Exact and Efficient Adversarial Robustness with Decomposable Neural Networks

Pranav Subramani [1] **Antonio Vergari** [2] **Gautam Kamath** [1] **Robert Peharz** [3]

Abstract

Randomized smoothing is a recent solution to certify the robustness of deep neural networks to adversarial examples. It provides 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*.

1. Introduction

The vulnerability of deep neural networks (DNNs) to adversarial examples—(Szegedy et al., 2014; Goodfellow et al., 2015)—small engineered input perturbations that lead to a dramatic change in the DNN's output has motivated the design of *adversarial defenses,* i.e., training and inference techniques to protect a model from these 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., 2019). This approach transforms a C-class classifier f (without robustness guarantees) into a certified *smoothed* classifier $\bar{f}_\sigma$, defined as $\bar{f}_\sigma(\mathbf{x}) = (\bar{f}_{\sigma,1}(\mathbf{x}), \ldots, \bar{f}_{\sigma,C}(\mathbf{x}))^\top$, where each entry is

$$\bar{f}_{\sigma,c}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right] \tag{1}$$

and $\boldsymbol{\epsilon}$ is zero-mean Gaussian noise with covariance $\sigma\mathbf{I}$. As shown in (Salman et al., 2019), $\bar{f}$ 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

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

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,

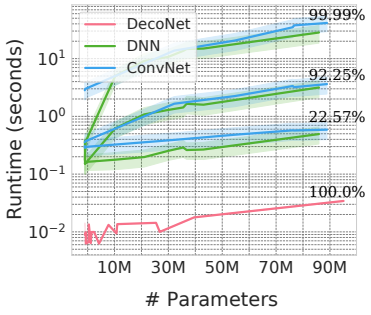

Figure 1. Scaling certified accuracy computation for $\sigma = 1$ and $R = 2\sigma$ (time in seconds, y-axis) for DecoNets, DNNs and ConvNets of equivalent number of parameters (x-axis). The Monte-Carlo estimate for DNNs and ConvNets is done for different confidence levels $(99.99, 92.25, 22.577)$ requiring 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.

[*]Equal contribution [1]Cheriton School of Computer Science, University of Waterloo, Ontario, Canada [2]University of California, Los Angeles, USA [3]Department of Mathematics and Computer Science, Eindhoven University of Technology , Netherlands. Correspondence to: Pranav Subramani <p3subram@uwaterloo.ca>.

Accepted by the ICML 2021 workshop on A Blessing in Disguise: The Prospects and Perils of Adversarial Machine Learning.

which have been shown to be tight (Cohen et al., 2019). However, a major challenge is that computing $\mathbb{E}_{\boldsymbol{\epsilon}}[f(\mathbf{x}+\boldsymbol{\epsilon})]$ cannot be done exactly in general, especially when f is represented by an arbitrary DNN. To overcome this limitation, the required 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 10^5 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 10^5 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. In the following, we introduce the layers in DecoNets as operations one is allowed to combine to obtain a deep architecture whose smoothed output can be computed exactly throughout its computational graph.

2. Decomposable Neural Networks

We start by noting 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 $g(\mathbf{x}) \coloneqq x_3 - x_1^2$, 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 *input*, the *decomposable multiplicative interaction*, the *normalization*, and the *output layers*—that satisfy the two aforementioned requirements and as such guarantee to exactly compute the expectations required for distilling a smoothed classifier efficiently when composed together in a DecoNet.

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. (7) 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.

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:*

$$\bar{f}_\sigma(\mathbf{x}) = (\boldsymbol{w}^T\mathbf{x} + b)\,\Phi(\gamma) + \exp\left(-\frac{\gamma^2}{2}\right)\frac{\sigma\|\boldsymbol{w}\|}{\sqrt{2\pi}}, \quad (3)$$

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 have input modules concatenating 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.

Decomposable Multiplicative Interaction Modules. Multiplicative interactions (Jayakumar et al., 2019) are a general way to combine multiple input streams as shown in a number of successful recent neural architectures

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

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. (7). 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. Decomposability does not hinder the universal approximator capabilities of DeMI layers, but is the key to unlocking tractable computation of Eq. (7), by commuting expectations with multiplicative interactions 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.

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 after 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 $\gamma_k \cdot ((\bar{f}_{\sigma,k}(\mathbf{x}) - \mu_k)/\sigma_{\mathsf{BN}}) + \beta_k$.

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. (2) as shown in Salman et al. (2019). To bound the output of a DecoNet, we adopt the following element-wise min-max transformation as the last layer:

$$f'_i(\mathbf{x}) = (f(\mathbf{x})_i - f_i^{\mathsf{min}})/(f_i^{\mathsf{max}} - f_i^{\mathsf{min}}) \quad (6)$$

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 some bounds over the predictions of f. As the bounds are constants at inference time, we can again easily commute Eq. (6) with the expectation operator to smooth it.

