Toward Certified Robustness Against Real-World Distribution Shifts

Haoze Wu[§1], Teruhiro Tagomori[§1,2], Alexander Robey[§3], Fengjun Yang[§3], Nikolai Matni[3], George Pappas[3], Hamed Hassani[3], Corina Pasareanu[4], Clark Barrett[1]
[1] Stanford University, USA. [2] NRI Secure, Japan. [3] University of Pennsylvania, USA.
[4] Carnegie Mellon University, USA.
[1] {haozewu,barrett}@cs.stanford.edu [2] tteruhiro@nri-secure.com
[3] {fengjun,nmatni,pappasg,hassani}@seas.upenn.edu [4] pcorina@andrew.cmu.edu

***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 considering a neural-symbolic verification framework in which generative models are trained to learn perturbations from data and specifications are defined with respect to the output of these learned models. A pervasive challenge arising from this setting is that although S-shaped activations (e.g., sigmoid, tanh) are common in the last layer of deep generative models, existing verifiers cannot tightly approximate S-shaped activations. To address this challenge, we propose a general meta-algorithm for handling S-shaped activations which leverages classical notions of counter-example-guided abstraction refinement. The key idea is to "lazily" refine the abstraction of S-shaped functions to exclude spurious counter-examples found in the previous abstraction, thus guaranteeing progress in the verification process while keeping the state-space small. For networks with sigmoid activations, we show that our technique outperforms state-of-the-art verifiers on certifying robustness against both canonical adversarial perturbations and numerous real-world distribution shifts. Furthermore, experiments on the MNIST and CIFAR-10 datasets show that distribution-shift-aware algorithms have significantly higher certified robustness against distribution shifts.**

***Index Terms*—certified robustness, distribution shift, generative models, S-shaped activations, CEGAR**

I. 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 [1–7] to distribution shifts in weather conditions in perception tasks [8–12]. 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 [13–51]. 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 [14, 31, 44, 52]. However, while such specifications are useful for certain applications, such as protecting against malicious security threats [53], there are many other applications where real-world distribution shifts, such as change in weather conditions, are more relevant, and these often cannot be described via a set of simple equations. While progress has been made toward broadening the range of specifications [54–57], it remains a crucial open challenge to narrow the gap between formal specifications and distribution shifts.

One promising approach for addressing this challenge involves incorporating neural network components in the specifications [57–59]. Such an approach has been used in the past to verify safety properties of neural network controllers [57] as well as robustness against continuous transformations between images [58]. More recently, Xie et al. [59] generalize this approach by proposing a specification language which can be used to specify complex specifications that are otherwise challenging to define. In this paper, we leverage and extend these insights to obtain a *neural-symbolic* (an integration of machine learning and formal reasoning) approach for verifying robustness against real-world distribution shifts. The key idea is to incorporate deep generative models that represent real-world distribution shifts [8, 9, 60, 61] in the formal specification.

To realize this idea, there remains one important technical challenge: all the previous work [57–59] assumes that both the neural networks being verified and the generative models are piecewise-linear. This assumption is made in order to leverage off-the-shelf neural network verifiers [57, 59], which focus on piecewise-linear activation functions such as ReLU. However, in practice the majority of (image) generative models [62–66] use S-shaped activation functions such as sigmoid and tanh in the output layer. While there are a few existing methods for verifying neural networks with sigmoidal activations [31, 34, 50, 52, 67], they all rely on a one-shot abstraction of the sigmoidal activations and suffer from lack of further progress if the verification fails on the abstraction. To bridge this gap and enable the use of a broad range of generative model architectures in the neural symbolic verification approach, 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.

Our neural-symbolic approach has the obvious limitation that the quality of the formal guarantee depends on the quality of the neural network used in the specification. However, the approach also possesses several pragmatically useful features that mitigate this limitation: if the verification fails, then a counter-example is produced; by examining the counter-example, we can determine whether it is a failure of the model being tested or a failure of the generative model. In either case, this gives us valuable information to improve the verification framework. For example, the failure of the generative model might point us to input regions where the generative model is under-trained and data augmentation is needed. On the other hand, if the verification succeeds, then the generative model represents a large class of inputs for which we know the model is robust.

Contributions. We summarize our contributions are as follows:

- We describe a framework for verifying DNNs against real-world distribution shifts by incorporating deep generative models that capture distribution shifts—e.g., changes in weather conditions or lighting in perception tasks—as verification specifications.
- We propose a novel counter-example-guided abstraction refinement strategy for verifying networks with 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, as well as on canonical adversarial robustness benchmarks.

II. 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 $y \in [k] := \{1, \ldots, k\}$. The goal of this task is to obtain a classifier $C_f : \mathbb{R}^d \to [k]$ 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

$$C_f(x) = \arg\max_{j \in [k]} \; f_j(x), \tag{1}$$

where we take $f : \mathbb{R}^{n_0} \to \mathbb{Y} \subseteq \mathbb{R}^{d_L}$ (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 $i \in [L]$ respectively. More specifically, we let $f(x) = \mathbf{n}^{(L)}(x)$ and recursively define

$$\begin{aligned} \mathbf{n}^{(i)}(x) &= \mathbf{W}^{(i)} \left(\hat{\mathbf{n}}^{(i-1)}(x) \right) + \mathbf{b}^{(i)}, \\ \hat{\mathbf{n}}^{(i)}(x) &= \rho \left(\mathbf{n}^{(i)}(x) \right), \quad \text{and} \\ \hat{\mathbf{n}}^{(0)}(x) &= x. \end{aligned} \tag{2}$$

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 ith 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. That is, given input and output properties Φ_{in} and Φ_{out} respectively, we express logical specifications Φ in the following way:

$$\Phi := (\Phi_{\text{in}}(x) \Rightarrow \Phi_{\text{out}}(y)). \tag{3}$$

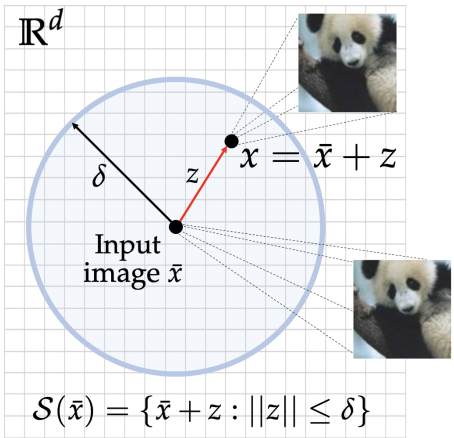
For example, given a fixed instance-label pair $(\overline{x}, \overline{y})$, the specification

$$\Phi := (||\overline{x} - x||_p \leq \epsilon \implies C_f(x) = \overline{y}) \tag{4}$$

