Exact and Efficient Adversarial Robustness with Decomposable Neural Networks

Abstract

As deep neural networks are notoriously vulnerable to adversarial attacks, there has been significant interest in defenses with provable guarantees. Recent solutions advocate for a randomized smoothing approach to provide probabilistic guarantees, by estimating the expectation of a network's output when the input is randomly perturbed. As the convergence of the estimated expectations depends on the number of Monte Carlo samples, and hence network evaluations, these techniques come at the price of considerable additional computation at inference time. We take a different route and introduce a novel class of deep models—decomposable neural networks (DecoNets)—which compute the required expectation *exactly* and efficiently using a *single network evaluation*. This remarkable feature of DecoNets stems from their network structure, implementing a hierarchy of *decomposable multiplicative interactions* over non-linear input features, which allows to reduce the overall expectation into many "small" expectations over input units, thus delivering *exact guarantees*.

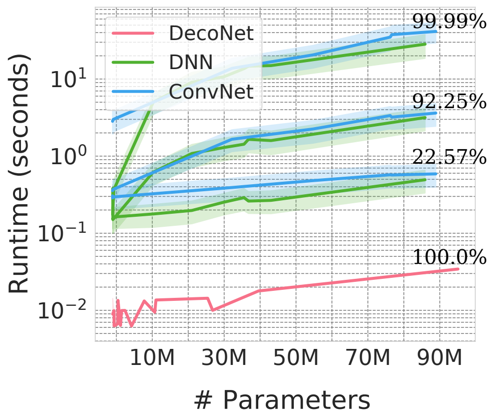

Figure 1: Scaling certified accuracy computation for $\sigma = 1$ and $R = 2\sigma$ (time in seconds, y-axis) for DecoNets, DNNs and ConvNets of equivalent number of parameters (x-axis). The Monte-Carlo estimate for DNNs and ConvNets is done for different confidence levels $(99.99, 92.25, 22.577)$ requiring respectively $10^4, 10^3, 10^2$ samples, evaluated in a single batch on a single GPU. DecoNets are able to obtain a non-probabilistic guarantee (100% confidence) in a single network evaluation. See Sec. 5 for details.

1 INTRODUCTION

Deep learning has revolutionized machine learning (ML) and artificial intelligence (AI) over the past decade and has seen a wide adoption in sensitive domains such as autonomous driving and decision support systems in hiring, jurisdiction, and medical diagnosis. Recent years, however, have also highlighted that deep neural networks (DNNs) are vulnerable to *adversarial attacks,*—small engineered input perturbations that lead to a dramatic change in the DNN's output [Szegedy et al., 2014, Goodfellow et al., 2015]—which makes their usage in such domains problematic.

This vulnerability has motivated the design of *adversarial defenses,* i.e., training and inference techniques which aim to protect a model from adversarial attacks. Many recent works focus on *certified* defenses, which allow one to prove the non-existence of adversarial examples in a region of interest, no matter what adversarial attack is employed. One of the most promising techniques in this area is *randomized smoothing* [Lecuyer et al., 2019, Cohen et al., 2019, Salman et al., 2019a]. This approach transforms a classifier f (without robustness guarantees) into a certified *smoothed* classifier g, defined by Cohen et al. [2019] as $g(\mathbf{x}) := \arg\max_c \mathbb{P}[f(\mathbf{x} + \boldsymbol{\epsilon}) = c]$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma\mathbf{I})$ is isotropic Gaussian noise. That is, the input $\mathbf{x}$ is assigned the class which is predicted with highest probability by f when the input $\mathbf{x}$ is perturbed with noise. The smoothed classified g guarantees a constant prediction in a certain ℓ_2-ball and hence the non-existence of an adversarial example in such a ball. More formally, let $p_A = \mathbb{P}[f(\mathbf{x}+\boldsymbol{\epsilon}) = g(\mathbf{x})]$ be the probability of the most likely class, and $p_B = \max_{c \neq g(\mathbf{x})} \mathbb{P}[f(\mathbf{x}+\boldsymbol{\epsilon}) = c]$, the probability of the runner-up class, then

$$g(\mathbf{x}+\boldsymbol{\delta}) = g(\mathbf{x}) \quad \forall \|\boldsymbol{\delta}\|_2 \leq R := \frac{\sigma}{2}(\Phi^{-1}(p_A) - \Phi^{-1}(p_B)), \tag{1}$$

where Φ is the standard Gaussian CDF. In Salman et al. [2019a], this result is generalized using guarantees on the Lipschitz constant of bounded, soft classifier functions when computing not p_A but their expected outputs, that is when *smoothing* them.

Definition 1.1 (Smoothed function)**.** Let $f\colon \mathcal{X} \mapsto \mathbb{R}^C$ be a C-output function. Its *smoothed* version $\bar{f}_\sigma$ is defined as $\bar{f}_\sigma(\mathbf{x}) = (\bar{f}_{\sigma,1}(\mathbf{x}), \ldots, \bar{f}_{\sigma,C}(\mathbf{x}))^\top$, where

$$\bar{f}_{\sigma,c}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right] \tag{2}$$

and $\boldsymbol{\epsilon}$ is zero-mean Gaussian noise with covariance $\sigma\mathbf{I}$.

As shown in Salman et al. [2019a], $\bar{f}$, the smoothed version of a classifier $f\colon \mathcal{X} \mapsto [0,1]^C$, essentially realizes a low-pass filtered version of f using a Gaussian filter and hence is $\sqrt{\frac{2}{\pi\sigma^2}}$ Lipschitz, leading to an immediate robustness guarantee within a certain ℓ_2-ball around a sample $\mathbf{x}$. In particular, let $c^* = \arg\max_{c=1,\ldots,C} \bar{f}_{\sigma,c}(\mathbf{x})$ be the predicted class and $c_* = \arg\max_{c\neq c^*} \bar{f}_{\sigma,c}(\mathbf{x})$ be the runner up class. Then an attacker needs to move $\mathbf{x}$ at least by a distance

$$R = (\bar{f}_{\sigma,c^*}(\mathbf{x}) - \bar{f}_{\sigma,c_*}(\mathbf{x}))/\sqrt{2\pi\sigma^2} \tag{3}$$

to flip the decision of the smoothed classifier. Hence, one can guarantee constant classification within an ℓ_2-ball of diameter R around $\mathbf{x}$: $\arg\max_c \bar{f}_{\sigma,c}(\mathbf{x}+\boldsymbol{\delta}) = \arg\max_c \bar{f}_{\sigma,c}(\mathbf{x})$ for $\|\boldsymbol{\delta}\| < R$.

