Negative Mixture Models via Squaring: Representation and Learning

Lorenzo Loconte[1] **Stefan Mengel**[2] **Nicolas Gillis**[3] **Antonio Vergari**[1]

[1]School of Informatics, University of Edinburgh, UK
[3]Department of Mathematics and Operational Research, Faculté polytechnique, Université de Mons, Belgium
[2]Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

Abstract

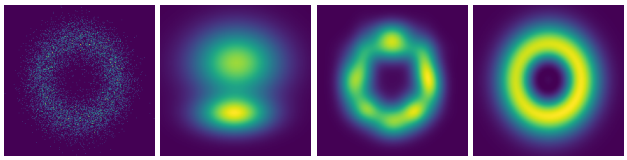
Negative mixture models (NMMs) can potentially be more expressive than classical non-negative ones by allowing negative coefficients, thus greatly reducing the number of components and parameters to fit. However, modeling NMMs features a number of challenges, from ensuring that negative combinations still encode valid densities or masses, to effectively learning them from data. In this paper, we investigate how we can model both shallow and hierarchical NMMs in a generic framework, via *squaring*. We do so by representing NMMs as probabilistic circuits (PCs) – structured computational graphs that ensure tractability. Then, we show when and how we can represent these squared NMMs as tensorized computational graphs efficiently, while theoretically proving that for certain function classes including negative parameters can exponentially reduce the model size.

1 INTRODUCTION

Finite mixture models (MMs) are a staple in tractable probabilistic reasoning and learning [McLachlan et al., 2019]. By combining simple probability mass or density functions in a convex combination, they can represent more expressive distributions. Given a set of random variables $\mathbf{X}$, a MM with K components encodes a function p over $\mathbf{X}$ as

$$p(\mathbf{X}) = \sum_{i=1}^{K} \theta_i p_i(\mathbf{X}), \quad \sum_{i=1}^{K} \theta_i = 1, \quad \theta_i \geq 0, \quad (1)$$

where each p_i is a base distribution, also called mixture component, and $\Theta = \{\theta_i\}_{i=1}^{K}$ are the mixture parameters. The convexity constraint on Θ is a *sufficient* and handy condition to guarantee that $p(\mathbf{X})$ encodes a valid probability distribution, i.e., its output is always non-negative, and it integrates to 1.[1] Classical examples of MMs with non-negative parameters are Gaussian mixtures (GMMs) and hidden Markov models [Rabiner and Juang, 1986].



(a) Samples (b) GMM-2 (c) GMM-16 (d) NGMM-2

Figure 1: **Negative mixture models can be more expressive than their non-negative counterpart**. Density functions estimated from points sampled from a ring-shaped distribution (a) by a Gaussian Mixture Model (GMM) with 2 and 16 components (figures (b) and (c)) and a GMM allowing negative weights (NGMM) with 2 components (d). See Appendix G.1 for details.

However, imposing a convex combination over Θ is not a necessary constraint to model valid distributions. Relaxing it can yield more expressive MMs that require far fewer components, and hence parameters, to fit complex distributions (Fig. 1). While appealing, MMs allowing negative parameters – called negative MMs (NMMs) or non-monotonic models [Shpilka and Yehudayoff, 2010] – are harder to represent and learn, as one would need to find alternative ways to enforce the non-negativity and integrability of $p(\mathbf{X})$.

So far, this problem has been investigated for simple parametric mixture families such as negative mixture of Weibulls [Jiang et al., 1999] and GMMs for which one can derive constraints in closed form [Zhang and Zhang, 2005], or apply a tensor power method to enforce them [Rabusseau and Denis, 2014]. In this paper, we propose a generic strategy to represent valid NMMs: squaring a base mixture that supports negative parameters (Section 2).

We then extend this approach to to compactly represent deep NMMs within the framework of probabilistic circuits

[1]We abuse the integral notation also for discrete variables.

Accepted for the 6th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2023).