Note that f_i^{min} and f_i^{max} need not to be tight bounds, but reasonable enough to normalize outputs within numerical precision. Therefore, we can easily write the DecoNet as quadratic optimization problem with non-linear constraints (Sec. B) and come up with reasonable bounds as initial (non-optimal) solutions to this problem that can be easily found by off-the-shelf solvers such as Gurobi.

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*.

Combining modules together. DecoNets can be understood as special probabilistic circuits (PCs)(Vergari et al., 2020; Choi et al., 2020)—computational graphs involving sum and product operations efficiently encoding probability distributions—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* and decomposable circuits whose input units encode DNNs. In fact, differently from PCs, which are required to have positive parameters (making them monotonic circuits), DecoNets are not restricted to model probability distributions, and therefore allow for arbitrary parameterizations. To combine several modules together while preserving decomposability, we exploit recent advancements in the PC literature (Poon and Domingos, 2011; Peharz et al., 2020a), which we detail in Sec. C.

3. Experiments and Conclusions

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 in terms of time and accuracy when certifying data points?

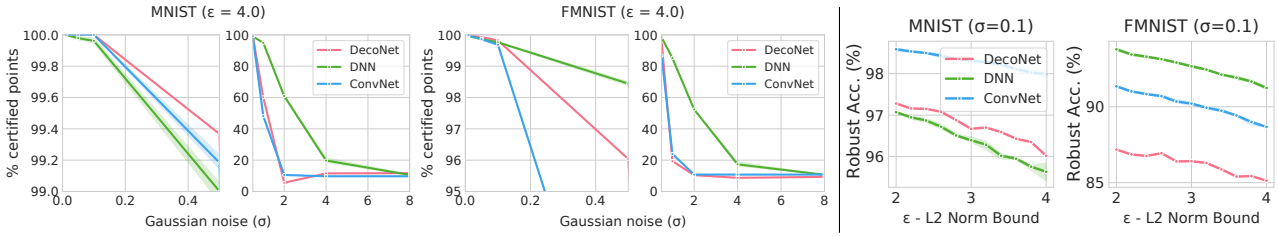

Figure 2. **On the left:** Percentage of certified points (y-axis) w.r.t. different Gaussian noise magnitudes (σ, x-axis) for MNIST and FMNIST. We can see that DecoNets are competitive with ConvNets for all σ values and with DNNs for $\sigma \leq 1$, for larger values adversarial samples are not easily recognizable by humans. **On the right:** Robust accuracy (y-axis) for all models under attacks with different ℓ_2 norm radiuses (x-axis) on MNIST and FMNIST.

Table 1. Natural accuracy of Baselines and DecoNets.

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

Datasets. To answer the aforementioned questions, we run preliminary experiments on standard benchmark datasets in computer vision, namely, MNIST (LeCun, 1998) and Fashion-MNIST (Xiao et al., 2017).

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 deep architecture as described in the Appendix in Sec. C. For DecoNets we use a weight decay of 0.0002 and experimented with dropout, finding it less beneficial for certified accuracies. Finally, we employ for DecoNets a uniform combination of multiple output layers as to facilitate gradients backpropagate at different depths as in GoogLeNetSzegedy et al. (2015). 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: decomposability. As the models have the same parameter counts, we can deduce that DecoNets can be as expressive efficient as ConvNets and DNNs and that a deep model that provides exact certification guarantees in a single pass can be as accurate as intractable models.

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). Tab. 1 collects our results. DecoNets have higher robust accuracy than DNNs on MNIST and only approximately 5% worse on FMNIST where also ConvNets underperform. Despite this, we observe that DecoNets share the same properties as deep networks when it comes to adversarial defenses: as the value of ϵ increases, the change in the robust accuracy is preserved.

Q3) Certified accuracy and times Fig. 2 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 clearly shows this: 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. Also note that DecoNets are significantly more memory efficient with GPU memory which makes it a viable choice for low memory production environments.

All in all, these preliminary results hint at the concrete possibility of having structured deep model that are both expressive and reliable 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 and second, employ more sophisticated modules from "the deep learning toolkit" such as residual connections and demonstrate their robustness properties as well.

REFERENCES

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

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.

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.

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.

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.

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.

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.

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, 2019.

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.

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.

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.