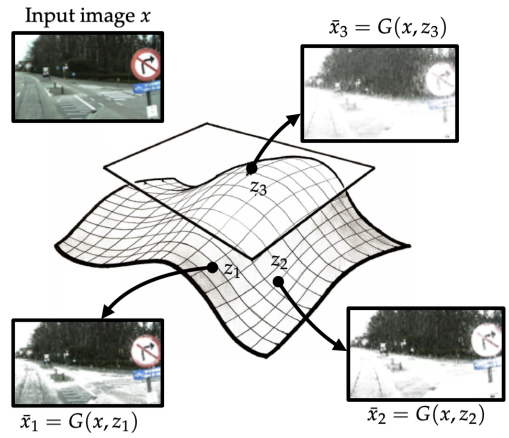
captures the property of robustness against norm-bounded perturbations by checking whether all points in an ℓ_p-norm ball centered at $\overline{x}$ are classified by C_f as having the label $\overline{y}$.

Although the study of specifications such as (4) 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 [9] 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 $\mathcal{S}(x)$ [8, 9, 60].

Learning perturbation sets from data. Designing abstract perturbation sets $\mathcal{S}(x)$ which accurately capture realistic 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 [9], 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

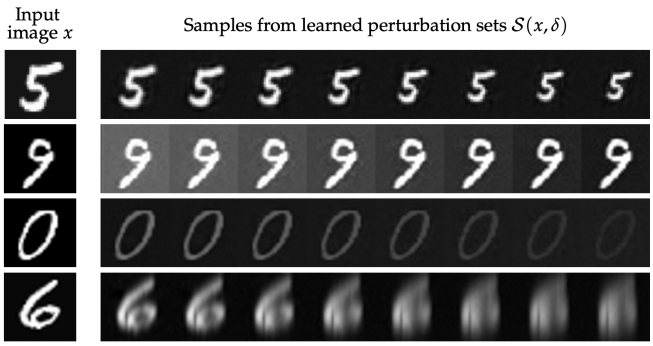


(a) Norm-bounded perturbation sets. The majority of the verification literature has focused on a limited set of specifications, such as ℓ_p-norm bounded perturbations, wherein perturbations can be defined by simple analytical expressions.

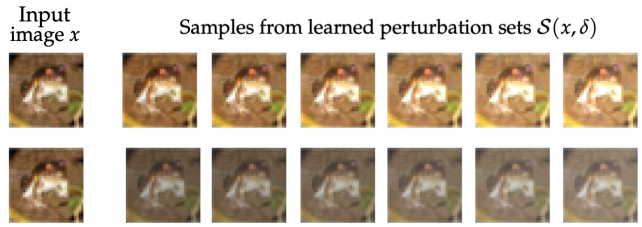


(b) Real-world perturbation sets. Most real-world perturbations cannot be described by simple analytical expressions. For example, obtaining a simple expression for a perturbation set $\mathcal{S}(x)$ that describes variation in snow would be challenging.

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



(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).

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

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

$$\mathcal{S}(x) := \{G(x, \mu(x) + z\sigma(x)) : ||z|| \leq \delta\}. \tag{5}$$

In particular, $\mu(x)$ denotes a neural network that maps each instance x to the mean of a normal distribution over the latent space $\mathcal{Z}$ conditioned on x. Similarly, $\sigma(x)$ maps x to the standard deviation of this distribution in the latent space.[1] It's noteworthy that in this framework, the perturbation upper bound δ is scaled by $\sigma(x)$, meaning that different instances x and indeed different models G will engender different relative certifiable radii.

Under favorable optimization conditions, the authors of [9] prove that CVAEs satisfy two statistical properties which guarantee that the data belonging to learned perturbation sets in the form of (5) produce realistic approximations of the true distribution shift (c.f. Assumption 1 and Thms. 1 and 2 in [9]). In particular, the authors of [9] argue that if a learned perturbation set $\mathcal{S}(x)$ in the form of (5) has been trained such that the population-level loss is bounded by two absolute constants, then $\mathcal{S}(x)$ well-approximates the true distribution shift in the following sense: with high probability over the latent space, for any clean and perturbed pair $(x, \tilde{x})$ corresponding to a real distribution shift, there exists a latent code z such that $||z||_2 \leq \alpha$ and $||G(x, \mu(x)z - \sigma(x)) - \tilde{x}|| \leq \beta$ for two small constants α and β depending on the CVAE loss. 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

[1]This notation is consistent with [9], wherein the authors use the same parameterization for conditional VAEs.

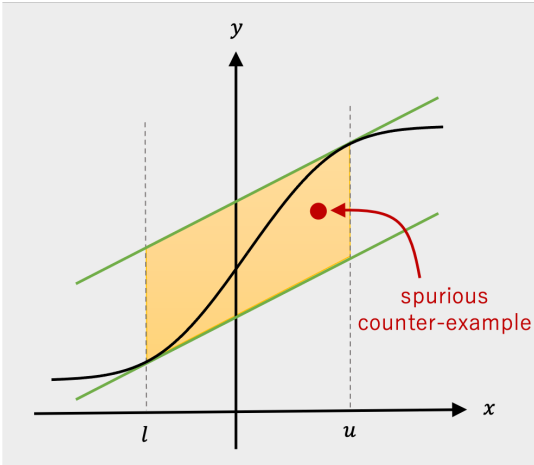


Fig. 3: An abstraction of the sigmoid activation function.

following problem statement.

Problem II.1. *Given a DNN-based classifier $C_f(x)$, a fixed instance-label pair $(\overline{x}, \overline{y})$, and an abstract perturbation set $\mathcal{S}(\overline{x})$ in the form of* (5) *that captures a real-world distribution shift, our goal is to determine whether the following neural-symbolic specification holds:*

$$\Phi := (x \in \mathcal{S}(\overline{x}) \implies C_f(x) = \overline{y}) \tag{6}$$

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.

III. Technical Approach and Challenges

The high-level idea of our approach is to consider the following equivalent specification to (6), wherein we absorb the generative model G into the classifier C:

$$\Phi = (||z|| \leq \delta \implies C_{Q_z}(\overline{x}) = \overline{y}) \tag{7}$$

In this expression, we define

$$Q_z(x) = (f \circ G)(x, \mu(x) + z\sigma(x)) \tag{8}$$

to be 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 typically use S-shaped activation functions (e.g., sigmoid, tanh) in the last layer to produce realistic data; however, the vast majority of the literature concerning DNN verification considers DNNs that are piece-wise linear functions. Therefore, existing methods for verification of generative models largely do not apply in this setting [57, 58].