(PCs). There, deep NMMs are encoded as tensorized computational graphs with structural properties that allow efficient squaring (Section 3). We theoretically prove that PCs with negative parameters, also called *non-monotonic circuits*, constructed in this way can encode distributions with exponentially fewer parameters than their *monotonic* counterpart, i.e., PCs with only non-negative parameters (Theorem 3.1).

We complement this generic algorithmic recipe for more flexible MMs and PCs with the *signed-log-sum-exp trick* (Section 3.1), a way to perform computations in log-space while allowing for negative values, which is crucial to retain numerical stability for very deep circuits. Finally, our experiments (Section 4) support the advantage of introducing negative parameters for both shallow and deep MMs.

2 NEGATIVE MIXTURES BY SQUARING

We define a squared NMM with K components as a possibly unnormalized distribution over variables $\mathbf{X}$ encoding

$$c^2(\mathbf{X}) = \left(\sum\nolimits_{i=1}^{K} \theta_i c_i(\mathbf{X})\right)^2 = \sum_{i=1}^{K}\sum_{j=1}^{K} \theta_i\theta_j c_i(\mathbf{X})c_j(\mathbf{X}),$$

where c_i are mixture components and the mixture parameters $\theta_i \in \mathbb{R}$ are unconstrained. Note that by construction c^2 is always non-negative, but can support negative parameters, and it can be written as a shallow mixture having $\binom{K}{2}$ components, each computing some $c_i(\mathbf{X})c_j(\mathbf{X})$ with possibly negative mixture parameters $2\theta_i\theta_j$ if $i \neq j$ and θ_i^2 otherwise. Furthermore, $c^2(\mathbf{X})$ can be easily renormalized as to model the distribution $p(\mathbf{X}) = c^2(\mathbf{X})/Z$, where $Z = \int c^2(\mathbf{x})d\mathbf{X} = \sum_{i=1}^{K}\sum_{j=1}^{K} \theta_i\theta_j \int c_i(\mathbf{x})c_j(\mathbf{x})d\mathbf{X}$ denotes the normalization constant. Computing Z requires evaluating $\binom{K}{2}$ integrals over products of components.

Therefore, we require that components c_i are chosen from a parameteric family that guarantees that their product can be tractably integrated. This property is satisfied for exponential families [Seeger, 2005]: e.g., the product of two Gaussians (resp. Categorical) distributions is another Gaussian [Rasmussen and Williams, 2005] (resp. Categorical), up to one normalization constant that can be computed efficiently. However, note that we are *not* restricted to components that model valid density functions, and allowing a wider class of (even negative) functions can increase the expressivity of MMs (see Section 3 and Appendix C).

Learning squared negative MMs can be done by *maximum-likelihood estimation* (MLE). Given a set $\mathcal{D}$ of training examples, we can write the MLE objective as

$$\sum\nolimits_{\mathbf{x}\in\mathcal{D}} \log(c^2(\mathbf{x})/Z) = -|\mathcal{D}| \log Z + 2\sum\nolimits_{\mathbf{x}\in\mathcal{D}} \log|c(\mathbf{x})|.$$

Therefore, materializing the squared mixture with $\binom{K}{2}$ components is needed only for computing Z or, more generally, to perform marginalization.

3 DEEP NEGATIVE MIXTURES VIA SQUARED PCS

While this squared formulation allows for NMMs that can represent more components than MMs within the same parameter budget, they are still limited to shallow mixtures. In this section we generalize them to deep mixtures within the framework of *circuits* (PCs) [Vergari et al., 2020], structured computational graphs allowing to recursively "stack" different mixtures. By representing deep NMMs as PCs we will be able to precisely characterize when we can allow negative parameters and keep inference tractable.

