Theorem 4.2 (CEGAR is sound). *Alg. 1 returns true only if $M \models \Phi$.*

4.1. Choice of the underlying verifier and initial abstraction

The **Prove** function can be instantiated with an existing DNN verifier. The verifier is required to (1) handle piecewise-linear constraints; and (2) produce counter-examples. There are many existing verifiers that meet these requirements (Singh et al., 2019b; Katz et al., 2019; Henriksen & Lomuscio, 2020; Wang et al., 2021). To ensure that these two requirements are sufficient, we also require that ϕ'_{aff} and ϕ'_{ρ} only contain linear and piecewise-linear formulas.

The **Abstract** function creates an initial abstraction. For simplicity, we assume that all piecewise-linear formulas are unchanged by the abstraction function. For sigmoid activations, we use piecewise-linear over-approximations. In principle, any sound piecewise-linear over-approximation of the sigmoid function could be used. One approach is to use a fine-grained over-approximation with piecewise-linear bounds (Sidrane et al., 2022). While this approach can arbitrarily reduce over-approximation error, it might easily lead to an explosion of the state space when reasoning about generative models due to the large number of transcendental activations (equal to the dimension of the generated image) present in the system. One key insight of CEGAR is that it is often the case that most of the complexity of the original system is unnecessary for proving the property and eagerly adding it upfront only increases the computational cost. We thus propose starting with a coarse (e.g., convex) over-approximation and only refining with additional piecewise-linear constraints when necessary. Suitable candidates for the initial abstraction of a sigmoid function include the abstraction proposed in (Zhang et al., 2018; Singh et al., 2019b; Henriksen & Lomuscio, 2020; Müller et al., 2021; Xu et al., 2020b), which considers the convex relaxation of the sigmoid activation.

4.2. Abstraction Refinement for the sigmoid activation function.

We now focus on the problem of abstraction refinement for models with sigmoid activation functions. Suppose that an assignment α is found by **Prove** such that $M'[\alpha] \models \neg\Phi$, but for some neuron with sigmoid activation ρ , represented by variables (v, \hat{v}) , $\alpha(\hat{v}) \neq \rho(\alpha(v))$. The goal is to refine the abstraction M' , so that α is no longer a counter-example for the refined model. Here we present a refinement strategy that is applicable to *any* sound abstraction of sigmoid functions. We propose using two linear segments to exclude spurious counter-examples. The key insight is that this is always sufficient for ruling out any counter-example. We assume that ϕ'_{aff} includes upper and lower bounds for each variable v that is an input to a sigmoid function. In practice, bounds can be computed with bound-

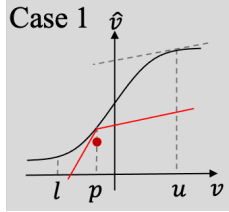
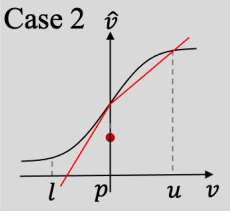
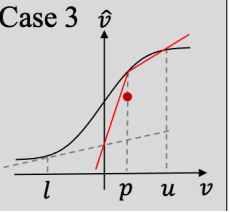
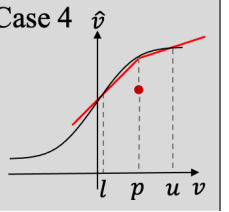
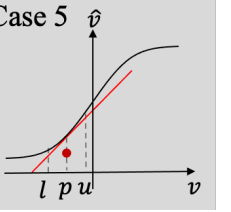
	Case 1	Case 2	Case 3	Case 4	Case 5
					
	$l < \eta, u > \eta$ $\rho''(p) > 0$	$l < \eta, u > \eta$ $\rho''(p) = 0$	$l < \eta, u > \eta$ $\rho''(p) < 0$	$l > \eta \vee u < \eta$ $\rho''(p) \leq 0$	$l > \eta \vee u < \eta$ $\rho''(p) > 0$
β	$\rho'(p)$	$\rho'(p)$	$\frac{\rho(l) - k \cdot l - \rho(p)}{\eta - p}$ where $k = \min(\rho'(l), \rho'(u))$	$\frac{\rho(p) - \rho(l)}{p - l}$	$\rho'(p)$
γ	$\min(\rho'(l), \rho'(u))$	$\frac{\rho(p) - \rho(u)}{p - u}$	$\frac{\rho(p) - \rho(u)}{p - u}$	$\frac{\rho(p) - \rho(u)}{p - u}$	$\rho'(p)$

Table 1: Choice of the slopes of the piece-wise linear abstraction refinement.

propagation techniques (Wang et al., 2018b; Zhang et al., 2018; Singh et al., 2019b).

Lemma 4.3. *Given an interval (l, u) , a sigmoid function ρ , and a point $(p, q) \in \mathbb{R}^2$, where $p \in (l, u)$ and $q \neq \rho(p)$, there exists a piecewise-linear function $h : \mathbb{R} \mapsto \mathbb{R}$ that 1) has two linear segments; 2) evaluates to $\rho(p)$ at p ; and 3) separates $\{(p, q)\}$ and $\{(x, y) | x \in (l, u) \wedge y = \rho(x)\}$.*

Leveraging this observation, given a point $(p, q) = (\alpha(v), \alpha(\hat{v}))$, we can construct a piecewise-linear function h of the following form:

$$h(x) = \rho(p) + \begin{cases} \beta(x - p) & \text{if } x \leq p \\ \gamma(x - p) & \text{if } x > p \end{cases}$$

that separates the counter-example and the sigmoid function. If $q > \rho(p)$, we add the formula $\hat{v} \leq h(v)$ to the abstraction. And if $q < \rho(p)$, we add $\hat{v} \geq h(v)$.

The values for the slopes β and γ should ideally be chosen to minimize the over-approximation error while maintaining soundness. Additionally, they should be easily computable. Table 1 presents a general recipe for choosing β and γ when the spurious counter-example point is below the sigmoid function. Choosing β and γ when the counter-example is above the sigmoid function is symmetric (details are shown in App. A). η denotes the inflection point of the sigmoid function.

Note that in case 5, β is the same as γ , meaning that a linear bound (the tangent line to ρ at p) suffices to exclude the counter-example. In terms of optimality, all but the γ value in case 1 and the β value in case 3 maximally reduce the over-approximation error among all valid slopes at $(p, \rho(p))$. In those two cases, linear or binary search techniques (Zhang et al., 2018; Henriksen & Lomuscio, 2020) could be applied to compute better slopes, but the formulas shown give the best approximations we could find without

using search.