Verification with S-shaped activations. In what follows, we describe the challenges inherent to verifying neural networks with S-shaped activations. For completeness, in the following definitions we provide a formal and general description of S-shaped activations, which will be crucial to our technical approach in this paper.

Definition III.1 (Inflection point)**.** *A function $f : \mathbb{R} \to \mathbb{R}$ 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 III.2 (S-shaped function)**.** *An S-shaped function $\rho : \mathbb{R} \to \mathbb{R}$ is a bounded, twice differentiable function which has a non-negative derivative at each point and has exactly one inflection point.*

In the wider verification literature, there are a handful of verification techniques that can handle S-shaped functions [18, 31, 34, 52, 67, 68]. Each of these methods is based on *abstraction*, which conservatively approximates the behavior of the neural network. We define abstraction formally in Sec. IV, but illustrate the key ideas here in Fig. 3, which shows the popular sigmoid activation function

$$\rho(x) = \frac{1}{1 + e^{-x}}. \tag{9}$$

Suppose the input x is bounded between l and u, the pre- and post- sigmoid values can be precisely described as $\mathcal{D} = \{(x, y) \mid y = \rho(x) \ \wedge \ l \leq x \leq u\}$. However, instead of using this precise representation, we could *over-approximate* $\mathcal{D}$ as $\mathcal{D}' = \{(x, y) \mid y \leq ax + b \ \wedge \ y \geq cx + d \ \wedge \ l \leq x \leq u\}$ (the yellow convex region in Fig. 3), where $ax + b$ and $cx + d$ are the two lines crossing the sigmoid function at $(l, \rho(l))$ and $(u, \rho(u))$. $\mathcal{D}'$ over-approximates $\mathcal{D}$ because the latter is a subset of the former. Abstraction-based methods typically over-approximate all non-linear connections in the neural network and check whether the specification holds on the over-approximation. The benefit is that reasoning about the (in this case linear) over-approximation is typically computationally easier than reasoning about the concrete representation, and moreover, if the specification holds on the abstraction, then it actually holds on the original network.

Previous work has studied different ways to over-approximate S-shaped activations, and there are trade-offs between how *precise* the over-approximation is and how *efficient* it is to reason about the over-approximation. For example, a piecewise-linear over-approximation can be more precise than the linear over-approximation (in Fig. 3), but reasoning about the former is more computationally challenging. However, whichever over-approximation one uses, *all* abstraction-based methods suffer from imprecision: if a counter-example to the specification is found on the over-approximation (which means the over-approximation violates the specification), we cannot conclude that the original network also violates the specification. This is because the counter-example may be spurious—inconsistent with the constraints imposed by the precise, unabstracted neural network (as shown in red in Fig. 3).

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

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

```
function VNN-CEGAR(M := ⟨V, X, Y, φ_aff, φ_ρ⟩, Φ)
    M' ← Abstract(M)
    while true do
        ⟨α, proven⟩ ← Prove(M', Φ)          ▷ Try to prove.
        if proven then return true          ▷ Property proved.
        ⟨M', refined⟩ ← Refine(M', Φ, α)    ▷ Try to refine.
        if ¬refined then return false       ▷ Counter-example is real.
```

refinement step: performing case analysis on the activation phases. However, when dealing with S-shaped 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 challenge, we propose a counter-example guided refinement strategy for S-shaped activation functions which is based on the CEGAR approach [69]. 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).

IV. A CEGAR FRAMEWORK FOR S-SHAPED ACTIVATIONS

In this section, we formalize our meta-algorithm for precisely reasoning about DNNs with S-shaped activations, which is based on the CEGAR framework [70]. 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 $\Phi := (\Phi_{in}(\mathcal{X}) \Rightarrow \Phi_{out}(\mathcal{Y}))$, where $\Phi_{in}(\mathcal{X})$ and $\Phi_{out}(\mathcal{Y})$ are linear arithmetic formulas over $\mathcal{X}$ and $\mathcal{Y}$, and we say that Φ holds on M if and only if the formula

$$\psi := \phi_{\text{aff}} \wedge \phi_\rho \wedge \Phi_{in}(\mathcal{X}) \wedge \neg \Phi_{out}(\mathcal{Y})$$

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 IV.1 (Sound abstraction)**.** *Given a tuple $M := \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \phi_{\text{aff}}, \phi_\rho \rangle$ and a property $\Phi = (\Phi_{in}(X) \Rightarrow \Phi_{out}(Y))$, 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 $\hat{\mathbf{v}}^{(\mathbf{i})}$ be a similar vector representing the post-activation values in layer i. Let $\hat{\mathbf{v}}^{(\mathbf{0})}$ 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 $\hat{\mathbf{v}}^{(\mathbf{0})}$, 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. IV.1:

Theorem IV.2 (CEGAR is sound)**.** *Algorithm 1 returns true only if $M \models \Phi$.*

A. 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 [14, 31, 44, 52]. 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 S-shaped activations, we use piecewise-linear over-approximations. In principle, any sound piecewise-linear over-approximation of the S-shaped function could be used. One approach is to use a fine-grained over-approximation with piecewise-linear bounds [71]. 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 S-shaped function include the abstraction proposed in [18, 31, 34, 52, 67], which considers the convex relaxation of the S-shaped activation.

B. Abstraction Refinement for the S-shaped activation function

We now focus on the problem of abstraction refinement for models with S-shaped activation functions. Suppose that an assignment α is found by Prove such that $M'[\alpha] \models \neg\Phi$, but for some neuron with S-shaped 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 the S-shaped 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 an S-shaped function. In practice, bounds can be computed with bound-propagation techniques [31, 34, 35].

Lemma IV.3. *Given an interval (l, u), an S-shaped 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 $\mathrm{h} : \mathbb{R} \mapsto \mathbb{R}$ 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 S-shaped function. If $q > \rho(p)$, we add the formula $\hat{v} \leq \mathrm{h}(v)$ to the abstraction. And if $q < \rho(p)$, we add $\hat{v} \geq \mathrm{h}(v)$.

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. I presents a general recipe for choosing β and γ when the spurious counter-example point is below the S-shaped function. Choosing β and γ when the counter-example is above the S-shaped function is symmetric (details are shown in App. A). η denotes the inflection point of the S-shaped 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 [34, 52] could be applied to compute better slopes, but the formulas shown give the best approximations we could find without using search.