A circuit c is a parametrized computational graph encoding a function $c(\mathbf{X})$ and consisting of three kinds of computational units: *input*, *product* and *sum* units. An input unit n models a simple, and tractable, function l_n over a predefined set of variables, also called its *scope* and denoted as $\mathsf{sc}(n) \subseteq \mathbf{X}$. A sum (resp. product) unit n defines a function $\sum_{i\in\mathsf{in}(n)} \theta_i c_i(\mathsf{sc}(c_i))$, parameterized by $\theta_i \in \mathbb{R}$ (resp. $\prod_{i\in\mathsf{in}(n)} c_i(\mathsf{sc}(c_i))$), over its scope, defined as the union of the scopes of its inputs, which we denote with $\mathsf{in}(n)$.

A *probabilistic circuit* (PC) is a circuit that encodes a non-negative function , thus representing an *unnormalized distribution*. To efficiently renormalize it, we require the PC to have two structural properties: *smoothness* and *decomposability* (see Definition A.2 and Proposition A.1). Shallow MMs can be readily represented as "flat" smooth and decomposable PCs equipped with a single sum unit. Alternatively, one can "flatten" a deep PC into a shallow mixture with a number of components that is exponential in the depth of the circuit [Choi et al., 2020], hence deep PCs are more succinct than vanilla shallow MMs. Next, we generalize squared NMMs to squared deep circuits.

Tractable squaring of circuits. If we allow a circuit c to support negative parameters (and possibly encode a negative function), we can retrieve a PC by computing its square $c^2 = c \cdot c$. Note that simply introducing a product unit over two copies of c would break decomposability, a crucial property required to renormalize the c^2 (Proposition A.1). Fortunately, we can tractably compute the square and obtain a decomposable representation for c^2 if c is *structured-decomposable*, as defined next.

Definition 3.1 (Structured-decomposability [Pipatsrisawat and Darwiche, 2008, Choi et al., 2020])**.** A circuit c is *structured-decomposable* if (1) it is smooth and decomposable , and (2) any pair of product units n, m in c having the same scope decompose their scope in the same way.

Proposition 3.1 (Tractable square [Vergari et al., 2021])**.** Let c be a structured-decomposable circuit. Constructing the squared circuit c^2 as a decomposable circuit having size $\mathcal{O}(|c|^2)$ can be done in time $\mathcal{O}(|c|^2)$, where $|c|$ denotes size of c (i.e., the number of edges of the computational graph). Moreover, c^2 is also structured-decomposable.

Algorithm 1 squareTensorizedCircuit$(\mathbf{c}, \mathcal{R})$

Input: A structured-decomposable and tensorized circuit $\mathbf{c}$ defined on a region graph $\mathcal{R}$.
Output: The tensorized squared circuit $\mathbf{c}^2$, defined on the same region graph $\mathcal{R}$.
if $\mathcal{R}$ is partitioned into $(\mathcal{R}_l, \mathcal{R}_r)$ **then**
    $\mathbf{c}_l, \mathbf{c}_r \leftarrow$ retrieveSubCircuits$(\mathbf{c}, \mathcal{R}_l, \mathcal{R}_r)$
    $\mathbf{c}_l^2 \leftarrow$ squareTensorizedCircuit$(\mathbf{c}_l, \mathcal{R}_l)$
    $\mathbf{c}_r^2 \leftarrow$ squareTensorizedCircuit$(\mathbf{c}_r, \mathcal{R}_r)$
    $\mathbf{c}^2 \leftarrow$ squareAndConnectLayer$(\mathbf{c}, \mathbf{c}_l^2, \mathbf{c}_r^2)$
else ▷ $\mathcal{R}$ is an input region
    $\mathbf{c}^2 \leftarrow \mathbf{c} \otimes \mathbf{c}$ ▷ $\mathbf{c}$ is a layer of input units
return $\mathbf{c}^2$

Succinctness result. By squaring a structured-decomposable circuit, we increase the number of components of its sum units, thus increasing its expressiveness by a polynomial factor. This works also for *monotonic* circuits [Valiant, 1979], i.e., circuits with non-negative parameters only. However, if we square a *non-monotonic* circuit, i.e., supporting negative parameters, we can obtain a non-monotonic PC which can be exponentially more compact than a monotonic one, as we formally prove in the following theorem.