Randomized smoothing is compelling since it is simple, scalable, model agnostic and provides strong guarantees, which have been shown to be tight [Cohen et al., 2019]. However, one of the central challenges in randomized smoothing is that smoothed classifiers are in fact *intractable*, since neither $\mathbb{P}[f(\mathbf{x}+\boldsymbol{\epsilon}) = c]$ nor $\mathbb{E}_{\boldsymbol{\epsilon}}[f(\mathbf{x}+\boldsymbol{\epsilon})]$ can be computed exactly in general, especially when f is represented by an arbitrary DNN. To overcome this limitation, the required probabilities (or expectations) are replaced with *Monte Carlo estimates, which can only give probabilistic guarantees for the non-existence of adversarial examples.* As usual in Monte Carlo techniques, this introduces a trade-off between *inference quality*—determined by the variance of the estimator, and scaling indirectly with number L of Monte Carlo samples—and *inference time*—scaling directly with L. Fewer samples lead to weaker probabilistic guarantees and smaller certified regions, while more samples require more network evaluations. For example, Cohen et al. [2019] report that $100,000$ Monte Carlo samples are needed for a confidence level of 99.9% for a radius 4σ—that is, in order to certify a *single* test sample, one needs to evaluate the DNN $100,000$ times. This incurs a significant increase in computational cost, which becomes prohibitive in real-time and high-throughput applications.

In this paper, we argue that (in-)tractability is a choice, and in particular, a *structural choice.* In particular, we introduce a novel class of deep models, *decomposable neural networks* (DecoNets), which are able to compute the functional $\mathbb{E}_{\boldsymbol{\epsilon}}[f(\mathbf{x}+\boldsymbol{\epsilon})]$ *exactly and in a single network evaluation*. Compared to the Monte Carlo approaches to randomized smoothing, DecoNets not only provide *exact* adversarial certification (i.e., with a confidence level of 100%) but also dramatically reduce the time needed to certify the non-existence of adversarial examples, as shown in Fig. 1. DecoNets are able to deliver this by combining expressive architectural patterns like multiplicative interaction layers [Jayakumar et al., 2019], from which they inherit universal approximation, with decomposable computational graphs such as probabilistic circuits (PCs) [Vergari et al., 2020, Choi et al., 2020] which enable the tractable computation of multivariate integrals. Specifically, DecoNets realize set-multilinear polynomials over arbitrary non-linear input features, represented as standard neural networks over sub-scopes of the inputs. In the following sections we introduce the layers in DecoNets as operations one is allowed to combine to obtain exact computation of their smoothed outputs and provide a general recipe to combine them for image data while retaining these guarantees throughout the whole architecture.

2 DECOMPOSABLE NEURAL NETWORKS

While randomized smoothing is simple and effective, it faces a major challenge, namely the computation of the expectations $\mathbb{E}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right]$ which are analytically intractable in general DNNs. In Cohen et al. [2019], Salman et al. [2019a] the expectations are approximated with Monte Carlo estimation, which comes with the usual caveats, namely i) high computational cost and ii) probabilistic guarantees only. These two caveats also form a trade-off, since tighter probabilistic guarantees require more computation, since the estimator variance decreases reciprocal with the number of Monte Carlo samples.

It seems to be a tacit assumption in the community that an *exact* and *efficient* computation $\mathbb{E}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right]$ must remain a hopeless endeavor, or vice versa, that any model which does allow this computation cannot live up to the performance of DNNs. In this paper, we challenge this assumption and propose *decomposable neural networks (DecoNets),* a flexible class of DNNs which *does* facilitate an exact and efficient computation of $\mathbb{E}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right]$.

In order to introduce DecoNets, note that any DNN classifier is a *directed acyclic computational graph* over modules (or layers), many-to-many functions whose inputs are given by the output of other modules or by the input vector $\mathbf{x}$, and whose outputs are arbitrary tensors, subsuming scalars, vectors, matrices, etc.

We can naturally distinguish two types of modules, namely *input modules*, whose input stems exclusively from $\mathbf{x}$, and *internal modules*, which receive input from at least one other module in the graph. An important property in DecoNets is that each module depends only on a *sub-vector* of $\mathbf{x}' \subseteq \mathbf{x}$,[1] which we call the *scope* or *receptive field* of the module, and which we denote with $\mathbf{x}_g$ for any module g. For example, if $\mathbf{x} = (x_1, x_2, x_3)^\top$ and $g(\mathbf{x}) := x_3 - x_1^2$, then $\mathbf{x}_g = (x_1, x_3)$. Note, that given the scopes of the input modules, the scopes of the internal modules can be easily determined by recursion.

The principle of DecoNets is relatively simple and is based on two requirements. First, if g is an input module, we require that the smoothing operator $\mathbb{E}\left[g(\mathbf{x}+\boldsymbol{\epsilon})\right]$ can be computed easily (e.g. in closed form). Note that $\mathbb{E}\left[g(\mathbf{x}+\boldsymbol{\epsilon})\right] = \mathbb{E}\left[g(\mathbf{x}_g+\boldsymbol{\epsilon}')\right]$ where $\boldsymbol{\epsilon}'$ is a isotropic Gaussian noise vector of the same length as $\mathbf{x}_g$, i.e. the smoothing operator can be restricted to the scope of g. Second, we require that internal modules are commutative with the smoothing operator, i.e., if $g(i, h, j, \dots)$ is an internal module, then $\mathbb{E}[g(h(\mathbf{x}+\boldsymbol{\epsilon}), i(\mathbf{x}+\boldsymbol{\epsilon}), j(\mathbf{x}+\boldsymbol{\epsilon}), \dots)] = g(\mathbb{E}[h(\mathbf{x}+\boldsymbol{\epsilon})], \mathbb{E}[i(\mathbf{x}+\boldsymbol{\epsilon})], \mathbb{E}[j(\mathbf{x}+\boldsymbol{\epsilon})], \dots)$.

In the following, we introduce four modules—the *decomposable multiplicative interaction*, the *input*, the *normalization*, and the *output layers*—that satisfy the two aforementioned desiderata and as such guarantee to exactly compute the expectations required for distilling a smoothed classifier (Def. 1.1) efficiently when composed together in a computational graph for a DecoNet. We then describe in Sec. 3 how to use them to build a DecoNet tailored for image data.

2.1 DECOMPOSABLE MULTIPLICATIVE INTERACTION MODULES

Multiplicative interactions [Jayakumar et al., 2019] are a general way to combine multiple input streams by realizing a bilinear tensor product. This module has been particularized in a number of successful recent neural architectures such as gating and attention layers [Vaswani et al., 2017, Bahdanau et al., 2014], dynamic convolutions [Wu et al., 2019], and hypernetworks [Ha et al., 2017]. Unfortunately, all these architectural variants cannot efficiently compute Eq. (2). To overcome this issue, we design a multiplicative interaction moduel with an additional structural constraint—*decomposability*—which ensures that the input streams are functions with non-overlapping scopes.

Definition 2.1 (Decomposable multiplicative interactions)**.** Let $g(\mathbf{X}_1) \in \mathbb{R}^m$ and $h(\mathbf{X}_2) \in \mathbb{R}^n$ be functions defined over disjoint sets of variables, i.e., $\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$. Then, a parameterized function $f(\mathbf{X}) \in \mathbb{R}^o$ realizes a *decomposable multiplicative interaction* (DeMI) module over variables $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2$ if it takes the following form for an input $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$:

$$f(\mathbf{x}) = g(\mathbf{x}_1)^\top \cdot \mathbb{W} \cdot h(\mathbf{x}_2) + g(\mathbf{x}_1)^\top \cdot \mathbf{U} + \mathbf{V} \cdot h(\mathbf{x}_2) + \mathbf{b} \quad (4)$$

where $\mathbb{W} \in \mathbb{R}^{m,o,n}$, $\mathbf{U} \in \mathbb{R}^{m,o}$, $\mathbf{V} \in \mathbb{R}^{n,o}$ and $\mathbf{b} \in \mathbb{R}^o$ are parameter tensors and the k-th entry of the first term computes $\sum_{i,j} g(\mathbf{x}_1)_i \cdot \mathbb{W}_{i,j,k} \cdot h(\mathbf{x}_2)_j$.