Lemma 4.4 (Soundness of slopes). *Choosing β and γ using the recipe shown in Table 1 results in a piecewise-linear function h that satisfies the conditions of Lemma 4.3.*

Algorithm 2 $\text{Refine}(M' := \langle \mathcal{V}', \mathcal{X}, \mathcal{Y}, \phi'_{\text{aff}}, \phi'_\rho \rangle, \Phi, \alpha : \mathcal{V} \mapsto \mathbb{R})$.

```

1: refined  $\leftarrow 0$ 
2: for  $(v, \hat{v}) \in \text{AllSigmoid}(\mathcal{V}')$  do
3:   if  $\alpha(\hat{v}) = \rho(\alpha(v))$  then
4:     continue
5:   refined  $\leftarrow$  refined + 1
6:    $\beta, \gamma \leftarrow \text{getSlopes}(l(v), u(v), \alpha(v), \alpha(\hat{v}))$ 
7:    $\phi'_\rho \leftarrow \phi'_\rho \cup \text{addPLBound}(\beta, \gamma, v, \hat{v}, \alpha)$ 
8:   if  $\text{stopCondition}(\text{refined})$  then
9:     break
10: return  $\langle \mathcal{V}', \mathcal{X}, \mathcal{Y}, \phi'_{\text{aff}}, \phi'_\rho \rangle, \text{refined} > 0$ 
    
```

An instantiation of the **Refine** function for neural networks with sigmoid activation is shown in Alg. 2. It iterates through each sigmoid activation function. For the ones that are violated by the current assignment, the algorithm computes the slopes following the strategy outlined above with the **getSlopes** function and adds the corresponding piecewise-linear bounds (e.g., $\hat{v} \geq h(v)$) with the **addPLBound** function. Finally, we also allow the flexibility to terminate the refinement early with a customized **stopCondition** function. This is likely desirable in practice, as introducing a piecewise-linear bound for each violated activation might be too aggressive. Furthermore, adding a single piecewise-linear bound already suffices to exclude α . We use an adaptive stopping strategy where we allow at most m piecewise-linear bounds to be added in the first invocation of Alg. 2. And then, in each subsequent round, this number is increased by a factor k . For our evaluation, below, we used $m = 30$ and $k = 2$, which were the values

Dataset	Gen.	Class.	DP	DP+BaB		DP+BaB+CEGAR		
			δ	δ	time(s)	δ	time(s)	# ref.
MNIST	MLP_GEN ₁	MLP_CLASS ₁	0.104 \pm 0.041	0.139 \pm 0.058	7.8	0.157 \pm 0.057	84.1	1.5 \pm 1.1
	MLP_GEN ₂	MLP_CLASS ₁	0.08 \pm 0.035	0.106 \pm 0.049	20.4	0.118 \pm 0.049	114.8	1.0 \pm 1.1
	MLP_GEN ₁	MLP_CLASS ₂	0.102 \pm 0.044	0.136 \pm 0.061	16.4	0.15 \pm 0.059	120.6	1.2 \pm 1.2
	MLP_GEN ₂	MLP_CLASS ₂	0.081 \pm 0.037	0.112 \pm 0.049	60.8	0.121 \pm 0.049	191.6	0.8 \pm 1.1
	MLP_GEN ₁	MLP_CLASS ₃	0.099 \pm 0.041	0.135 \pm 0.062	41.3	0.146 \pm 0.059	186.9	1.0 \pm 1.1
	MLP_GEN ₂	MLP_CLASS ₃	0.082 \pm 0.036	0.116 \pm 0.044	75.7	0.122 \pm 0.041	163.3	0.6 \pm 1.0
CIFAR	CONV_GEN ₁	CONV_CLASS ₁	0.219 \pm 0.112	0.273 \pm 0.153	33.5	0.287 \pm 0.148	140.8	4.5 \pm 9.2
	CONV_GEN ₂	CONV_CLASS ₁	0.131 \pm 0.094	0.18 \pm 0.117	13.7	0.194 \pm 0.115	112.5	3.1 \pm 6.0
	CONV_GEN ₁	CONV_CLASS ₂	0.176 \pm 0.108	0.242 \pm 0.14	16.0	0.253 \pm 0.136	57.7	1.6 \pm 2.4
	CONV_GEN ₂	CONV_CLASS ₂	0.12 \pm 0.077	0.154 \pm 0.087	7.9	0.172 \pm 0.085	140.2	3.3 \pm 4.2

Table 2: Evaluation results of three solver configurations.

that performed best in an empirical analysis.

Theorem 4.5 (Soundness of refinement). *Given a sound abstraction M' of tuple M , a property Φ , and a spurious counter-example α s.t. $M'[\alpha] \models \neg\Phi$ and $M[\alpha] \models \Phi$, Alg. 2 produces a sound abstraction of M , M'' , s.t. $M''[\alpha] \models \Phi$.*

5. Experimental evaluation

In this section, we evaluate the performance of our proposed verification framework. We begin by comparing our method with existing approaches on two real-world distribution shifts (§5.1). Next, we benchmark the performance of our verifier on a range of challenging distribution shifts (§5.2). Finally, we use our method to show that robust training tends to result in higher levels of certified robustness against distribution shifts (§5.3).

Datasets. We consider a diverse array of distribution shifts on the MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky et al., 2009) datasets. The code used to generate the perturbations is adapted from (Hendrycks & Dietterich, 2019).

Training algorithms. For each distribution shift we consider, we train a CVAE using the framework outlined in (Wong & Kolter, 2020). For each dataset, the number of sigmoid activations used in the CVAE is the same as the (flattened) output dimension; that is, we use 784 (28×28) sigmoids for MNIST and 3072 ($3 \times 32 \times 32$) sigmoids for CIFAR-10. Throughout this section, we use various training algorithms, including empirical risk minimization (ERM) (Vapnik, 1999), invariant risk minimization (IRM) (Arjovsky et al., 2019), projected gradient descent (PGD) (Madry et al., 2017), and model-based dataset augmentation (MDA) (Robey et al., 2020).

Implementation details. We use the DeepPoly (Singh et al., 2019b) method, which over-approximates the sigmoid output with two linear inequalities (an upper and a lower bound), to obtain an initial abstraction (the Abstract function in Alg. 1) for each sigmoid and instantiate the Prove function with the Marabou neural network verifi-

cation tool. (Katz et al., 2019)¹ All experiments are run on a cluster equipped with Intel Xeon E5-2637 v4 CPUs running Ubuntu 16.04 with 8 threads and 32GB memory.