Theorem 3.1. There is a class of non-negative functions $\mathcal{F}$ over variables $\mathbf{X}$ that can be compactly represented as shallow squared NMMs (and hence squared non-monotonic PCs) but for which the smallest structured-decomposable monotonic PC computing any $F \in \mathcal{F}$ has size $2^{\Omega(|\mathbf{X}|)}$.

We prove it in Appendix B.1 by showing a lower bound for a variant of the unique disjointness problem [Fiorini et al., 2015]. Intuitively, this result tells us that, given a fixed number of parameters, structured-decomposable and non-monotonic PCs can potentially be far more expressive than their monotonic counterparts. Lastly, we observe *there can be no advantage in squaring certain classes of circuits*, e.g., those supporting tractable MAP inference via *determinism* (see Definition A.4), as we prove in Proposition B.3.

Squaring tensorized circuits. A recipe to build modern deep PCs such as RAT-SPNs [Peharz et al., 2019], EiNets [Peharz et al., 2020] and HCLTs [Liu and Van den Broeck, 2021] is to define a hierarchical variable partitioning that dictates how sets of variables recursively factorize, also called *a region graph* [Dennis and Ventura, 2012], denoted as $\mathcal{R}$, and then stack tensorized product and sum layers according to the partitioning defined by $\mathcal{R}$. Algorithm 1 illustrates our simplified algorithm to square a structured-decomposable and tensorized circuit $\mathbf{c}$ defined on a region graph $\mathcal{R}$, as to compactly output $\mathbf{c}^2$ in tensorized form. There, squareAndConnectLayer is a generic function that can be specialized in the case of RAT-SPNs and EiNets or HCLTs, as detailed in Algorithm D.1. Note that this does not increase the number of parameters of the original circuit. Similarly to training NMMs by MLE, evaluating the tensorized $\mathbf{c}^2$ is required only to compute the normalization constant Z which can be done once per batch during training or only once if $\mathbf{c}^2$ is given.

Expressive input polynomials. As anticipated in Section 2, in the framework of squared non-monotonic PCs, we are allowed to use not only distributions such as exponential families, but any (negative) function class whose product can be still tractably integrated. Polynomials defined on fixed intervals are an appealing family for this purpose, as the product of two polynomials is again a polynomial with higher degree that can be integrated efficiently. We experiment with piecewise polynomials such as *B-splines*, which are a linear combination of basis functions. Integrating products of B-splines can be done efficiently with several methods [Mørken, 1991, Vermeulen et al., 1992]. We discuss B-splines as input units for squared non-monotonic PCs in detail in Appendix E.

3.1 THE SIGNED LOG-SUM-EXP TRICK

Modern PC architectures can encode very deep mixture models in computational graphs with billions of product units. Computing probability values in such circuits, and in their squared counterparts, can easily lead to underflow or overflows (see Appendix F). This can be addressed in monotonic PCs by performing computations in the log-space and utilize the log-sum-exp trick [Blanchard et al., 2021]. However, this is not possible for non-monotonic circuits, as the logarithm of negative values is undefined. To solve this, we propose the *signed log-sum-exp trick*, in which we represent non-zero real values $v \in \mathbb{R}_{<>0}$ in terms of the logarithm of their absolute value $\log|v|$ and their sign $\operatorname{sign}(v) \in \{-1, 1\}$, i.e., $v = \operatorname{sign}(v)\exp(\log|v|)$.