Algorithm 2 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** $(v, \hat{v}) \in$ AllSigmoidal($\mathcal{V}'$) **do**
3: **if** $\alpha(\hat{v}) = \rho(\alpha(v))$ **then continue** ▷ Skip satisfied activations.
4: *refined* $\leftarrow$ *refined* $+ 1$
5: $\beta, \gamma \leftarrow$ getSlopes$(l(v), u(v), \alpha(v), \alpha(\hat{v}))$ ▷ Compute slopes.
6: $\phi'_{\rho} \leftarrow \phi'_{\rho} \cup$ addPLBound$(\beta, \gamma, v, \hat{v}, \alpha)$ ▷ Refine the abstraction.
7: **if** stopCondition(*refined*) **then break** ▷ Check termination condition.
8: **return** $\langle \mathcal{V}', \mathcal{X}, \mathcal{Y}, \phi'_{\text{aff}}, \phi'_{\rho} \rangle$, *refined* > 0

Lemma IV.4 (Soundness of slopes)**.** *Choosing β and γ using the recipe shown in Table I results in a piecewise-linear function h that satisfies the conditions of Lemma IV.3.*

An instantiation of the Refine function for neural networks with S-shaped activation is shown in Alg. 2. It iterates through each S-shaped 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., $\hat{v} \geq \mathrm{h}(v)$) 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 that performed best in an empirical analysis.

Theorem IV.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. $M''[\alpha] \models \Phi$.*

V. Experimental evaluation

In this section, we evaluate the performance of our proposed verification framework. We begin by investigating the effectiveness of our CEGAR-based approach on boosting the verification accuracy of existing approaches based on one-shot over-approximation. We evaluate on both robustness queries on real-world distribution shifts (§V-A) and existing benchmarks on robustness against norm-bounded perturbations [67, 72] (§V-C). Next, we benchmark the performance of our verifier on a range of challenging distribution shifts (§ V-D). Finally, we use our method to show that robust training tends to result in higher levels of certified robustness against distribution shifts (§ V-E).

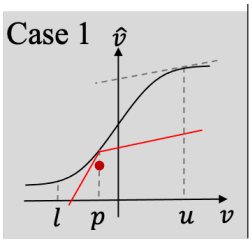
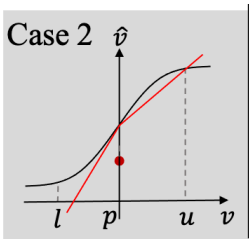
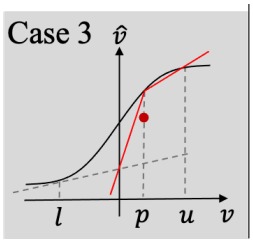
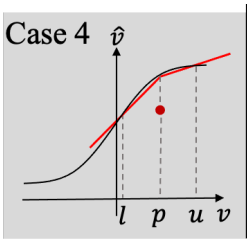
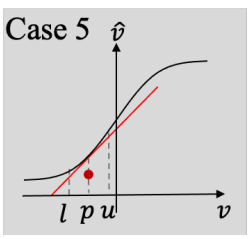
	Case 1 	Case 2 	Case 3 	Case 4 	Case 5 
	$l < \eta, u > \eta$ $\rho''(p) > 0$	$l < \eta, u > \eta$ $\rho''(p) = 0$	$l < \eta, u > \eta$ $\rho''(p) < 0$	$l > \eta \vee u < \eta$ $\rho''(p) \leq 0$	$l > \eta \vee u < \eta$ $\rho''(p) > 0$
β	$\rho'(p)$	$\rho'(p)$	$\frac{\rho(l)+\rho'(l)(\eta-l)-\rho(p)}{\eta-p}$	$\frac{\rho(p)-\rho(l)}{p-l}$	$\rho'(p)$
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 I: Slopes for the piece-wise linear abstraction refinement. The figures illustrate the refinement on the sigmoid function.

Datasets for distribution shifts. We consider a diverse array of distribution shifts on the MNIST [73] and CIFAR-10 [74] datasets. The code used to generate the perturbations is adapted from [10].

Training algorithms. For each distribution shift we consider, we train a CVAE using the framework outlined in [9]. All generators use sigmoid activations in the output layer and ReLU activations in the hidden layers. This is a typical architecture for generative models. For each dataset, the number of sigmoid activations used in the CVAE is the same as the (flattened) output dimension; that is, 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) [75], invariant risk minimization (IRM) [76], projected gradient descent (PGD) [2], and model-based dataset augmentation (MDA) [8].

Implementation details. We use the DeepPoly/CROWN [31, 34] 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 verification tool [14].[2] 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.

A. Evaluation of CEGAR-based verification procedure

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/CROWN 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/CROWN linear bounds and the other parts are precisely encoded; and (3) `DP+CEGAR`, which is the CEGAR method proposed in this work.[3] 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 II, 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 `DP+CEGAR`, as well as the average number of abstraction refinement rounds by `DP+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 counter-example guided abstraction refinement scheme can successfully boost the precision when reasoning about sigmoid activations by leveraging existing verifiers.

B. Effect of hyper-parameters m and k

The refinement procedure (Alg. 2) has two numerical hyper-parameters, m and k, which together control the number of new bounds introduced in a given refinement round (at round i, at most $m \times k^i$ new bounds are introduced). If too few new bounds are introduced, then there is not enough refinement, and the number of refinement rounds needed to prove the property increases. On the other hand, if too many bounds are

[2]We note that our framework is general and can be used with other abstractions and solvers. To motivate more efficient ways to encode and check the refined abstraction, we describe in App. C how to encode the added piecewise-linear bounds as LeakyReLUs. This leaves open the possibility of further leveraging neural network verifiers that support LeakyReLUs.