5.1. Comparison of verification performance

We first compare the performance of our proposed CEGAR procedure to other baseline verifiers that do not perform abstraction refinement. To do so, we compare the largest perturbation δ in the latent space of generative models G that each verifier can certify. In our comparison, we consider three distinct configurations: (1) DP, which runs the DeepPoly abstract interpretation procedure without any search; (2) DP+BaB, which runs a branch-and-bound procedure (Marabou) on an encoding where each sigmoid is abstracted with the DeepPoly linear bounds and the other parts are precisely encoded; and (3) DP+BaB+CEGAR, which is the CEGAR method proposed in this work.² For each verifier, we perform a linear search for the largest perturbation bound each configuration can certify. Specifically, starting from $\delta = 0$, we repeatedly increase δ by 0.02 and check whether the configuration can prove robustness with respect to $\mathcal{S}(x)$ within a given time budget (20 minutes). The process terminates when a verifier fails to prove or disprove a given specification.

For this experiment, we consider the *shear* distribution shift on MNIST and the *fog* distribution shift on the CIFAR-10 dataset (see Figure 2). All classifiers are trained using ERM. To provide a thorough evaluation, we consider several generator and classifier architectures; details can be found in App. D. Our results are enumerated in Table 2, which shows the mean and standard deviation of the largest δ each configuration is able to prove for the first 100 correctly classified test images. We also report the average runtime on the largest δ proven robust by DP+BaB and

¹We note that our framework is general and can be used with other abstractions and solvers.

²We also tried eagerly abstracting the sigmoid with fine-grained piecewise-linear bounds, but the resulting configuration performs much worse than a lazy approach in terms of runtime. Details are shown in App. E

Dataset	Perturbation	$\delta = 0.1$		$\delta = 0.2$		$\delta = 0.5$	
		robust	time(s)	robust	time(s)	robust	time(s)
MNIST	brightness	99	3.4	96	5.0	89	13.7
	rotation	51	38.6	11	80.1	1	177.9
	gaussian-blur	86	4.7	79	10.8	65	36.5
	shear	76	21.4	4	102.6	0	135.6
	contrast	90	5.9	85	11.1	74	51.0
	scale	95	8.0	84	30.8	3	122.7
CIFAR10	brightness	97	3.2	96	5.2	86	18.5
	contrast	97	3.0	95	4.6	77	40.0
	fog	84	34.3	64	69.1	11	256.0
	gaussian-blur	100	2.9	99	3	94	10.7

Table 3: Robustness of ERM against different perturbations.

DP+BaB+CEGAR, as well as the average number of abstraction refinement rounds by DP+BaB+CEGAR on those δ values. Across all configurations, our proposed technique effectively improves the verifiable perturbation bound with moderate runtime overhead. This suggests that the counterexample guided abstraction refinement scheme can successfully boost the precision when reasoning about sigmoid activations by leveraging existing verifiers. Further comparison of our techniques against existing verifiers on adversarial robustness verification of sigmoid-based classifiers are provided in App. E.

5.2. Benchmarking our approach on an array of real-world distribution shifts

We next use our proposed verification procedure to evaluate the robustness of classifiers trained using ERM against a wide range of distribution shifts. We select the first 100 correctly classified test images from the respective dataset for each perturbation and verify the robustness of the classifier against the perturbation set. Three values of the perturbation variable δ are considered: 0.1, 0.2, and 0.5. The architectures we consider for MNIST are MLP_GEN₂ and MLP_CLASS₃. For CIFAR-10 we use CONV_GEN₂ and CONV_CLASS₂. The verification results are shown in Figure 3. The “robust” columns show the number of instances that our verification procedure is able to certify within a 20 minute timeout. As one would expect, the robustness of each classifier deteriorates as the perturbation budget δ increases. For instance, for the shear transformation, the classifier is robust on 76 out of the 100 instances when $\delta = 0.1$, but is only certified robust on 4 instances when δ increases to 0.2. Information like this could help system developers to identify perturbation classes for which the network is especially vulnerable and potentially retrain the network accordingly.

5.3. Verification for various robust training algorithms

Finally, we compare the robustness and accuracy of classifiers trained using ERM, IRM, PGD, and MDA against the shear distribution shifts on the MNIST dataset. To this end,

Dataset	Train. Alg.	Test set Accuracy %		Certified Robust %	
		Standard	Generative	$\delta = 0.05$	$\delta = 0.1$
MNIST	ERM	97.9	71.6	73.2	62.4
	IRM	97.8	78.7	91.4	37.0
	PGD	97.0	79.5	91.0	73.8
	MDA	97.2	96.5	97.2	86.6

Table 4: Test set accuracy and verification accuracy

we measure the accuracy on the entire test set under the learned perturbation generative models. For each classifier, we then select the first 500 correctly classified images in its dataset and verify the targeted robustness of the classifier against the perturbation. The architectures we use are MLP_GEN₂ and MLP_CLASS₃.

Accuracy and robustness results are presented in Figure 4. Interestingly, MDA, which is perturbation-aware, outperforms the other three perturbation-agnostic training methods, on both test accuracy and robustness, suggesting that knowing what type of perturbation to anticipate is highly useful. Notice that accuracy on the perturbation set is not necessarily a good proxy for robustness: while the IRM-trained classifier has similar accuracy as the PGD-trained classifier, the former is significantly less robust on the perturbation set with $\delta = 0.1$. This further supports the need for including formal verification in the systematic evaluation of neural networks and training algorithms.

6. Related Work

Beyond norm-bounded perturbations. While the literature concerning DNN verification has predominantly focused on robustness against norm-bounded perturbations, some work has considered other forms of robustness, e.g., against geometric transformations of data (Balunović et al., 2019; Paterson et al., 2021; Mohapatra et al., 2019). However, the perturbations considered are hand-crafted and can be analytically defined by simple models. In contrast, our goal is to verify against real-world distribution shifts that are defined via the output set of a generative model. Our approach also complements recent work which has sought to incorporate neural symbolic components into formal specifications (Xie et al., 2022). Our main contribution with respect to that work is to propose the use of deep generative models in such specifications and to design new tools which can handle the unique technical challenges that arise from this setting.