A. Appendix

Definition A.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 $\bar{f}_\sigma(\mathbf{x}) = (\bar{f}_{\sigma,1}(\mathbf{x}), \dots, \bar{f}_{\sigma,C}(\mathbf{x}))^\top$, where

$$\bar{f}_{\sigma,c}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f_c(\mathbf{x}+\boldsymbol{\epsilon})\right] \tag{7}$$

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

Proof of Thm. 2.1

Proof. By linearity of expectations and by grouping expectations into tensors as in Def. A.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

$$\begin{aligned}
\mathbb{E}\left[f(\mathbf{x}+\boldsymbol{\epsilon})\right] &= \mathbb{E}\left[\max(0, \boldsymbol{w}^\top(\mathbf{x}+\boldsymbol{\epsilon}) + b)\right] &(8)\\
&= \mathbb{E}\left[(\boldsymbol{w}^\top(\mathbf{x}+\boldsymbol{\epsilon}) + b)\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x} - b\}\right]. &(9)
\end{aligned}$$

Equation (9) is further

$$\begin{aligned}
&\mathbb{E}\left[(\boldsymbol{w}^\top\mathbf{x} + b)\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x} - b\}\right] + \mathbb{E}\left[(\boldsymbol{w}^\top\boldsymbol{\epsilon})\,\mathbf{1}\{\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x} - b\}\right] = &(10)\\
&(\boldsymbol{w}^\top\mathbf{x} + b)\,\mathbb{P}\left[\boldsymbol{w}^\top\boldsymbol{\epsilon} \geq -\boldsymbol{w}^\top\mathbf{x} - b\right] + \int_{\mathcal{H}} p(\boldsymbol{\epsilon})\,\boldsymbol{w}^\top\boldsymbol{\epsilon}\,\mathrm{d}\boldsymbol{\epsilon}, &(11)
\end{aligned}$$

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

Concerning the first term in (11), 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,

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

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 (11) 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

$$\int_{\mathcal{H}} p(\boldsymbol{\epsilon}')\, \sigma \boldsymbol{w}^\top \boldsymbol{\epsilon}'\, \mathrm{d}\boldsymbol{\epsilon}' = \exp\left(-\frac{\gamma^2}{2}\right) \frac{\sigma \|\boldsymbol{w}\|}{\sqrt{2\pi}}, \tag{12}$$

concluding the proof. □

B. Computing bounds

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

$$\max_{\mathbf{x}} \quad f_i(f^{\mathsf{in}_1}(\mathbf{z}), \ldots, f^{\mathsf{in}_K}(\mathbf{z})) \quad \text{s.t.} \qquad \mathbf{x} \in \mathcal{X}_{\mathsf{f}}, \quad \mathbf{z} = \mathsf{max}(\boldsymbol{w}^\top \mathbf{x} + \underline{b}, 0) \tag{13}$$

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. (6) 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. (13) 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. (6) 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.

C. 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. 3.

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. 3, 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. 3, with scope (r, s, t, u, v, w)

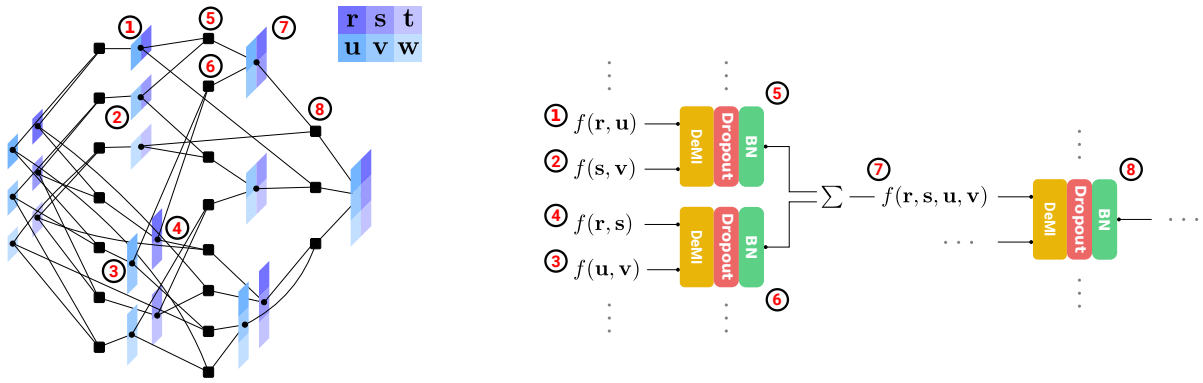

Figure 3. On the left, a region graph created over the set of variables (r, s, t, u, v, w) defined over a 2×3 image patch. Regions are color-coded as to denote their scope while partitions are denoted as black squares. On the right: a fragment of the computational graph of a DecoNet where DeMI, Dropout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

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. 3 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. 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. 3 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 operation 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. 3).

D. Experiments

D.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.