Decomposability restricts the multiplicative interactions to encode set-multilinear polynomials [Shpilka and Yehudayoff, 2010] whose indeterminates are given by the output entries of the functions g and h. However, imposing decomposability does not hinder the universal approximator capabilities of DeMI layers, as discussed in Sec. 2.5, but is the key to unlocking tractable computation of Eq. (2), as illustrated by the following theorem.

Theorem 2.1 (Smoothed decomposable multiplicative interactions)**.** *Let f be a decomposable multiplicative interaction module. Then, its smoothed output $\bar{f}_\sigma(\mathbf{x})$ for an input $\mathbf{x} \in \mathcal{X}$ and Gaussian noise magnitude σ is given by*

$$(\bar{g}_\sigma(\mathbf{x}_1))^\top \cdot \mathbb{W} \cdot \bar{h}_\sigma(\mathbf{x}_2) + (\bar{g}_\sigma(\mathbf{x}_1))^\top \cdot \mathbf{U} + \mathbf{V} \cdot \bar{h}_\sigma(\mathbf{x}_2) + \mathbf{b}, \quad (5)$$

where $\bar{g}_\sigma$ and $\bar{h}_\sigma$ are the smoothed versions of g and h.

In essence, Thm. 2.1 states that if the inputs of a multiplicative interaction module admit tractable smoothing, so does the module itself. We can ensure this recursively by composing several multiplicative interaction modules while paying preserving decomposability (we will provide our architectural recipe to do so in Sec. 3) and alternating them with normalization layers (Sec. 2.3). We enlarge these options by including standard perceptrons and convolutions as possible input layers showing how they can be tractably smoothed.

2.2 INPUT MODULES: RELU PERCEPTRONS AND CONVOLUTIONS

Even for the simple case of a single perceptron $f(\mathbf{x}) = r(\boldsymbol{w}^T \cdot \mathbf{x})$, i.e., an affine function followed by a non-linear activation r, tractable computation of its smoothing depends on the choice of r. For example, if r is the sigmoid activation, then computing Eq. (2) exactly might not be possible, as no closed-form integral is known for the expectation of an arbitrary sigmoid with Gaussian inputs. Fortunately, this is not the case for rectified linear units (ReLUs), as the next proposition shows.

[1]For convenience, and with slight abuse of notation, we use set notation for vectors.

Proposition 2.1. *Let f denote an affine function with a ReLU activation, i.e., $f(\mathbf{x}) = \max(0, \boldsymbol{w}^\top \mathbf{x} + b)$. The smoothing of f has the following expectation:*

$$\bar{f}_\sigma(\mathbf{x}) = (\boldsymbol{w}^T\mathbf{x} + b)\,\Phi(\gamma) + \exp\left(-\frac{\gamma^2}{2}\right)\frac{\sigma\|\boldsymbol{w}\|}{\sqrt{2\pi}}, \quad (6)$$

where $\gamma = \frac{\boldsymbol{w}^T\mathbf{x}+b}{\sigma\|\boldsymbol{w}\|}$ and Φ is the standard normal CDF.

As such, we are allowed to create input modules that concatenate the outputs of I perceptrons. Note that this result extends also to convolutions followed by ReLU activations, as they realize linear operators with parameter sharing [LeCun, 1998].

2.3 NORMALIZATION AND REGULARIZATION MODULES

A miscellaneous of regularization and normalization layers commonly used in deep learning can be readily incorporated in DecoNets as modules whose computations at inference time would easily commute with the expectation as they realize affine functions. For example, a **dropout** layer receiving input from a module f would simply realize $p_d \cdot f(\mathbf{x})$ at inference time, where $1 - p_d$ is the probability of zeroing the outputs of f at training time. Hence, its smoothed version can be easily computed as $p_d \cdot \bar{f}_\sigma(\mathbf{x})$. Similarly, as a **batch norm** module would realize the affine transformation of the k-th dimension as $\gamma_k \cdot ((f(\mathbf{x})_k - \mu_k)/\sigma_{\mathsf{BN}}) + \beta_k$, where γ_k and β_k are some learned parameters and μ_k and σ_{BN} are the mean and standard deviation statistics estimated on the training set (and hence constant at inference time), then smoothing it equals computing $\gamma_k \cdot ((\bar{f}_{\sigma,k}(\mathbf{x}) - \mu_k)/\sigma_{\mathsf{BN}}) + \beta_k$.

2.4 OUTPUT MODULE

The layers introduced so far guarantee the tractable computation of their smoothed equivalents, but they do not guarantee their outputs to be bounded in $[0, 1]^C$, which is a requirement for a classifier f to retrieve the radius guarantees in Eq. (3) as shown in Salman et al. [2019a]. Since we are not allowed to simply normalize the DecoNet predictions in $[0, 1]$ through a sigmoid or softmax layer, as a closed-form expectation for it is not known (Sec. 2.2), we propose to scale them with an affine function. Specifically, we adopt the following element-wise min-max transformation as the last layer:

$$f'_i(\mathbf{x}) = (f(\mathbf{x})_i - f_i^{\mathsf{min}})/(f_i^{\mathsf{max}} - f_i^{\mathsf{min}}) \quad (7)$$

where $f(\mathbf{x})_i$ is the unnormalized prediction for the i-th class as output by the penultimate layer, and f_i^{min} (resp. f_i^{max}) are theoretical bounds over the predictions of f. As the bounds are constants at inference time, we can again easily commute Eq. (7) with the expectation operator to smooth it.