Existing verification approaches. Existing DNN verification algorithms broadly fall into one of two categories: search-based methods (Katz et al., 2017; 2019; Tjeng et al., 2017; De Palma et al., 2021; Bunel et al., 2020b; Xu et al., 2020b; Ehlers, 2017; Botoeva et al., 2020; Anderson et al., 2019; Khedr et al., 2020; Fischetti & Jo, 2017; Bunel et al., 2020a; Dvijotham et al., 2018; 2020; Wu et al., 2022; Ferrari et al., 2022; Tran et al., 2020; Huang et al., 2017)

and abstraction-based methods (Singh et al., 2019b;a; Lyu et al., 2020; Zhang et al., 2018; Wang et al., 2018b;a; Dutta et al., 2018; Weng et al., 2018; Salman et al., 2019; Tjandraatmadja et al., 2020; Raghunathan et al., 2018; Xiang et al., 2018; Gehr et al., 2018; Wang et al., 2021; Singh et al., 2019c; Boopathy et al., 2019; Singh et al., 2018; Müller et al., 2021; Elboher et al., 2020; Ryou et al., 2021). While several existing solvers can handle sigmoid activation functions (Singh et al., 2019b; Henriksen & Lomuscio, 2020; Zhang et al., 2018; Müller et al., 2021; Xu et al., 2020b), they rely on one-shot abstraction and lack a refinement scheme for continuous progress. On the other hand, a separate line of work has shown that verifying DNNs containing a single layer of logistic activations is decidable (Ivanov et al., 2019), but the decision procedure proposed in this work is computationally prohibitive. To overcome these limitations, we propose a meta-algorithm inspired by counter-example-guided abstraction refinement (Clarke et al., 2000) that leverages existing verifiers to solve increasingly refined abstractions.

Verification against distribution shifts. The authors of (Wong & Kolter, 2020) also considered the task of evaluating DNN robustness to real-world distribution shifts; in particular, the approach used in (Wong & Kolter, 2020) relies on randomized smoothing (Cohen et al., 2019). This scheme provides *probabilistic* guarantees on robustness, whereas our approach (as well as the aforementioned approaches) provides *deterministic* guarantees. In a separate line of work, several authors have sought to perform verification of deep generative models (Katz et al., 2021; Mirman et al., 2021). However, each of these works assumes that generative models are piece-wise linear functions, which precludes the use of state-of-the-art models.

7. Conclusion

In this paper, we presented a framework for certifying robustness against real-world distribution shifts. We proposed using provably trained deep generative models to define formal specifications and a new abstraction-refinement algorithm for verifying them. Experiments show that our method can certify against larger perturbation sets than previous techniques.

Limitations. We now discuss some limitations of our framework. First, like many verification tools, the classifier architectures that our approach can verify are smaller than popular architectures such as ResNet (He et al., 2016) and DenseNet (Iandola et al., 2014). This stems from the limited capacity of existing DNN verification tools for piecewise-linear functions, which we invoke in the CE-GAR loop. Given the rapid growth of the DNN verification community, we are optimistic that the scalability of verifiers will continue to grow rapidly, enabling their use on

larger and larger networks. Another limitation is that the quality of our neural symbolic specification is determined by how well the generative model captures real-world distribution shifts. The mismatch between formal specification and reality is in fact common (and often unavoidable) in formal verification. And while (Wong & Kolter, 2020) shows that under favorable conditions, CVAEs can capture distribution shifts, these assumptions may not hold in practice. For this reason, we envision that in addition to these theoretical results, a necessary future direction will be to involve humans in the verification loop to validate the shifts captured by generative models and the produced counterexamples. This resembles how verification teams work closely with product teams to continually re-evaluate and adjust the specifications in existing industrial settings.

References

- Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proc. Programming Language Design and Implementation (PLDI)*, pp. 731–744, 2019.
- Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.
- Balunović, M., Baader, M., Singh, G., Gehr, T., and Vechev, M. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems* 32, 2019.
- Bashyam, V. M., Doshi, J., Erus, G., Srinivasan, D., Abdulkadir, A., Singh, A., Habes, M., Fan, Y., Masters, C. L., Maruff, P., et al. Deep generative medical image harmonization for improving cross-site generalization in deep learning predictors. *Journal of Magnetic Resonance Imaging*, 55(3):908–916, 2022.
- Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402. Springer, 2013.
- Boopathy, A., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3240–3247, 2019.
- Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3291–3299, 2020.
- Bunel, R., De Palma, A., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P., and Kumar, M. P. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, pp. 370–379. PMLR, 2020a.
- Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., and Mudigonda, P. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020b.
- Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pp. 154–169. Springer, 2000.
- Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320. PMLR, 2019.
- De Palma, A., Behl, H. S., Bunel, R., Torr, P. H., and Kumar, M. P. Scaling the convex barrier with active sets. *arXiv preprint arXiv:2101.05844*, 2021.
- Dutta, S., Jha, S., Sankaranarayanan, S., and Tiwari, A. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, 2018.
- Dvijotham, K., Stanforth, R., Goyal, S., Mann, T. A., and Kohli, P. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, pp. 3, 2018.
- Dvijotham, K. D., Stanforth, R., Goyal, S., Qin, C., De, S., and Kohli, P. Efficient neural network verification with exactness characterization. In *Uncertainty in Artificial Intelligence*, pp. 497–507. PMLR, 2020.
- Ehlers, R. Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320, 2017. URL <http://arxiv.org/abs/1705.01320>.
- Elboher, Y. Y., Gottschlich, J., and Katz, G. An abstraction-based framework for neural network verification. In *International Conference on Computer Aided Verification*, pp. 43–65. Springer, 2020.
- Ferrari, C., Muller, M. N., Jovanovic, N., and Vechev, M. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263*, 2022.
- Fischetti, M. and Jo, J. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *CoRR*, abs/1712.06174, 2017.
- Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. T. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pp. 3–18, 2018. doi: 10.1109/SP.2018.00058. URL <https://doi.org/10.1109/SP.2018.00058>.
- Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- Goyal, S., Qin, C., Huang, P.-S., Cemgil, T., Dvijotham, K., Mann, T., and Kohli, P. Achieving robustness in