Let n be a product unit over $\mathbf{X}$ that computes $c_n(\mathbf{X}) = \prod_{i\in\mathsf{in}(n)} c_i(\mathbf{X}_i)$, for some set of input units $\mathsf{in}(n)$ defined over a partitioning $\mathbf{X}_1, \ldots, \mathbf{X}_{|\mathsf{in}(n)|}$ of $\mathbf{X}$. We can compute $c_n(\mathbf{X})$ in terms of $\log|c_n(\mathbf{X})|$ and $\operatorname{sign}(c_n(\mathbf{X}))$ as

$$\log|c_n(\mathbf{X})| = \sum\nolimits_{i\in\mathsf{in}(n)} \log|c_i(\mathbf{X}_i)|,$$
$$\operatorname{sign}(c_n(\mathbf{X})) = \prod\nolimits_{i\in\mathsf{in}(n)} \operatorname{sign}(c_i(\mathbf{X}_i)).$$

Similarly, let n be a sum unit over $\mathbf{X}$ that computes $c_n(\mathbf{X}) = \sum_{i\in\mathsf{in}(n)} \theta_i c_i(\mathbf{X})$, where i are units over $\mathbf{X}$ and $\theta_i \in \mathbb{R}$. To compute $c_n(\mathbf{X})$ in a numerically stable way, we (i) compute it in terms of $\log|c_n(\mathbf{X})|$ and $\operatorname{sign}(c_n(\mathbf{X}))$, and (ii) leverage the traditional log-sum-exp trick, i.e.,

$$\log|c_n(\mathbf{X})| = \alpha + \log|S|\,, \quad \operatorname{sign}(c_n(\mathbf{X})) = \operatorname{sign}(S) \quad (2)$$

where $S := \sum_{i\in\mathsf{in}(n)} \theta_i \operatorname{sign}(c_i(\mathbf{X}))\exp(\log|c_i(\mathbf{X})| - \alpha)$ with $S \neq 0$, and $\alpha = \max\{\log|c_i(\mathbf{X})|\}_{i\in\mathsf{in}(n)}$. It follows then that $c_n(\mathbf{X}) = \operatorname{sign}(c_n(\mathbf{X}))\exp(\log|c_n(\mathbf{X})|)$, where $\log|c_n(\mathbf{X})|$ and $\operatorname{sign}(c_n(\mathbf{X}))$ are computed as in Eq. (2).

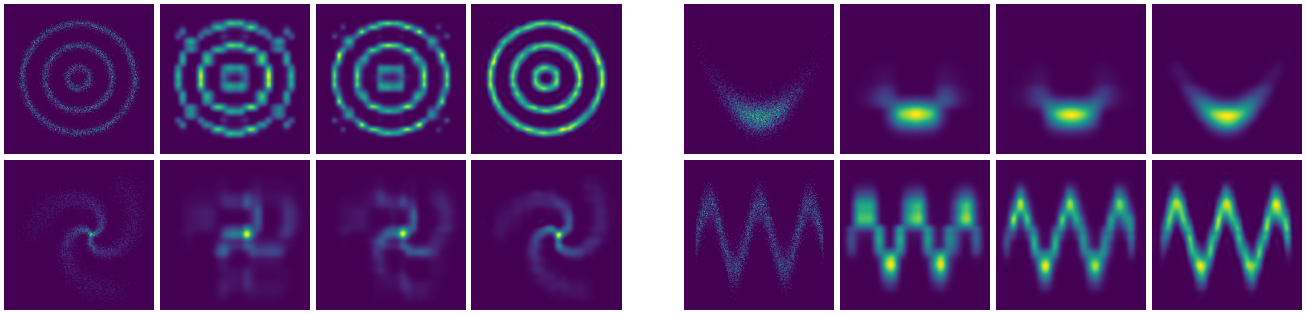
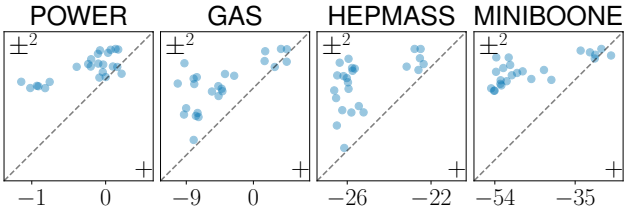


Figure 2: **Negative parameters increase the expressiveness of mixture models.** For each bivariate density, we show the ground truth and the density function estimated by a monotonic PC, a squared monotonic PC, and a squared non-monotonic PC with B-splines as input units and with the same number of parameters.