For a DecoNet with input modules realizing ReLU perceptrons, we can efficiently compute a practical upper (resp. lower) bound for f_i^{max} (resp. f_i^{min}). Specifically, for an upper bound, we can formulate the following optimization problem

$$\begin{aligned} \max_{\mathbf{x}} \quad & f_i(f^{\mathsf{in}_1}(\mathbf{z}), \ldots, f^{\mathsf{in}_K}(\mathbf{z})) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{X}_{\mathsf{f}}, \quad \mathbf{z} = \mathsf{max}(\boldsymbol{w}^\top\mathbf{x} + \underline{,}0) \end{aligned} \quad (8)$$

where $\mathcal{X}_{\mathsf{f}}$ is the range for feasible input values, e.g., $[-0.5, 0.5]^D$ for zero-centered images comprising D pixels and $f^{\mathsf{in}_1}(\mathbf{z}), \ldots, f^{\mathsf{in}_K}(\mathbf{z})$ denote K different input modules in the DecoNet. For f_i^{min} we can formulate an equivalent minimization problem. Note that to compute Eq. (7) we do not require optimal bounds on the values of f_i^{min} and f_i^{max}, only relaxations f_i^{upp} and f_i^{low} such that $f_i^{\mathsf{min}} \geq f_i^{\mathsf{low}}$ and $f_i^{\mathsf{max}} \leq f_i^{\mathsf{upp}}$. We rewrite Eq. (8) as a quadratic program with non-linear constraints making it amenable to off-the-shelf solvers like Gurobi. Specifically we introduce for every quadratic term multiplied by another linear (or quadratic) term appearing in our DeMI modules a new variable in the objective function which has a hard constraint equal to the product of the higher order polynomial term. As a result, we can use a few iterations of Gurobi to compute f_i^{upp} and f_i^{low} to plug in Eq. (7) and ensure that the output of the DecoNet lies in $[0, 1]^C$. For a dataset like FMNIST and a DecoNet of one million parameters, this computation takes roughly 300-400 seconds depending on the tightness of the bound required. Note that we need to perform this computation *only once* and therefore we can amortize its cost for all possible queries to certify every input at inference time.

2.5 REPRESENTATIONAL POWER OF DECONETS

DecoNets realize a hierarchy of multiplicative interactions over non-linear inputs that are ReLU perceptrons. At the same time, as pointed out in Sec. 2.1, DecoNets are compact representations for set-multilinear polynomials with potentially exponentially many terms. Each term in such a polynomial would comprise a product of non-linear functions defined over disjoint sets of variables. It follows from the above statements that DecoNets inherit from their neural and polynomial nature the universal approximator property: by adding more neurons either in the input layers as ReLU perceptrons or by increasing the parameterization of the DeMI layers DecoNets could better approximate any target function with reduced error.

While the above statements tell us that DecoNets can be as expressive as general DNNs and other feedforward networks such as ConvNets, they do not guarantee that they are equivalently *expressive efficient* [Shpilka and Yehudayoff, 2010, Martens and Medabalimi, 2014], that is that they can approximate a target function equivalently well than other models with the same model capacity, here expressed in terms of number of parameters. We settle this question with our empirical evaluation in Sec. 5 where we show that DecoNets are competitive in terms of natural accuracy on a number of datasets when compared to DNNs and ConvNets with an equivalent number of parameters.

Lastly, we clarify the relationship between DecoNets and probabilistic circuits (PCs) [Vergari et al., 2020, Choi et al., 2020]. PCs compactly represent complex distributions as computational graphs comprising only sum, product and input units. Imposing structural constraints on their graphs equals to guarantee tractability for certain computations. For instance, decomposability guarantees tractable marginalization in PCs when in conjunction with *smoothness*,[2] and maximization when associated to *determinism*, two additional structural properties for circuits [Choi et al., 2020].

DecoNets can be understood as special circuits if we abstract their input layers into single computational units, as their inner layers already comprise only sum and product operations. Specifically, under this light DecoNets can be represented as *non-monotonic*[3] and decomposable circuits whose input units encode DNNs. In fact, differently from PCs, which are required to have positive parameters (also called monotonic circuits), DecoNets are not restricted to model probability distributions, and therefore allow for arbitrary parameterizations. Furthermore, DecoNets are also less restricted structure-wise: while all the aforementioned special classes of PCs require some combination of decomposability, smoothness and determinism for tractable inference, only decomposability is truly necessary in DecoNets to tractably compute Eq. (2). This additional flexibility allows us to leverage architectural biases in DecoNets that are not allowed in standard PCs because they would “break” some other structural property or the probabilistic semantics of the circuit (e.g., BatchNorm). At the same time, we can leverage the rich literature of PCs for effortless ways to build a structure for DecoNets whose layers are guaranteed to preserve decomposability, as the next section discusses.

3 A DECONET FOR IMAGES

We now give a simple recipe to compose the tractable ingredients introduced in the previous section to obtain a DecoNet that can robustly classify and efficiently certify image data. The main challenge in stacking multiple modules of this kind together is preserving decomposability across all DeMI layers. To this end, we adapt an approach to build decomposable PCs first introduced by Poon and Domingos [2011] and then extended to tensorized representations by Peharz et al. [2020a]. At a high level, this process consists of two steps: i) building a hierarchical decomposition of the global function scope, also called a *region graph* and then ii) following it to build a computational graph that properly stacks DeMI layers. We describe these two steps next, illustrated in Fig. 2.

Building the region graph. The region graph is a bipartite graph comprising two kinds of nodes: regions and partitions. Regions encode subsets of variables $\mathbf{X}$, among them the so-called "top region" representing a the full scope $\mathbf{X}$. “Leaf regions” indicate the regions in the graph with the smallest scope. In the context of images, regions correspond to patches, e.g., collection of pixels that are highlighted with different shades of blue in Fig. 2, left. On the other hand, partitions are collections of disjoint regions and represent ways to decompose a region. Without loss of generality, we consider partitioning regions into two sub-regions only. More formally, for a region with scope $\mathbf{X}_r$, a partition denotes the collection $(\mathbf{X}_{r_1}, \mathbf{X}_{r_2})$ where $\mathbf{X}_{r_1} \cap \mathbf{X}_{r_2} = \emptyset$ and $\mathbf{X}_{r_1} \cup \mathbf{X}_{r_2} = \mathbf{X}_r$. For example, the top region in Fig. 2, with scope (r, s, t, u, v, w) is decomposed by the partition denoted by a black square indexed by 8 into two sub-regions, one of which (indexed as 7) has scope (r, s, u, v) and the other one (t, w). Note that a region can be partitioned in different ways. It is easy to see that such a region graph admits only decomposable partitions by construction and realizes a *poset* over the possible subsets of the top region.

Given some image data with height H and width W, to build a region graph we perform a recursive partitioning process that starts from the top region—the full image—and then keeps on partitioning it into axis-aligned patches, until the generated regions have a large enough scope—becoming the leaf regions. Specifically, we execute d horizontal (resp. vertical) splits on a region r to partition H_r (resp W_r) in equal parts (up to integer rounding) until we cannot split a region’s dimension in d parts anymore. We then connect each region to its corresponding sub-regions via partitions, by caching the nodes we introduced in the region graph to obtain a DAG as the one depicted in Fig. 2 where each region or partition are unique.

Tensorizing DecoNet layers. Once a region graph is available, we can build a computational graph for DecoNets in the following way. For each leaf in the region graph we introduce an input layer comprised of I neurons with ReLU activations as described in Sec. 2.2. Then, for each partition node in the region graph, partitioning a region r with scope $\mathbf{X}_r$ into two sub-regions with scopes $\mathbf{X}_{r_1}$ and $\mathbf{X}_{r_2}$ we introduce a block comprised of a DeMI layer followed by Dropout and Batch Norm layers, as indicated by colors in Fig. 2 right. The DeMI layer is parameterized by $\mathbb{W}$ and realizes the function $g(\mathbf{x}_{r_1})^T \cdot \mathbb{W} \cdot h(\mathbf{x}_{r_2})$, where g and h represent the outputs of the blocks associated to the two sub-regions in the current partition. To speed up computations, we implement these DeMI layers as a monolithic einsum op-

[2]Not to be confused with functional smoothness, smoothness in PCs constraints the input to a sum units to be functions defined over the same sets of variables.