- the wild via adversarial mixing with disentangled representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1211–1220, 2020.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.
- Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., Desai, R., Zhu, T., Parajuli, S., Guo, M., et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. *arXiv preprint arXiv:2006.16241*, 2020.
- Henriksen, P. and Lomuscio, A. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pp. 2513–2520. IOS Press, 2020.
- Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. Safety verification of deep neural networks. In *CAV*, 2017.
- Huang, X., Liu, M.-Y., Belongie, S., and Kautz, J. Multi-modal unsupervised image-to-image translation. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 172–189, 2018.
- Iandola, F., Moskewicz, M., Karayev, S., Girshick, R., Darrell, T., and Keutzer, K. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.
- Ivanov, R., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 169–178, 2019.
- Kannan, H., Kurakin, A., and Goodfellow, I. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.
- Katz, G., Barrett, C., Dill, D., Julian, K., and Kochenderfer, M. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pp. 97–117, 2017.
- Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pp. 443–452, 2019.
- Katz, S. M., Corso, A. L., Strong, C. A., and Kochenderfer, M. J. Verification of image-based neural network controllers using generative models. In *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, pp. 1–10. IEEE, 2021.
- Khedr, H., Ferlez, J., and Shoukry, Y. Peregrinn: Penalized-relaxation greedy neural network verifier. *arXiv preprint arXiv:2006.10864*, 2020.
- Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., et al. Wilds: A benchmark of in-the-wild distribution shifts. *arXiv preprint arXiv:2012.07421*, 2020.
- Krizhevsky, A., Hinton, G., et al. 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*, 86(11):2278–2324, 1998.
- Lyu, Z., Ko, C.-Y., Kong, Z., Wong, N., Lin, D., and Daniel, L. Fastened crown: Tightened neural network robustness certificates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5037–5044, 2020.
- Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- Mann, M., Irfan, A., Griggio, A., Padon, O., and Barrett, C. Counterexample-guided prophecy for model checking modulo the theory of arrays. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 113–132. Springer, 2021.
- Mirman, M., Hägele, A., Bielik, P., Gehr, T., and Vechev, M. Robustness certification with generative models. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 1141–1154, 2021.
- Mohapatra, J., Chen, P.-Y., Liu, S., Daniel, L., et al. Towards verifying robustness of neural networks against semantic perturbations. *arXiv preprint arXiv:1912.09533*, 2019.
- Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

- Müller, M. N., Makarchuk, G., Singh, G., Püschel, M., and Vechev, M. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021.
- Paterson, C., Wu, H., Grese, J., Calinescu, R., Pasareanu, C. S., and Barrett, C. Deepcert: Verification of contextually relevant robustness for neural network image classifiers. *arXiv preprint arXiv:2103.01629*, 2021.
- Raghunathan, A., Steinhart, J., and Liang, P. Semidefinite relaxations for certifying robustness to adversarial examples. *arXiv preprint arXiv:1811.01057*, 2018.
- Robey, A., Hassani, H., and Pappas, G. J. Model-based robust deep learning. *arXiv preprint arXiv:2005.10247*, 2020.
- Robey, A., Chamon, L., Pappas, G. J., Hassani, H., and Ribeiro, A. Adversarial robustness with semi-infinite constrained learning. *Advances in Neural Information Processing Systems*, 34:6198–6215, 2021a.
- Robey, A., Pappas, G. J., and Hassani, H. Model-based domain generalization. *arXiv preprint arXiv:2102.11436*, 2021b.
- Ryou, W., Chen, J., Balunovic, M., Singh, G., Dan, A., and Vechev, M. Scalable polyhedral verification of recurrent neural networks. In *International Conference on Computer Aided Verification*, pp. 225–248. Springer, 2021.
- Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A convex relaxation barrier to tight robustness verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019.
- Sidrane, C., Maleki, A., Irfan, A., and Kochenderfer, M. J. Overt: An algorithm for safety verification of neural network control policies for nonlinear systems. *Journal of Machine Learning Research*, 23(117):1–45, 2022.
- Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. *Advances in Neural Information Processing Systems*, 31:10802–10813, 2018.
- Singh, G., Ganvir, R., Püschel, M., and Vechev, M. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32:15098–15109, 2019a.
- Singh, G., Gehr, T., Püschel, M., and Vechev, M. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL): 1–30, 2019b.
- Singh, G., Gehr, T., Püschel, M., and Vechev, M. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2019c.
- Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K., and Vielma, J. P. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint arXiv:2006.14076*, 2020.
- Tjeng, V., Xiao, K., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.
- Tran, H.-D., Bak, S., Xiang, W., and Johnson, T. T. Verification of deep convolutional neural networks using imagestars. In *International Conference on Computer Aided Verification*, pp. 18–42. Springer, 2020.
- Vapnik, V. *The nature of statistical learning theory*. Springer science & business media, 1999.
- Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 6369–6379, 2018a.
- Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pp. 1599–1614, 2018b. URL <https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi>.
- Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.-J., and Kolter, J. Z. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*, 2021.
- Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Daniel, L., Boning, D., and Dhillon, I. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pp. 5276–5285. PMLR, 2018.
- Wong, E. and Kolter, J. Z. Learning perturbation sets for robust machine learning. *arXiv preprint arXiv:2007.08450*, 2020.
- Wong, E. and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5286–5295. PMLR, 2018.

- Wu, H., Zeljić, A., Katz, G., and Barrett, C. Efficient neural network analysis with sum-of-infeasibilities. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 143–163. Springer, 2022.
- Xiang, W., Tran, H.-D., and Johnson, T. T. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5777–5783, 2018.
- Xie, X., Kersting, K., and Neider, D. Neuro-symbolic verification of deep neural networks. 2022.
- Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.-W., Huang, M., Kailkhura, B., Lin, X., and Hsieh, C.-J. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020a.
- Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C.-J. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020b.
- You, Y., Chen, T., Shen, Y., and Wang, Z. Graph contrastive learning automated. In *International Conference on Machine Learning*, pp. 12121–12132. PMLR, 2021.
- Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. *arXiv preprint arXiv:1811.00866*, 2018.
- Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., and Jordan, M. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pp. 7472–7482. PMLR, 2019.

A. Choices of slopes (Cont.)

We present the general recipe of choosing β and γ in the case when the violation point is above the sigmoid function.

	Case 6	Case 7	Case 8	Case 9	Case 10
	$l < \eta, u > \eta$ $\rho''(p) > 0$	$l < \eta, u > \eta$ $\rho''(p) = 0$	$l < \eta, u > \eta$ $\rho''(p) < 0$	$l > \eta \vee u < \eta$ $\rho''(p) \leq 0$	$l > \eta \vee u < \eta$ $\rho''(p) > 0$
β	$\frac{\rho(p)-\rho(l)}{p-l}$	$\frac{\rho(p)-\rho(l)}{p-l}$	$\min(\rho'(l), \rho'(u))$	$\rho'(p)$	$\frac{\rho(p)-\rho(l)}{p-l}$
γ	$\frac{\rho(u)-k \cdot u - \rho(p)}{\eta-p}$ where $k = \min(\rho'(l), \rho'(u))$	$\rho'(p)$	$\rho'(p)$	$\rho'(p)$	$\frac{\rho(p)-\rho(u)}{p-u}$

Table 5: Choice of the slopes of the piece-wise linear abstraction refinement.