[3]We 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. F

Dataset	Gen.	Class.	DP	DP+BaB		DP+CEGAR		
			δ	δ	time(s)	δ	time(s)	# ref.
MNIST	MLP_GEN$_1$	MLP_CLS$_1$	0.104 ± 0.041	0.139 ± 0.058	7.8	$\mathbf{0.157} \pm 0.057$	84.1	1.5 ± 1.1
	MLP_GEN$_2$	MLP_CLS$_1$	0.08 ± 0.035	0.106 ± 0.049	20.4	$\mathbf{0.118} \pm 0.049$	114.8	1.0 ± 1.1
	MLP_GEN$_1$	MLP_CLS$_2$	0.102 ± 0.044	0.136 ± 0.061	16.4	$\mathbf{0.15} \pm 0.059$	120.6	1.2 ± 1.2
	MLP_GEN$_2$	MLP_CLS$_2$	0.081 ± 0.037	0.112 ± 0.049	60.8	$\mathbf{0.121} \pm 0.049$	191.6	0.8 ± 1.1
	MLP_GEN$_1$	MLP_CLS$_3$	0.099 ± 0.041	0.135 ± 0.062	41.3	$\mathbf{0.146} \pm 0.059$	186.9	1.0 ± 1.1
	MLP_GEN$_2$	MLP_CLS$_3$	0.082 ± 0.036	0.116 ± 0.044	75.7	$\mathbf{0.122} \pm 0.041$	163.3	0.6 ± 1.0
CIFAR	CONV_GEN$_1$	CONV_CLS$_1$	0.219 ± 0.112	0.273 ± 0.153	33.5	$\mathbf{0.287} \pm 0.148$	140.8	4.5 ± 9.2
	CONV_GEN$_2$	CONV_CLS$_1$	0.131 ± 0.094	0.18 ± 0.117	13.7	$\mathbf{0.194} \pm 0.115$	112.5	3.1 ± 6.0
	CONV_GEN$_1$	CONV_CLS$_2$	0.176 ± 0.108	0.242 ± 0.14	16.0	$\mathbf{0.253} \pm 0.136$	57.7	1.6 ± 2.4
	CONV_GEN$_2$	CONV_CLS$_2$	0.12 ± 0.077	0.154 ± 0.087	7.9	$\mathbf{0.172} \pm 0.085$	140.2	3.3 ± 4.2

TABLE II: Evaluation results of three solver configurations.

introduced, then the time required by the solver to check the abstraction might unnecessarily increase, resulting in timeouts.

To study the effect of m and k more closely, we evaluate the runtime performance of DP+CEGAR instantiated with different combinations of m and k. In particular, we run all combinations of $m \in \{10, 30, 50\}$ and $k \in \{1.5, 2\}$ on the first 10 verification instances for the first two generator-classifier pairs in Table II–$(\text{MLP_GEN}_1, \text{MLP_CLS}_1)$ and $(\text{CONV_GEN}_1, \text{CONV_CLS}_1)$. The perturbation bound is the largest δ proven robust for each instance.

Dataset	m	k	# solved	time(s)	# ref.
MNIST	10	1.5	10	139.5	2.1
	30	1.5	8	52.4	0.9
	50	1.5	7	39.8	0.9
	10	2	10	113.6	1.8
	30	2	10	121.0	1.1
	50	2	9	64.6	0.9
CIFAR	10	1.5	9	77.0	2.3
	30	1.5	9	60.4	1.5
	50	1.5	9	38.2	1.4
	10	2	9	55.3	2.4
	30	2	10	100.7	2.2
	50	2	10	82.2	1.6

TABLE III: Effect of m and k on the runtime performance of CEGAR.

Table III shows the number of solved instances, the average runtime on solved instances, and the average number of refinement rounds on solved instances. For the same value of k, the number of refinement rounds on solved instances consistently decreases as m increases. This is expected, because refinements are performed more eagerly as m increases. However, the decrease in the number of refinement rounds does not necessarily imply improvements in performance. For example, $(50, 2)$ solves one fewer MNIST benchmark than $(30, 2)$. On the other hand, if the strategy is too "lazy" (e.g., m is too small), the increased number of refinement rounds can also result in runtime overhead. For example, on the CIFAR10 benchmarks, when $k = 1.5$, the average runtime decreases as m increases, even though all three configurations solve the same number of instances.

Overall, this study suggests that the optimal values of m and k vary across different benchmarks, and exploring adaptive heuristics for choosing these hyper-parameters is a promising direction for boosting the runtime performance of the proposed algorithm.

C. Further evaluation of CEGAR on adversarial robustness benchmarks.

To better understand the effectiveness of our abstraction-refinement technique on boosting the verification accuracy over one-shot approximation, we consider a different initial abstraction, PRIMA [67], a more recently proposed abstraction that considers convex relaxations over groups of activation functions and is empirically more precise than DeepPoly/CROWN. In particular, we focus on the *same* sigmoid benchmarks used in [67]. We use the PRIMA implementation in the artifact associated with the paper[4] and run a configuration PRIMA+CEGAR which is the same as DP+CEGAR, except that we run PRIMA instead of DeepPoly/CROWN for the initial abstraction. Each job is given 16 threads and a 30 minute wall-clock time limit. Table IV shows the number of verified instances and the average runtime on verified instances for the two configurations. Our configuration is able to consistently solve more instances with only a moderate increase in 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.

TABLE IV: Comparison with PRIMA [67] on the same benchmarks used in [67]

Model	Acc.	ϵ	PRIMA		PRIMA+CEGAR	
			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

We have also evaluated our techniques on the sigmoid benchmarks used in VNN-COMP 2021 (there are no sigomid benchmarks in VNN-COMP 2022), and we are also able to boost the verification precision over other SoTA solvers such as $\alpha-\beta$-CROWN and VeriNet [52] with our approach. Details can be found 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

Fig. 4: Robustness of ERM against different perturbations.

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

Fig. 5: Test set accuracy and verification accuracy

D. 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_GEN_2 and MLP_CLS_3. For CIFAR-10 we use CONV_GEN_2 and CONV_CLS_2. The verification results are shown in Figure 4. 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.

E. 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, 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_GEN_2 and MLP_CLS_3.

Accuracy and robustness results are presented in Figure 5. 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.

[4]https://dl.acm.org/do/10.1145/3462308/full/

VI. 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 [54–56]. 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 [59]. Our work differs from [59] in two ways. Firstly, Xie et al. use classifiers and regressors in their neural specifications, while we use generative models to express perturbation sets in our specifications. Secondly, while Xie et al. make black-box use of existing verifiers, we propose a new verification scheme for analyzing networks with sigmoid activations. In general, we believe the idea of leveraging neural components for specification is very promising and should be explored further.

Existing verification approaches. Existing DNN verification algorithms broadly fall into one of two categories: search-based methods [13–30] and abstraction-based methods [31–51]. While several existing solvers can handle sigmoid activation functions [18, 31, 34, 52, 67], 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 [77], 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 [69] that leverages existing verifiers to solve increasingly refined abstractions. We notice a concurrent work [78] on formal reasoning of sigmoidal activations which is made available a week before the deadline. The technique is orthogonal to our approach as it again performs one-shot over-approximation of sigmoidal activations.