[3]Also referred to as *general circuits* in Vergari et al. [2021].

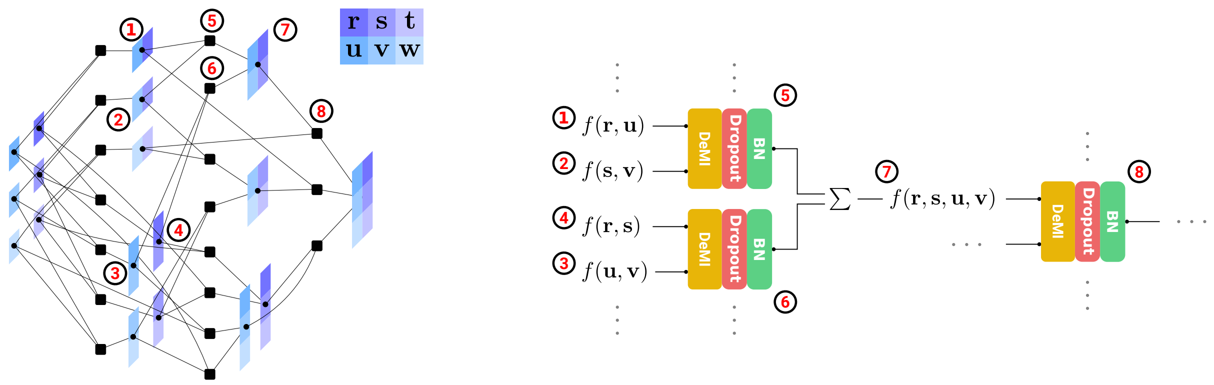
r s t
u v w
1
2
3
4
5
6
7
8
$f(\mathbf{r},\mathbf{u})$
$f(\mathbf{s},\mathbf{v})$
$f(\mathbf{r},\mathbf{s})$
$f(\mathbf{u},\mathbf{v})$
DeMI
Dropout
BN
$\sum$
$f(\mathbf{r},\mathbf{s},\mathbf{u},\mathbf{v})$

Figure 2: On the left, a region graph created over the set of variables (r, s, t, u, v, w) defined over a 2×3 image patch. Regions are color-coded as to denote their scope while partitions are denoted as black squares. On the right: a fragment of the computational graph of a DecoNet where DeMI, Dropout and BatchNorm modules are stacked to represent the corresponding partition in the region graph on the left.

eration as in [Peharz et al., 2020b]. Lastly, as one region can be partitioned in multiple ways, we mix the aforementioned blocks associated to the partitions into a single output by a weighted sum operation (indicated by $\sum$ in Fig. 2).

4 RELATED WORK

Decomposability and Circuit Representations. Constraining computational graphs to enable the tractable computation of certain quantities of interest has a long history in ML and AI [Davis et al., 1962, Darwiche and Marquis, 2002, Bacchus et al., 2009, Kimmig et al., 2017]. Decomposability, in particular, has been first introduced as a structural property enabling tractable computations of SAT and weighted model counting in the literature of logical circuits [Darwiche and Marquis, 2002], the logical counterpart of PCs, encoding Boolean functions with computational graphs of and and or gates. Within the literature of PCs, only a few works have investigated decomposable discriminative circuits for classification [Rooshenas and Lowd, 2016, Rahman et al., 2019, Liang and Van den Broeck, 2019, Shao et al., 2020]. However, all of them introduce additional structural properties such as determinism or structured-decomposability that further limit their expressive efficiency as well as their ability to accommodate for the most recent neural architectural advancements (Sec. 2.5). Equally crucially, to retain a probabilistic semantics they additionally restrict their computational graphs to compute valid probabilities throughout it by limiting the choice of input layers, forcing network parameters to be in a simplex, or squashing the output value through a sigmoid. DecoNets, on the other hand, require only decomposability and do not enforce a probabilistic semantics, as such they can easily accommodate convolutions and batch norm layers (Sec. 2). Furthermore, circuits directly encoding a conditional probability distribution $P(\mathbf{Y} \mid \mathbf{X})$ [Rooshenas and Lowd, 2016, Rahman et al., 2019, Shao et al., 2020] can only impose decomposability over the output variables $\mathbf{Y}$ thus hindering the possibility to solve Eq. (2) efficiently.

Non-certified Defenses. There have been countless works proposing empirical defenses against adversarial examples. Perhaps the most successful approach so far as been adversarial training Kurakin et al. [2016], Madry et al. [2018], in which training is performed on worst-case examples within a local neighborhood of each training example. However, since none of these defenses possess provable guarantees, there is always the possibility that a new proposed attack could break them. Indeed, most empirical techniques which are purported to be robust are quickly circumvented by better design of adversarial examples Athalye et al. [2018], Athalye and Carlini [2018], Tramer et al. [2020]. As a result, many in the community have directed their attention towards approaches which provide certified protection.

Certified Defenses. Many recent approaches provide certifiable guarantees that no adversarial example exists within a certain radius of a point. Several of these approaches are exact, including those based on SMT solvers Katz et al. [2017], Ehlers [2017] or mixed integer linear programs Tjeng et al. [2017], Lomuscio and Maganti [2017], Fischetti and Jo [2017]. Unfortunately, these approaches are generally not computationally efficient due to the underlying optimization techniques. Other recent works sacrifice these exact guarantees for conservative ones (i.e., a point's certified radius may be more narrow than its true one) in order to reduce computation Wong and Kolter [2018], Wang et al. [2018], Raghunathan et al. [2018a,b], Dvijotham et al. [2018a,b], Mirman et al. [2018], Zhang et al. [2018], Salman et al. [2019b], though it still has trouble scaling to large network sizes. Randomized smoothing is the most relevant approach to our investigation. This achieves better scalability than other certified methods, though the guarantees

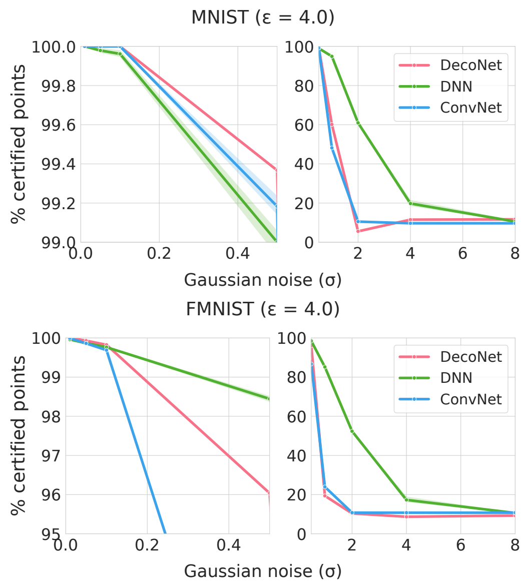

Figure 3: Percentage of certified points (y-axis) w.r.t. different Gaussian noise magnitudes (σ, x-axis) for MNIST (top) where DecoNets and FMNIST (bottom). We can see that DecoNets are competitive with ConvNets for all σ values and with DNNs for $\sigma \leq 1$, for larger values adversarial samples are not easily recognizable by humans.