B. Proofs

Proof. **Theorem 4.2.** Alg. 1 returns true only if the property holds on a sound abstraction of M , which following Def. 4.1 means the property holds on M . \square

Proof. **Lemma 4.3.** This can be proved by construction using the β and γ values in Table 1 and Table 5. We next prove that those choices are sound in Lemma 4.4. \square

Before proving Lemma 4.4, we first state the following definitions and facts.³

Definition B.1 (Tangent line). The tangent line at a to the function f , denoted with $\text{TanLine}_{f,a}(x)$, is defined as: $\text{TanLine}_{f,a}(x) = f(a) + f'(a) * (x - a)$.

Definition B.2 (Secant line). Definition 2.2. Given $a, b \in \mathbb{R}$, the secant line at $[a, b]$ to a function f , denoted with $\text{SecLine}_{f,a,b}(x)$, is defined as: $\text{SecLine}_{f,a,b}(x) = \frac{f(a)-f(b)}{a-b} * (x - a) + f(a)$.

Proposition B.3. Let f be a twice differentiable univariate function. If $f''(x) \geq 0$ for all $x \in [l, u]$, then for all $a, x \in [l, u]$, $\text{TanLine}_{f,a}(x) \leq f(x)$, and for all $a, b, x \in [l, u]$, where $a < b$ and $a \leq x \leq b$, $\text{SecLine}_{f,a,b}(x) \geq f(x)$.

Proposition B.4. Let f be a twice differentiable univariate function. If $f''(x) \leq 0$ for all $x \in [l, u]$, then for all $a, x \in [l, u]$, $\text{TanLine}_{f,a}(x) \geq f(x)$, and for all $a, b, x \in [l, u]$, where $a < b$ and $a \leq x \leq b$, $\text{SecLine}_{f,a,b}(x) \leq f(x)$.

Proposition B.5. Let f be a differentiable univariate function with non-negative derivative at each point. For all γ if $\gamma \leq f'(x)$ for all $x \in [l, u]$, then $f(l) + \gamma(x - l) \leq f(x)$ for all $x \in [l, u]$.

Proof. **Lemma 4.4.** Cond. 1 and Cond. 2 hold trivially. Since $q < h(p)$, for Cond. 3, it suffices to show that $\forall (m, n) \in \{(x, y) | x \in (l, u) \wedge y = \rho(x)\}, n \geq h(m)$. More concretely, we show that a) $n \geq \rho(p) + \beta(m - p)$ for $m \in [l, p]$, and b) $n \geq \rho(p) + \gamma(m - p)$ for $m \in (p, u]$. We prove this is true for each case in Table. 1.

- **Case 1:** The segment corresponding to β is $\text{TanLine}_{\rho,p}$, and Cond. a holds by Prop. B.3. On the other hand, the choice γ is such that $\gamma \leq \rho'(x)$, for all $x \in [p, u]$. Thus Cond. b holds by Prop. B.5.
- **Case 2:** The segment corresponding to β is $\text{TanLine}_{\rho,p}$, and Cond. a holds by Prop. B.3. The segment corresponding to γ is $\text{SecLine}_{\rho,p,u}$, and Cond. b holds by Prop. B.4.

³These are partially adapted from “Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions” by Cimatti et al. We will cite the paper in the revised version of the paper.

- **Case 3:** For Cond. a, we further break it into 2 cases: $m \leq \eta$ and $m > \eta$. In the former case, the line $\rho(p) + \beta(m - p)$ is below the line $\rho(l) + \min(\rho'(l), \rho'(u))(x - l)$, which by Prop. B.5 is below ρ . When $m > \eta$, $\rho(p) + \beta(m - p)$ is below the secant line $\text{SecLine}_{\rho, \eta, p}$, which by Prop. B.4 is below ρ . On the other hand, the segment corresponding to γ is $\text{SecLine}_{\rho, p, u}$, and Cond. b holds by Prop. B.4.
- **Case 4:** The segments are both secant lines, $\text{SecLine}_{\rho, l, p}$ and $\text{SecLine}_{\rho, p, u}$, thus the conditions hold by Prop. B.4.
- **Case 5:** The segments are both tangent lines, $\text{TanLine}_{\rho, p}$, thus the conditions hold by Prop. B.3.

The proof of the soundness of the slope choices in Table. 5 is symmetric. \square

Proof. Theorem 4.5. We can prove the soundness of M'' by induction on the number of invocations of the ADDPLBOUND method. If it is never invoked, then $M'' = M'$ which is a sound abstraction. In the inductive case, it follows from Lemma 4.4 that adding an additional piecewise-linear bound does not exclude variable assignments that respect the precise sigmoid function. On the other hand, when $M[\alpha] \models \Phi$, the ADDPLBOUND method will be invoked at least once, which precludes α as a counter-example with respect to M'' . That is, $M''[\alpha] \models \Phi$. \square

C. Encoding piece-wise linear refinement using LeakyReLU

We observe that it is possible to encode the piecewise-linear bounds that we add during abstraction refinement using LeakyReLU functions. While we do not leverage this fact, we lay out the reduction to LeakyReLU in this section and leave it as future work to further leverage verification tools supporting LeakyReLU.

A LeakyReLU r_α is a piecewise linear function with two linear segments:

$$r_\alpha(x) = \begin{cases} \alpha \cdot x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases},$$

where $\alpha \geq 0$ is a hyper-parameter.