CEGAR [69, 79] is a well-known technique. Our contribution in this area lies in investigating the choices of its parameters (i.e., Abstract, Prove, and Refine) when it comes to verifying neural networks with S-shaped activations.

Verification against distribution shifts. The authors of [9] also considered the task of evaluating DNN robustness to real-world distribution shifts; in particular, the approach used

in [9] relies on randomized smoothing [80]. 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 [57, 58]. However, each of these works assumes that generative models are piece-wise linear functions, which precludes the use of state-of-the-art models.

VII. 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 [81] and DenseNet [82]. This stems from the limited capacity of existing DNN verification tools for piecewise-linear functions, which we invoke in the CEGAR 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 [9] 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. In general, we believe that closing this verification loop (verify, debug, verify again, etc.) is a very interesting future research direction. Finally, when applying our techniques in real industrial settings, another challenge is that collecting training data corresponding to new distribution shifts may be costly (e.g., collecting the same street view under different times of day). However, this cost may be justified in safety-critical domains.

References

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

[2] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[3] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5286–5295.

[4] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *International conference on machine learning*. PMLR, 2019, pp. 7472–7482.

[5] H. Kannan, A. Kurakin, and I. Goodfellow, "Adversarial logit pairing," *arXiv preprint arXiv:1803.06373*, 2018.

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[7] A. Robey, L. Chamon, G. J. Pappas, H. Hassani, and A. Ribeiro, "Adversarial robustness with semi-infinite constrained learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6198–6215, 2021.

[8] A. Robey, H. Hassani, and G. J. Pappas, "Model-based robust deep learning," *arXiv preprint arXiv:2005.10247*, 2020.

[9] E. Wong and J. Z. Kolter, "Learning perturbation sets for robust machine learning," *arXiv preprint arXiv:2007.08450*, 2020.

[10] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[11] D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo, R. Desai, T. Zhu, S. Parajuli, M. Guo *et al.*, "The many faces of robustness: A critical analysis of out-of-distribution generalization," *arXiv preprint arXiv:2006.16241*, 2020.

[12] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao *et al.*, "Wilds: A benchmark of in-the-wild distribution shifts," *arXiv preprint arXiv:2012.07421*, 2020.

[13] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, 2017, pp. 97–117.

[14] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*, 2019, pp. 443–452.

[15] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[16] A. De Palma, H. S. Behl, R. Bunel, P. H. Torr, and M. P. Kumar, "Scaling the convex barrier with active sets," *arXiv preprint arXiv:2101.05844*, 2021.

[17] R. Bunel, J. Lu, I. Turkaslan, P. Kohli, P. Torr, and P. Mudigonda, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learn-*

ing Research, vol. 21, no. 2020, 2020.

[18] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," *arXiv preprint arXiv:2011.13824*, 2020.

[19] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," *CoRR*, vol. abs/1705.01320, 2017. [Online]. Available: http://arxiv.org/abs/1705.01320

[20] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3291–3299.

[21] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri, "Optimization and abstraction: A synergistic approach for analyzing neural network robustness," in *Proc. Programming Language Design and Implementation (PLDI)*, 2019, p. 731–744.

[22] H. Khedr, J. Ferlez, and Y. Shoukry, "Peregrinn: Penalized-relaxation greedy neural network verifier," *arXiv preprint arXiv:2006.10864*, 2020.

[23] M. Fischetti and J. Jo, "Deep neural networks as 0-1 mixed integer linear programs: A feasibility study," *CoRR*, vol. abs/1712.06174, 2017.

[24] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. Torr, and M. P. Kumar, "Lagrangian decomposition for neural network verification," in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 370–379.

[25] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks." in *UAI*, vol. 1, no. 2, 2018, p. 3.

[26] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli, "Efficient neural network verification with exactness characterization," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 497–507.

[27] H. Wu, A. Zeljić, G. Katz, and C. Barrett, "Efficient neural network analysis with sum-of-infeasibilities," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022, pp. 143–163.

[28] C. Ferrari, M. N. Muller, N. Jovanovic, and M. Vechev, "Complete verification via multi-neuron relaxation guided branch-and-bound," *arXiv preprint arXiv:2205.00263*, 2022.

[29] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 18–42.

[30] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV*, 2017.

[31] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[32] G. Singh, R. Ganvir, M. Püschel, and M. Vechev, "Beyond the single neuron convex barrier for neural network certification," *Advances in Neural Information Processing Systems*, vol. 32, pp. 15 098–15 109, 2019.

[33] Z. Lyu, C.-Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel, "Fastened crown: Tightened neural network robustness certificates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5037–5044.

[34] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *arXiv preprint arXiv:1811.00866*, 2018.

[35] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, 2018, pp. 1599–1614. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi

[36] ——, "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*, 2018, pp. 6369–6379. [Online]. Available: http://papers.nips.cc/paper/7873-efficient-formal-safety-analysis-of-neural-networks

[37] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "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.

[38] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5276–5285.

[39] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," *arXiv preprint arXiv:1902.08722*, 2019.

[40] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. Patel, and J. P. Vielma, "The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification," *arXiv preprint arXiv:2006.14076*, 2020.

[41] A. Raghunathan, J. Steinhardt, and P. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," *arXiv preprint arXiv:1811.01057*, 2018.

[42] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[43] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "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*, 2018, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2018.00058

[44] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," *arXiv preprint arXiv:2103.06624*, 2021.

[45] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International Conference on Learning Representations*, 2019.

[46] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3240–3247.

[47] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in Neural Information Processing Systems*, vol. 31, pp. 10 802–10 813, 2018.

[48] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev, "Prima: general and precise neural network certification via scalable convex hull approximations," *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 1–33, 2022.

[49] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 43–65.

[50] W. Ryou, J. Chen, M. Balunovic, G. Singh, A. Dan, and M. Vechev, "Scalable polyhedral verification of recurrent neural networks," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 225–248.

[51] H. Wu, C. Barrett, M. Sharif, N. Narodytska, and G. Singh, "Scalable verification of gnn-based job schedulers," *arXiv preprint arXiv:2203.03153*, 2022.

[52] P. Henriksen and A. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *ECAI 2020*. IOS Press, 2020, pp. 2513–2520.

[53] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[54] M. Balunović, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying geometric robustness of neural networks," *Advances in Neural Information Processing Systems 32*, 2019.

[55] C. Paterson, H. Wu, J. Grese, R. Calinescu, C. S. Pasareanu, and C. Barrett, "Deepcert: Verification of contextually relevant robustness for neural network image classifiers," *arXiv preprint arXiv:2103.01629*, 2021.