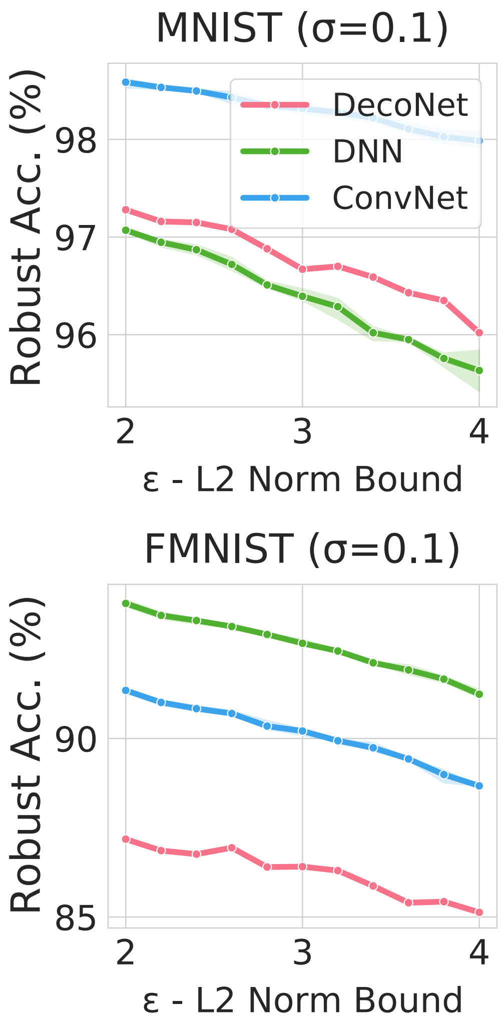

Figure 4: Robust accuracy (y-axis) for all models under attacks with different L_2 norm radiuses (x-axis) on MNIST (top) and FMNIST (bottom).

provided are only probabilistic. The first applications of randomized smoothing were empirical defenses Liu et al. [2018], Cao and Gong [2017], but Lecuyer et al. Lecuyer et al. [2019] proved certified ℓ_2 robustness using techniques from differential privacy Dwork et al. [2006], which later works refined Cohen et al. [2019], Salman et al. [2019a]. Other works have applied randomized smoothing for certified robustness in other distances, including ℓ_1, ℓ_∞, ℓ_0, and Wasserstein distance Teng et al. [2019], Zhang et al. [2019], Levine and Feizi [2020a,b], Lee et al. [2019], Yang et al. [2020]. Finally, yet another line of work guarantees certified ℓ_∞-robustness by imposing a Lipschitz property on the network via alternative architectures Anil et al. [2019], Li et al. [2019]. While our approach also introduces a new architecture for robustness, we differ in that our architecture instead supports closed-form computation of expectations, and we certify ℓ_2 rather than ℓ_∞ robustness.

5 EXPERIMENTS

In this section, we aim to answer the following questions: **Q1)** Are DecoNets as expressive as other deep models like DNNs and convolutional neural networks (ConvNets)? **Q2)** Do DecoNets possess similar robustness properties as these other architectures? **Q3)** How do Deconets compare to these other models in terms of time and accuracy when certifying data points?

Datasets. To answer the aforementioned questions, we utilize MNIST LeCun [1998] and Fashion-MNIST Xiao et al. [2017]. In future works we plan to scale DecoNets to accurately classify larger image problems such as ImageNet Deng et al. [2009], which undoubtedly requires to accomodate additional architectural biases in our framework as well as more computational resources to run a systematical empirical evaluation.

Model Architectures. The baseline models being used are fully connected DNNs with ReLU activations and ConvNets with a fully-connected classification layer. For the DecoNet trained on MNIST (resp. FMNIST), we build a region graph splitting each region dimensions into 5 (d, Sec. 3) pieces and we use 8 (resp. 12) input perceptrons per input regions. For DecoNets we use a weight decay of 0.0002 and experimented with dropout, finding it less

MNIST (300K)			FMNIST (1M)		
DNN	ConvNet	DecoNet	DNN	ConvNet	DecoNet
98.7	99.3	99.1	88.6	90.3	90.1

Table 1: Natural accuracy of Baselines and DecoNet and as we can see, the DecoNet is able to obtain comparable results to the baseline architectures

beneficial for certified accuracies. Finally, we employ for DecoNets a uniform combination of multiple classification output layers as to have gradients backpropagate at different depths as in GoogLeNetSzegedy et al. [2015]. Additional architectural details are reported in the Appendix. All models on MNIST (resp. FMNIST) have roughly 300K (resp. 1 million) parameters.

Q1) Natural accuracy. Tab. 1 summarizes the accuracy of all models on all datasets. DecoNets perform comparably to all baselines despite having a an additional structural constraint like decomposability. As the models have the same parameter counts, we can deduce that DecoNets can be as expressive efficient as ConvNets and DNNs. These results are quite promising since they hint that a deep model that allows to perform smoothing exactly and in a single pass can be as accurate as intractable ones, therefore we can answer our research question affirmatively.

Q2) Robust accuracy. To answer this question we considered the ℓ_2-PGD attack Madry et al. [2017]. Note that this attack produces similar results to the DDN ℓ_2 attack Rony et al. [2019], which just requires fewer iterations. For this attack, we fix the number of iterations to be 40 and the epsilon threshold to vary from 2 to 4 in increments of 0.2. The method for adversarial training employed in this paper is the same as the robust training procedure in Madry et al. [2017]. This procedure takes the attacked data point as the input for the DecoNet and trains on this sample instead of the existing sample. Tab. 1 collects our results. DecoNets have comparable robust accuracy as the baseline model comparisons with results for Fashion-MNIST, DecoNets that are only approximately 2% worse in robust accuracy. Despite this, we observe that DecoNets share the same properties as deep networks when it comes to adversarial defenses. In Fig. 4 we notice that as the value of ϵ increases, the change in the robust accuracy is not large.

Q3) Certified accuracy and times Fig. 3 illustrates our results. For small values of $\sigma \leq 1$, DecoNets are able to certify more samples than ConvNets and DNNs for MNIST and still surpass ConvNets on FMNIST. For larger σ, DNNs certify a higher percentage but have a lower natural accuracy. On the other hand, DecoNets are always competitive with ConvNets. This is quite promising for a novel deep model class such as DecoNets: with respect to these other baselines they introduce an additional structural constraint, decomposability, have not access to the myriad of "tricks of the trade" brought up in the burgeoning deep learning literature and yet perform competitively. More importantly, they can deliver much faster and exact certification guarantees. Fig. 1 shows that despite GPU computations and batching Monte Carlo samples, to certify a single input point for $\sigma = 1$, a radius of 2σ and with 99.99% confidence for both DNNs and ConvNets of different sizes takes up to 4 orders of magnitude more than for DecoNets.