Given a piecewise linear function with two linear segments:

$$h(x) - \rho(p) = \begin{cases} \beta(x - p) & \text{if } x \leq p \\ \gamma(x - p) & \text{if } x > p \end{cases}$$

We can rewrite h as the following:

$$h(x) = \gamma * r_\alpha(x - p) + \rho(p), \text{ where } \alpha := \frac{\beta}{\gamma}$$

Note that the α value is always valid (≥ 0) because both β and γ that we choose are positive. This means that we can potentially encode the piecewise linear bounds as affine and leaky relu layers. For example, the piecewise linear upper bound $y \leq h(x)$ for a sigmoid $y = \rho(x)$ can be encoded as

$$a_1 = x - p \tag{5a}$$

$$a_2 = r_\alpha(a_1) \tag{5b}$$

$$a_3 = a_2 + \rho(p) \tag{5c}$$

$$y = a_3 + a_4 \tag{5d}$$

$$a_4 \leq 0 \tag{5e}$$

, where a_1, a_2, a_3, a_4 are fresh auxilliary variables. Eqs. a) and c) can be modeled by feed-forward layers. Eq. b) can be modeled by a leaky relu layer. If we treat a_4 as an input variable to the neural network, Eq. d) can be modeled as a residual connection. This suggest that we could in principle express the abstraction as an actual piecewise-linear neural networks (with bounds on the input variables (e.g., a_4), making it possible to leverage verifiers built on top of Tensorflow/Pytorch.

D. Details on training and CVAEs

D.1. Dataset

We consider the well-known MNIST and CIFAR-10 dataset. The MNIST dataset contains 70,000 grayscale images of handwritten digits with dimensions 28×28 , where we used 60,000 images for training and held 10,000 for testing. The CIFAR-10 dataset contains 60,000 colored images of 10 classes with dimensions $3 \times 32 \times 32$, where we used 50,000 images for training and held 10,000 for testing.

To perturb the images, we adapt the perturbations implemented in (Hendrycks & Dietterich, 2019).⁴ When training and testing the models, we sample images from the dataset and randomly perturb each image with a strength parameter c that is sampled uniformly from the ranges given in Table 6.

Table 6: Perturbation range in the training data

Dataset	Perturbation	Range of c
MNIST	brightness	[.0, .5]
	rotation	[−60, 60]
	gaussian blur	[1.0, 6.0]
	shear	[0.2, 1.0]
	contrast	[0.0, 0.4]
	translate	[1.0, 5.0]
	scale	[0.5, 0.9]
CIFAR10	brightness	[.05, .3]
	contrast	[.15, .75]
	fog	[.2, 1.5], [1.75, 3]
	gaussian blur	[.4, 1]

D.2. Architecture

On each dataset, we train a conditional variational encoder (CVAE) with three components: prior network, encoder network, and a decoder network (generator). We also train a set of classifiers. In this section, we detail the architecture of these networks. The architecture of MNIST networks are shown in Tables 7–13. Those of the CIFAR networks are shown in Tables 14–19. The output layers of the generators are activated with sigmoid function. All hidden-layers use ReLU activations.

Table 7: Prior

Type	Parameters/Shape
Input	28×28
Dense	784×1
Dense	300×1
Dense	8×2

Table 8: Encoder

Type	Parameters/Shape
Input	$28 \times 28 \times 2$
Dense	784×1
Dense	300×1
Dense	8×2

Table 9: MLP_GEN₁

Type	Param./Shape
Input	$28 \times 28 + 8$
Dense	200×1
Dense	784×1

Table 10: MLP_GEN₂

Type	Param./Shape
Input	$28 \times 28 + 8$
Dense	400×1
Dense	784×1

Table 11: MLP_CLASS₁

Type	Parameters/Shape
Input	28×28
Dense	32×1
Dense	32×1
Dense	10×1

Table 12: MLP_CLASS₂

Type	Parameters/Shape
Input	28×28
Dense	64×1
Dense	32×1
Dense	10×1

Table 13: MLP_CLASS₃

Type	Parameters/Shape
Input	28×28
Dense	128×1
Dense	64×1
Dense	10×1

D.3. Optimization

We implement our models and training in PyTorch. The CVAE implementation is adapted from that in (Wong & Kolter, 2020). On both datasets, we trained our CVAE networks for 150 epochs using the ADAM optimizer with a learning rate of 10^{-4} and forgetting factors of 0.9 and 0.999. In addition, we applied cosine annealing learning rate scheduling. Similar to

⁴https://github.com/hendrycks/robustness/blob/master/ImageNet-C/create_c/make_cifar_c.py

Table 14: Prior

Type	Parameters/Shape
Input	$32 \times 32 \times 3$
Dense	3072×1
Dense	300×1
Dense	8×2

Table 15: Encoder

Type	Parameters/Shape
Input	$32 \times 32 \times 3 \times 2$
Dense	3072×1
Dense	300×1
Dense	8×2

Table 16: CONV_GEN₁

Type	Param./Shape
Input	$32 \times 32 \times 3 + 8$
Dense	$32 \times 32 \times 4$
Conv	$3 \times 1 \times 1$ filters, padding 0

Table 17: CONV_GEN₂

Type	Param./Shape
Input	$32 \times 32 \times 3 + 8$
Dense	$32 \times 32 \times 4$
Conv	$3 \times 3 \times 3$ filters, padding 1

Table 18: CONV_CLASS₁

Type	Params./Shape
Input	$32 \times 32 \times 3$
Conv	$3 \times 3 \times 3$ filters, stride 3
Conv	$3 \times 2 \times 2$ filters, stride 2
Dense	25×1
Dense	10×1

Table 19: CONV_CLASS₂

Type	Params./Shape
Input	$32 \times 32 \times 3$
Conv	$3 \times 3 \times 3$ filters, stride 2
Conv	$3 \times 2 \times 2$ filters, stride 2
Dense	25×1
Dense	10×1

(Wong & Kolter, 2018), we increase β linearly from $\beta = 0$ at epoch 1 to $\beta = 0.01$ at epoch 40, before keeping $\beta = 0.01$ for the remaining epochs. We use a batch size of 256.

The ERM classifiers on the MNIST dataset are trained with the ADAM optimizer with a learning rate of 10^{-3} for 20 epochs. The classifiers for the CIFAR-10 dataset are trained with the ADAM optimizer with learning rate 10^{-3} for 200 epochs. The classifiers in Sec. 5.3 are trained also all trained with the ADAM optimizer with a learning rate of 10^{-3} for 20 epochs. For PGD, we use a step size of $\alpha = 0.1$, a perturbation budget of $\epsilon = 0.3$, and we use 7 steps of projected gradient ascent. For IRM, we use a small held-out validation set to select $\lambda \in \{0.1, 1, 10, 100, 1000\}$. For MDA, a step size of $\alpha = 0.1$, a perturbation budget of $\epsilon = 1.0$, and we use 10 steps of projected gradient ascent.