[56] J. Mohapatra, P.-Y. Chen, S. Liu, L. Daniel *et al.*, "Towards verifying robustness of neural networks against semantic perturbations," *arXiv preprint arXiv:1912.09533*, 2019.

[57] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, "Verification of image-based neural network controllers using generative models," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–10.

[58] M. Mirman, A. Hägele, P. Bielik, T. Gehr, and M. Vechev, "Robustness certification with generative models," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 1141–1154.

[59] X. Xie, K. Kersting, and D. Neider, "Neuro-symbolic verification of deep neural networks," 2022.

[60] S. Gowal, C. Qin, P.-S. Huang, T. Cemgil, K. Dvijotham, T. Mann, and P. Kohli, "Achieving robustness in the wild via adversarial mixing with disentangled representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1211–1220.

[61] A. Robey, G. J. Pappas, and H. Hassani, "Model-based domain generalization," *arXiv preprint arXiv:2102.11436*, 2021.

[62] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[63] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[64] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[65] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[66] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 172–189.

[67] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev, "Precise multi-neuron abstractions for neural network certification," *arXiv preprint arXiv:2103.03638*, 2021.

[68] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1129–1141, 2020.

[69] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*. Springer, 2000, pp. 154–169.

[70] M. Mann, A. Irfan, A. Griggio, O. Padon, and C. Barrett, "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*. Springer, 2021, pp. 113–132.

[71] C. Sidrane, A. Maleki, A. Irfan, and M. J. Kochenderfer, "Overt: An algorithm for safety verification of neural network control policies for nonlinear systems," *Journal of Machine Learning Research*, vol. 23, no. 117, pp. 1–45, 2022.

[72] "International verification of neural networks competition (vnn-comp)," 2020, https://sites.google.com/view/vnn20/vnncomp.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[74] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[75] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1999.

[76] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," *arXiv preprint arXiv:1907.02893*, 2019.

[77] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "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*, 2019, pp. 169–178.

[78] Z. Zhang, Y. Wu, S. Liu, J. Liu, and M. Zhang, "Provably tightest linear approximation for robustness verification of sigmoid-like neural networks," *arXiv preprint arXiv:2208.09872*, 2022.

[79] A. Cimatti, A. Griggio, A. Irfan, M. Roveri, and R. Sebastiani, "Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions," *ACM Transactions on Computational Logic (TOCL)*, vol. 19, no. 3, pp. 1–52, 2018.

[80] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1310–1320.

[81] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[82] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

APPENDIX A
CHOICES OF SLOPES (CONT.)

We present in Table V the general recipe for choosing β and γ in the case when the violation point is above the S-shaped function.

APPENDIX B
PROOFS

Proof. **Theorem IV.2**. Alg. 1 returns true only if the property holds on a sound abstraction of M, which following Def. IV.1 means the property holds on M. □

Proof. **Lemma IV.3**. This can be proved by construction using the β and γ values in Table I and Table V. We next prove that those choices are sound in Lemma IV.4. □

Before proving Lemma IV.4, we first state the following definitions and facts. [5]

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

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 by $\texttt{SecLine}_{f,a,b}(x)$, is defined as: $\texttt{SecLine}_{f,a,b}(x) = \frac{f(a)-f(b)}{a-b} * (x - a) + f(a)$.*

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 $a, x \in [l, u]$, $\texttt{TanLine}_{f,a}(x) \leq f(x)$, and for all $a, b, x \in [l, u]$, where $a < b$ and $a \leq x \leq b$, $\texttt{SecLine}_{f,a,b}(x) \geq f(x)$.*

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 $a, x \in [l, u]$, $\texttt{TanLine}_{f,a}(x) \geq f(x)$, and for all $a, b, x \in [l, u]$, where $a < b$ and $a \leq x \leq b$, $\texttt{SecLine}_{f,a,b}(x) \leq f(x)$.*

Proposition B.5. *Let f be a univariate function, differentiable with non-negative derivative on $[l, u]$. 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 IV.4**. Cond. 1 and Cond. 2 hold trivially. Since $q < h(p)$, for Cond. 3, it suffices to show that whenever $x \in (l, u)$, $\rho(x) \geq h(x)$. More concretely, we show that (a) $\rho(x) \geq \rho(p) + \beta(x - p)$ for $x \in [l, p]$, and (b) $\rho(x) \geq \rho(p) + \gamma(x - p)$ for $x \in (p, u]$. We prove this is true for each case in Table. I.

- **Case 1**: The segment coresponding to β is $\texttt{TanLine}_{\rho,p}$, 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 coresponding to β is $\texttt{TanLine}_{\rho,p}$, so Cond. (a) holds by Prop. B.3. The segment corresponding to γ is $\texttt{SecLine}_{\rho,p,u}$, so Cond. (b) holds by Prop. B.4.
- **Case 3**: For Cond. (a), we further break it into 2 cases: $x \leq \eta$ and $x > \eta$. In the former case, the line $\rho(p) + \beta(x - p)$ is below the line $\rho(l) + \min(\rho'(l), \rho'(u))(x - l)$, which

[5] These are partially adapted from [79].

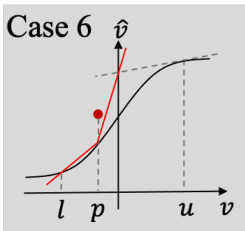
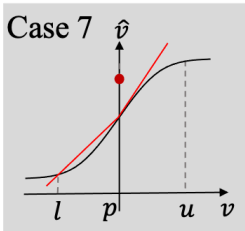
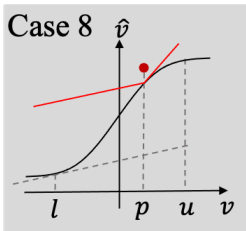
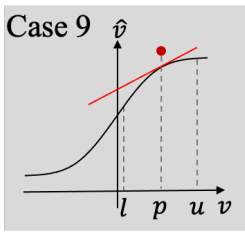
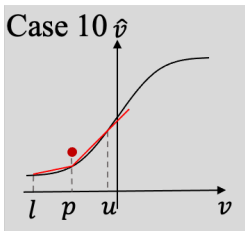
					