6 DISCUSSION AND CONCLUSION

In this paper, we introduced DecoNets, a promising novel class of deep classifiers that is able to provide exact guarantees for adversarial robustness in a fraction of the time that comparably expressive deep and convolutional neural networks would require to match them with very high probability. We also proposed a simple way to build DecoNets for image data, by combining recent advancements in the literature of deep classifiers and probabilistic circuits.

At the same time, several interesting questions arise around this novel model class. First, we would like to scale building and learning DecoNets for larger benchmarks including Cifar-10 and ImageNet, employ more sophisticated modules from "the deep learning toolkit" such as residual connections and demonstrate their robustness properties as well. Finally, we would like to investigate which other inference scenarios the structural properties of DecoNets can tractably deal with, thus distilling deep classifiers that are reliable and yet efficient.

References

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In *International Conference on Machine Learning*, pages 291–301. PMLR, 2019.

Anish Athalye and Nicholas Carlini. On the robustness of the cvpr 2018 white-box adversarial example defenses. *arXiv preprint arXiv:1804.03286*, 2018.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, pages 274–283. PMLR, 2018.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Solving# sat and bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34: 391–442, 2009.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 278–287, 2017.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320, 2019.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018a.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, page 3, 2018b.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Conference on Theory of Cryptography*, TCC '06, pages 265–284, Berlin, Heidelberg, 2006. Springer.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *arXiv preprint arXiv:1712.06174*, 2017.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *ICLR (Poster)*. OpenReview.net, 2017.

Siddhant M Jayakumar, Wojciech M Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2019.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22: 46–62, 2017.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019.

Guang-He Lee, Yang Yuan, Shiyu Chang, and Tommi S Jaakkola. Tight certificates of adversarial robustness for randomly smoothed classifiers. *arXiv preprint arXiv:1906.04948*, 2019.

Alexander Levine and Soheil Feizi. Robustness certificates for sparse adversarial attacks by randomized ablation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4585–4593, 2020a.

Alexander Levine and Soheil Feizi. Wasserstein smoothing: Certified robustness against wasserstein adversarial attacks. In *International Conference on Artificial Intelligence and Statistics*, pages 3938–3947. PMLR, 2020b.

Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *arXiv preprint arXiv:1911.00937*, 2019.

Y. Liang and G. Van den Broeck. Learning logistic circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4277–4286, 2019.

Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *CoRR*, abs/1411.7717, 2014.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586. PMLR, 2018.

R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van den Broeck, K. Kersting, and Z. Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574, 2020a.

R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of UAI*, pages 337–346, 2011.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *arXiv preprint arXiv:1811.01057*, 2018a.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018b.

Tahrima Rahman, Shasha Jin, and Vibhav Gogate. Cutset bayesian networks: A new representation for learning rao-blackwellised graphical models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

Amirmohammad Rooshenas and Daniel Lowd. Discriminative structure learning of arithmetic circuits. In *Artificial Intelligence and Statistics*, pages 1506–1514. PMLR, 2016.

H. Salman, J. Li, I. Razenshteyn, P. Zhang, H. Zhang, S. Bubeck, and G. Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, volume 32, 2019a.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019b.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pages 401–412. PMLR, 2020.

J. J. Sharples and J. C. V. Pezzey. Expectations of linear functions with respect to truncated multinormal distributions–with applications for uncertainty analysis in environmental modelling. *Environmental Modelling & Software*, 22 (7):915–923, 2007.

Amir Shpilka and Amir Yehudayoff. *Arithmetic circuits: A survey of recent results and open questions*. Now Publishers Inc, 2010.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Jiaye Teng, Guang-He Lee, and Yang Yuan. ℓ_1 adversarial robustness certificates: a randomized smoothing approach. 2019.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

A. Vergari, YJ. Choi, R. Peharz, and G Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. `http://starai.cs.ucla.edu/slides/AAAI20.pdf`, 2020. Tutorial at AAAI 2020.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations: From simple transformations to complex information-theoretic queries. *arXiv preprint arXiv:2102.06137*, 2021.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *arXiv preprint arXiv:1809.08098*, 2018.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. In *ICLR*. OpenReview.net, 2019.

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

Greg Yang, Tony Duan, J Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning*, pages 10693–10705. PMLR, 2020.

Dinghuai Zhang, Mao Ye, Chengyue Gong, Zhanxing Zhu, and Qiang Liu. Filling the soap bubbles: Efficient black-box adversarial certification with non-gaussian smoothing. 2019.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4944–4953, 2018.

A APPENDIX

Proof of Thm. 2.1

Proof. By linearity of expectations and by grouping expectations into tensors as in Def. 1.1 we obtain that:

$$\bar{f}^{f}_{\sigma}(\mathbf{x}) = \bar{f}^{a}_{\sigma}(\mathbf{x}) + \bar{f}^{b}_{\sigma}(\mathbf{x}) + \bar{f}^{c}_{\sigma}(\mathbf{x}) + \mathbf{b}.$$

where a, b, c denote the functionals $a(\mathbf{x}) = g(\mathbf{x}_1)^T \cdot \mathbb{W} \cdot h(\mathbf{x}_2)$, $b = g(\mathbf{x}_1)^T \cdot \mathbf{U}$ and $c = \mathbf{V} \cdot h(\mathbf{x}_2) + \mathbf{b}$.

Then, by noting that each i-th entry in a computes $\sum_{m,n} \mathbb{W}_{m,i,n} \cdot g(\mathbf{x}_1)_m \cdot h(\mathbf{x}_2)_n$ and that the joint Gaussian density over the noise fully factorizes as $\mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \sigma^2\mathbf{I}) = \mathcal{N}(\boldsymbol{\epsilon}_1; \mathbf{0}_m, \sigma^2\mathbf{I}_m) \cdot \mathcal{N}(\boldsymbol{\epsilon}_2; \mathbf{0}_n, \sigma^2\mathbf{I}_n)$ where $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}_1; \boldsymbol{\epsilon}_2]$ denotes the Gaussian noise vector partitioned in the same way as $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2]$ we obtain that

$$\begin{aligned}
\bar{f}^{a}_{\sigma}(\mathbf{x})_i =& \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0}_D,\sigma^2\mathbf{I}_D)}\left[\sum_{m,n} \mathbb{W}_{m,i,n} \cdot g(\mathbf{x}_1+\boldsymbol{\epsilon}_1)_m \cdot h(\mathbf{x}_2+\boldsymbol{\epsilon}_2)_n\right] \\
=& \sum_{m,n} \mathbb{W}_{m,i,n} \cdot \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0}_D,\sigma^2\mathbf{I}_D)}\left[g(\mathbf{x}_1+\boldsymbol{\epsilon}_1)_m \cdot h(\mathbf{x}_2+\boldsymbol{\epsilon}_2)_n\right] \\
=& \sum_{m,n} \mathbb{W}_{m,i,n} \cdot \int\int \mathcal{N}(\boldsymbol{\epsilon}_1; \mathbf{0}_m, \sigma^2\mathbf{I}_m) \cdot \mathcal{N}(\boldsymbol{\epsilon}_2; \mathbf{0}_n, \sigma^2\mathbf{I}_n) \\
& \cdot g(\mathbf{x}_1+\boldsymbol{\epsilon}_1)_m \cdot h(\mathbf{x}_2+\boldsymbol{\epsilon}_2)_n \, d\boldsymbol{\epsilon}_1 \, d\boldsymbol{\epsilon}_2 \\
=& \sum_{m,n} \mathbb{W}_{m,i,n} \cdot \mathbb{E}_{\boldsymbol{\epsilon}_1}\left[g(\mathbf{x}_1+\boldsymbol{\epsilon}_1)_m\right] \cdot \mathbb{E}_{\boldsymbol{\epsilon}_2}\left[h(\mathbf{x}_2+\boldsymbol{\epsilon}_2)_n\right]
\end{aligned}$$