D.4. Computing resources

The classifiers used in Subsec. 5.3 were trained using a single NVIDIA RTX 5000 GPU. The other networks were trained using 8 AMD Ryzen 7 2700 Eight-Core Processors.

E. Additional Evaluation of CEGAR

E.1. Comparison with PRIMA (Müller et al., 2021)

A recent abstraction-based technique that handles transcendental activation functions is proposed in the PRIMA framework (Müller et al., 2021), which goes beyond single neuron abstractions and consider convex relaxations over groups of activation functions. In this section, we compare against PRIMA on the *same* sigmoid benchmarks used in its evaluation (Müller et al., 2021). We run the PRIMA implementation in the artifact associated with the paper,⁵ and run a configuration PRIMA+BaB+CEGAR which is the same as DP+BaB+CEGAR, the configuration we used in Sec. 5, except that we first use PRIMA to derive tighter variable bounds. Both PRIMA and our configurations are run on uniform hardware (the same as described in Sec. 5). Each job is given 16 threads and 30 minutes wall-clock time limit. PRIMA terminates without error on all instances. Table 20 shows the number of verified instances and the average runtime on verified instances by the two configurations. Our configuration is able to consistently solve more instances with only moderate increase on average solving time. This suggests that the meta-algorithm that we propose can leverage the tighter bounds derived by existing abstraction-based methods and boost the state-of-the-art in verification accuracy on sigmoid-based networks.

E.2. Evaluation on VNN-COMP-21 benchmarks

We also evaluate our techniques on the 36 sigmoid benchmarks used in VNN-COMP-2021. We exclude the benchmark where a counter-example can be found using PGD attack, and evaluate on the remaining 35 benchmarks. In particular, we run a sequential portfolio approach where we first attempt to solve the query with α - β -CROWN (Zhang et al., 2018; Wang et al., 2021; Xu et al., 2020b) (competition version), and if the problem is not solved, we run PRIMA+BaB+CEGAR.

⁵<https://dl.acm.org/doi/10.1145/3462308/full/>

Table 20: Comparison with PRIMA (Müller et al., 2021) on the same benchmarks used in (Müller et al., 2021)

Model	Acc.	ϵ	PRIMA		Ours	
			robust	time(s)	robust	time(s)
6x100	99	0.015	52	106.5	65	119.5
9x100	99	0.015	57	136.0	96	323.7
6x200	99	0.012	65	197.9	75	260.7
ConvSmall	99	0.014	56	100.5	63	157.8

Table 20 shows the results. As a point of comparison, we also report the numbers of the top three performing tools (Xu et al., 2020b; Wang et al., 2021; Henriksen & Lomuscio, 2020; Singh et al., 2019b; Müller et al., 2021) during VNN-COMP-21 on these benchmarks.⁶ While α - β -CROWN is already able to solve 29 of the 35 benchmarks, with the abstraction refinement scheme, we are able to solve 1 additional benchmark. We note that during the competition α - β -crown did not use up the 5 minute per-instance timeout limitation on any of these benchmarks.⁷ This suggests that the solver was not able to make further progress once the analysis is inconclusive on the one-shot abstraction of the sigmoid activations. On the other hand, our technique provides a viable way to make continuous progress if the verification attempt by the state-of-the-art fails.

Table 21: Comparison on the VNN-COMP-21 benchmarks

Model	# Bench.	α - β -CROWN		VeriNet		ERAN		Ours	
		robust	time(s)	robust	time(s)	robust	time(s)	robust	time(s)
6x200	35	29	12.9	20	2.5	19	145.5	30	83.2

E.3. Evaluation of an eager refinement strategy

We also compare the lazy abstraction refinement strategy with an eager approach where piecewise-linear bounds are added for each sigmoid from the beginning instead of added lazily as guided by counter-examples. In particular, we attempt to add one piecewise-linear upper-bound and one piecewise-linear lower-bound with K linear segments for each sigmoid activations. The segment points are even along the x-axis. We try 2 and 3 for the value of K . We then evaluate on the same MNIST benchmarks as in Table 2. The results are shown in Table 22. While the two strategies are still able to improve upon the perturbation bounds found by the pure abstract-interpretation-based approach DP, the means of the largest certified δ are significantly smaller than those of the CEGAR-based configuration we propose. Interestingly, while $K=3$ uses a finer-grained over-approximation compared with $K=2$, the bounds the former can certify is only larger on one of the six benchmark sets. This suggests that the finer-grained abstraction increases the overhead to the solver and is overall not effective at excluding spurious counter-examples on the set of benchmarks that we consider, which supports the need of a more informed abstraction refinement strategy such as the one we propose.

F. Licenses

The MNIST and CIFAR-10 datasets are under The MIT License (MIT). The Marabou verification tool is under the terms of the modified BSD license (<https://github.com/NeuralNetworkVerification/Marabou/blob/master/COPYING>).

⁶<https://arxiv.org/abs/2109.00498>

⁷https://github.com/stanleybak/vnncomp2021_results/blob/main/results_csv/a-b-CROWN.csv

Dataset	Gen.	Class.	K=2		K=3		DP+BaB+CEGAR		
			δ	time(s)	δ	time(s)	δ	time(s)	# ref.
MNIST	MLP_GEN ₁	MLP_CLASS ₁	0.137 ± 0.043	88.9	0.137 ± 0.042	109.8	0.157 ± 0.057	84.1	1.5 ± 1.1
	MLP_GEN ₂	MLP_CLASS ₁	0.109 ± 0.031	114.5	0.109 ± 0.031	199.0	0.118 ± 0.049	114.8	1.0 ± 1.1
	MLP_GEN ₁	MLP_CLASS ₂	0.126 ± 0.045	64.0	0.129 ± 0.044	95.9	0.15 ± 0.059	120.6	1.2 ± 1.2
	MLP_GEN ₂	MLP_CLASS ₂	0.108 ± 0.038	159.0	0.106 ± 0.036	133.8	0.121 ± 0.049	191.6	0.8 ± 1.1
	MLP_GEN ₁	MLP_CLASS ₃	0.132 ± 0.043	139.5	0.131 ± 0.042	190.0	0.146 ± 0.059	186.9	1.0 ± 1.1
	MLP_GEN ₂	MLP_CLASS ₃	0.105 ± 0.033	107.1	0.098 ± 0.035	87.5	0.122 ± 0.041	163.3	0.6 ± 1.0

Table 22: Evaluation results of the eager approach. We also report again the results of DP+BaB+CEGAR, which is the same as Table 2.