	$l < \eta, u > \eta$ $\rho''(p) > 0$	$l < \eta, u > \eta$ $\rho''(p) = 0$	$l < \eta, u > \eta$ $\rho''(p) < 0$	$l > \eta \vee u < \eta$ $\rho''(p) \leq 0$	$l > \eta \vee u < \eta$ $\rho''(p) > 0$
β	$\frac{\rho(p)-\rho(l)}{p-l}$	$\frac{\rho(p)-\rho(l)}{p-l}$	$\min(\rho'(l), \rho'(u))$	$\rho'(p)$	$\frac{\rho(p)-\rho(l)}{p-l}$
γ	$\frac{\rho(u)-\rho'(u)(u-\eta)-\rho(p)}{\eta-p}$	$\rho'(p)$	$\rho'(p)$	$\rho'(p)$	$\frac{\rho(p)-\rho(u)}{p-u}$

TABLE V: Slopes for the piece-wise linear abstraction refinement.

by Prop. B.5 is below ρ. When $x > \eta$, $\rho(p) + \beta(x-p)$ is below the secant line $\texttt{SecLine}_{\rho,\eta,p}$, which by Prop. B.4 is below ρ. On the other hand, the segment corresponding to γ is $\texttt{SecLine}_{\rho,p,u}$, so Cond. (b) holds by Prop. B.4.

- **Case 4**: The segments are both secant lines, $\texttt{SecLine}_{\rho,l,p}$ and $\texttt{SecLine}_{\rho,p,u}$, and thus the conditions hold by Prop. B.4.
- **Case 5**: The segments are both tangent lines, $\texttt{TanLine}_{\rho,p}$, and thus the conditions hold by Prop. B.3.

The proof for the cases shown in Table. V is analogous. □

Proof. **Theorem IV.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 IV.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, $M''[\alpha] \models \Phi$. □

APPENDIX 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 in this paper, we lay out the reduction to LeakyReLU in this section to show how future work usingverification tools supporting LeakyReLUs could benefit.

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 (i.e., $\alpha \geq 0$) because we always choose both β and γ to be 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

$$a_1 = x - p \tag{10a}$$
$$a_2 = r_\alpha(a_1) \tag{10b}$$
$$a_3 = a_2 + \rho(p) \tag{10c}$$
$$y = a_3 + a_4 \tag{10d}$$
$$a_4 \leq 0, \tag{10e}$$

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 suggests that we could, in principle, express the abstraction as an actual piecewise-linear neural network (with bounds on the input variables (e.g., a_4), making it possible to leverage verifiers built on top of neural network software platforms such as Tensorflow or Pytorch.

APPENDIX D
DETAILS ON TRAINING AND CVAES

A. Dataset

We consider the well-known MNIST and CIFAR-10 datasets. 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 [10].[6] When training and testing the models, we

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

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 VI.

TABLE VI: 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	$[.4, 1]$

B. Architecture

On each dataset, we train a conditional variational encoder (CVAE) with three components: prior network, encoder network, and decoder network (generator). We also train a set of classifiers. In this section, we detail the architecture of these networks. The architectures for the MNIST networks are shown in Tables VII–XIII. Those of the CIFAR networks are shown in Tables XIV–XIX. The output layers of the generators use sigmoid activation functions. All hidden-layers use ReLU activation functions.

TABLE VII: Prior

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

TABLE VIII: Encoder

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

TABLE IX: MLP_GEN$_1$

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

TABLE X: MLP_GEN$_2$

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

TABLE XI: MLP_CLS$_1$

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

TABLE XII: MLP_CLS$_2$

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

TABLE XIII: MLP_CLS$_3$

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

TABLE XIV: Prior

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

TABLE XV: Encoder

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

TABLE XVI: CONV_GEN$_1$

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

TABLE XVII: CONV_GEN$_2$

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

TABLE XVIII: CONV_CLS$_1$

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

TABLE XIX: CONV_CLS$_2$

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

C. Optimization

We implement our models and training in PyTorch. The CVAE implementation is adapted from that in [9]. 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 [3], 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. V-E are 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, we use a step size of $\alpha = 0.1$, a perturbation budget of $\epsilon = 1.0$, and we use 10 steps of projected gradient ascent.

D. Computing resources

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

Appendix E
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 the 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 [18, 34, 44] (competition version), and if the problem is not solved, we run `PRIMA+CEGAR`. Table IV shows the results. As a point of comparison, we also report the numbers of the top three performing tools [18, 31, 44, 52, 67] during VNN-COMP-21 on these benchmarks. [7] 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 exhaust the 5 minute per-instance timeout on any of these benchmarks.[8] 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 one-shot verification attempt fails.

Appendix F
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, each with K linear segments, for each sigmoid activation function. The segment points are evenly distributed along the x-axis. We evaluate on the same MNIST benchmarks as in Table II, using $K = 2$ and $K = 3$. The results are shown in Table XXI. 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 values for the eager approach 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 former only improves on one of the six benchmark sets. This suggests that the finer-grained abstraction increases the overhead to the solver and is not particularly effective at excluding spurious counter-examples on the set of benchmarks that we consider, which supports the need for a more informed abstraction refinement strategy such as the one we propose.

Appendix G
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).

[7] https://arxiv.org/abs/2109.00498

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

TABLE XX: 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

Dataset	Gen.	Class.	K=2		K=3		DP+CEGAR		
			δ	time(s)	δ	time(s)	δ	time(s)	# ref.
MNIST	MLP_GEN_1	MLP_CLS_1	0.137 ± 0.043	88.9	0.137 ± 0.042	109.8	$\mathbf{0.157} \pm 0.057$	84.1	1.5 ± 1.1
	MLP_GEN_2	MLP_CLS_1	0.109 ± 0.031	114.5	0.109 ± 0.031	199.0	$\mathbf{0.118} \pm 0.049$	114.8	1.0 ± 1.1
	MLP_GEN_1	MLP_CLS_2	0.126 ± 0.045	64.0	0.129 ± 0.044	95.9	$\mathbf{0.15} \pm 0.059$	120.6	1.2 ± 1.2
	MLP_GEN_2	MLP_CLS_2	0.108 ± 0.038	159.0	0.106 ± 0.036	133.8	$\mathbf{0.121} \pm 0.049$	191.6	0.8 ± 1.1
	MLP_GEN_1	MLP_CLS_3	0.132 ± 0.043	139.5	0.131 ± 0.042	190.0	$\mathbf{0.146} \pm 0.059$	186.9	1.0 ± 1.1
	MLP_GEN_2	MLP_CLS_3	0.105 ± 0.033	107.1	0.098 ± 0.035	87.5	$\mathbf{0.122} \pm 0.041$	163.3	0.6 ± 1.0

TABLE XXI: Evaluation results of the eager approach. We also report again the results of DP+CEGAR, which is the same as Table II.