by first applying the linearity of expectations and then by noting that we can break the expectation over the product of functions over independent variables as the product of expectations. By collecting all these entries in tensors we obtain that $\bar{f}^{a}_{\sigma}(\mathbf{x}) = \left(\bar{f}^{g}_{\sigma}(\mathbf{x}_1)\right)^T \cdot \mathbb{W} \cdot \bar{f}^{h}_{\sigma}(\mathbf{x}_2)$.

Lastly, we can retrieve the second and third term in Eq. (5) by noting that the i-th component of $\bar{f}^{b}_{\sigma}(\mathbf{x})$ can be computed as $\bar{f}^{b}_{\sigma}(\mathbf{x})_i = \int\int \mathcal{N}(\boldsymbol{\epsilon}_1; \mathbf{0}_m, \sigma^2\mathbf{I}_m) \cdot \mathcal{N}(\boldsymbol{\epsilon}_2; \mathbf{0}_n, \sigma^2\mathbf{I}_n) \cdot g(\mathbf{x}_1+\boldsymbol{\epsilon}_1) \cdot \mathbf{U}_{:,i} \, d\boldsymbol{\epsilon}_1 \, d\boldsymbol{\epsilon}_2 = \int \mathcal{N}(\boldsymbol{\epsilon}_1; \mathbf{0}_m, \sigma^2\mathbf{I}_m) \cdot g(\mathbf{x}_1+\boldsymbol{\epsilon}_1) \cdot \mathbf{U}_{:,i} \, d\boldsymbol{\epsilon}_1 \times \int \mathcal{N}(\boldsymbol{\epsilon}_2; \mathbf{0}_n, \sigma^2\mathbf{I}_n) \, d\boldsymbol{\epsilon}_2 = \left(\bar{f}^{g}_{\sigma}(\mathbf{x}_1)\right)^T \cdot \mathbf{U}_{:,i}$ and equivalently $\bar{f}^{c}_{\sigma}(\mathbf{x})_j = \mathbf{V}_{j,:} \cdot \bar{f}^{h}_{\sigma} \cdot (\mathbf{x}_2)$. Rearranging these entries in tensor form completes the proof.

□

Proof of Prop. 2.1

Proof. First note that

$$\begin{aligned}
\mathbb{E}\left[f(\mathbf{x}+\boldsymbol{\epsilon})\right] &= \mathbb{E}\left[\max(0, \boldsymbol{w}^\top(\mathbf{x}+\boldsymbol{\epsilon})+b)\right] && (9)\\
&= \mathbb{E}\left[(\boldsymbol{w}^\top(\mathbf{x}+\boldsymbol{\epsilon})+b)\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\}\right]. && (10)
\end{aligned}$$

Equation (10) is further

$$\begin{aligned}
\mathbb{E}\left[(\boldsymbol{w}^\top\mathbf{x}+b)\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\}\right] + \mathbb{E}\left[(\boldsymbol{w}^\top\boldsymbol{\epsilon})\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\}\right] = && (11)\\
(\boldsymbol{w}^\top\mathbf{x}+b)\,\mathbb{P}\left[\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\right] + \int_{\mathcal{H}} p(\boldsymbol{\epsilon})\,\boldsymbol{w}^\top\boldsymbol{\epsilon}\,\mathrm{d}\boldsymbol{\epsilon}, && (12)
\end{aligned}$$

where $\mathcal{H} = \{\boldsymbol{\epsilon}\colon \boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\}$.

Concerning the first term in (12), note that $\boldsymbol{w}^\top\boldsymbol{\epsilon}$ is a *univariate Gaussian* with $\boldsymbol{w}^\top\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\|\boldsymbol{w}\|^2)$. Thus,

$$(\boldsymbol{w}^\top\mathbf{x}+b)\,\mathbb{P}\left[\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x}-b\right] = (\boldsymbol{w}^T\mathbf{x}+b)\,\Phi(\gamma),$$

where Φ is the standard normal CDF and $\gamma = \frac{\boldsymbol{w}^T\mathbf{x}+b}{\sigma\|\boldsymbol{w}\|}$.

Note that the second term in (12) can be written as $\int_{\mathcal{H}} p(\boldsymbol{\epsilon})\, \boldsymbol{w}^\top \boldsymbol{\epsilon}\, \mathrm{d}\boldsymbol{\epsilon} = \int_{\mathcal{H}} p(\boldsymbol{\epsilon}')\, \sigma \boldsymbol{w}^\top \boldsymbol{\epsilon}'\, \mathrm{d}\boldsymbol{\epsilon}'$, where $\boldsymbol{\epsilon}' \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Furthermore we can re-write the half-space $\mathcal{H}$ as $\mathcal{H} = \left\{ \boldsymbol{\epsilon}' \colon \frac{\boldsymbol{w}^\top}{\|\boldsymbol{w}\|} \boldsymbol{\epsilon}' \geq \frac{-\boldsymbol{w}^\top \mathbf{x} - b}{\sigma \|\boldsymbol{w}\|} \right\}$. We can use now Theorem 5 in Sharples and Pezzey [2007], which provides an exact expression for integrals of linear functions with respect to the Gaussian measure over halfspaces, precisely the form of this term. This yields

$$\int_{\mathcal{H}} p(\boldsymbol{\epsilon}')\, \sigma \boldsymbol{w}^\top \boldsymbol{\epsilon}'\, \mathrm{d}\boldsymbol{\epsilon}' = \exp\left(-\frac{\gamma^2}{2}\right) \frac{\sigma \|\boldsymbol{w}\|}{\sqrt{2\pi}}, \tag{13}$$

concluding the proof. □

B EXPERIMENTS

B.1 MODEL ARCHITECTURES

The architecture for the DNNs is a fully connected layer, followed by a ReLU activation, repeated three times and finally a fully connected head to the number of classes to be predicted followed by a softmax layer. The number of units in each layer is dependent on which dataset is being used. For MNIST, the units at each layer are [340, 150, 50, 10]. For Fashion-MNIST, the units are [700, 500, 200, 10]. For MNIST, the number of parameters is 326110 and for Fashion-MNIST, the number of parameters is 1,002,210. The ConvNet's architecture is two convolutional layers with a kernel size of 5, followed by 4 fully connected layers and a softmax layer. The ConvNet contains 305,510 parameters for MNIST and 1,033,140 parameters for Fashion-MNIST.

The DecoNet has 319,440 parameters for MNIST and 1,051,260 parameters for FMNIST.