Model	POWER	GAS	HEPMASS	MINIBOONE
Gaussian	-7.74±0.02	-3.58±0.75	-27.93±0.02	-37.24±1.07
MADE	-3.08±0.03	3.56±0.04	-20.98±0.02	-15.59±0.50
RealNVP	0.17±0.01	8.33±0.14	-18.71±0.02	-13.84±0.52
PC + (G)	0.06±0.01	3.50±0.02	-22.40±0.02	-30.98±0.43
PC + (S)	0.11±0.01	-3.51±0.02	-25.76±0.01	-41.13±0.45
PC $\pm^2$(G)	0.13±0.01	4.78±0.02	-21.31±0.02	-26.98±0.44
PC $\pm^2$(S)	0.41±0.01	2.61±0.01	-21.92±0.01	-31.18±0.46

Figure 3: **Squared non-monotonic PCs are more expressive with the same model size.** Above figures: test average log-likelihoods of squared non-monotonic PCs ($\pm^2$, vertical) and monotonic PCs (+, horizontal) paired by hyperparameters. Below table: best test average log-likelihoods and two standard errors. For PCs we show results given by using either Gaussian (G) or B-splines (S) as input units, and baseline results are taken from Papamakarios et al. [2017].

4 EXPERIMENTS

Since squared non-monotonic PCs would have a higher number of components than monotonic PCs (Section 2), we first need to disentangle the increased number of components given by squaring *and* the presence of negative parameters. Then, we empirically show that squared non-monotonic PCs are more expressive than monotonic PCs for density estimation given a fixed number of parameters.

Higher number of components in squared circuits. We perform experiments on bivariate synthetic data sets by comparing squared non-monotonic PCs and squared PCs whose parameters are enforced to be non-negative via exponentiation. Moreover, we leverage splines as input units for all PCs, as they can encode flexible functions (Section 3). For details about the hyperparameters see Appendix G.4. Fig. 2 shows the density functions estimated by monotonic PCs, their squared version, and squared non-monotonic PCs with the same number of parameters. While squaring monotonic PCs can boosts distribution estimation thanks to the higher number of components, squared non-monotonic PCs are able to almost perfectly capture the data distribution. This result is consistent with the average log-likelihoods computed on unseen samples we show in Appendix G.5.

Squared non-monotonic PCs are more expressive. Next we evaluate both monotonic and squared non-monotonic PCs on data sets taken from the UCI repository [Dua and Graff, 2017]: POWER [Hebrail and Berard, 2012], GAS [Fonollosa et al., 2015], HEPMASS [Baldi et al., 2016] and MINIBOONE [Roe et al., 2004] (see Appendix G.2). In Appendix G.3 we detail the architecture of PCs. We search for several hyperparameters (see Appendix G.4), including the input units to be either Gaussian or B-splines (Section 3). To allow a fair comparison, we evaluate PCs on the same hyperparameters and in terms of log-likelihood computed on unseen samples. In addition, we compare against a single multivariate Gaussian and two deep generative models: MADE [Germain et al., 2015] and RealNVP [Dinh et al., 2016]. Fig. 3 shows that squared non-monotonic PCs estimate the density function better than monotonic PCs on almost every configuration of hyperparameters we choose.

5 CONCLUSIONS

Squared non-monotonic PCs open up several interesting future directions in tractable probabilistic modeling. First, we plan to efficiently learn their structure [Di Mauro et al., 2017] and use them for imposing constraints on neural networks [Ahmed et al., 2022]. Second, we will investigate whether it is possible to distill large-scale monotonic PCs into way smaller non-monotonic ones, as Theorem 3.1 implicitly suggests, as well as other intractable models. Finally, we plant to retrieve a probabilistic semantic for the inner units, as well as to train by EM and to sample efficiently.

Acknowledgements

We thank Raul Garcia-Patron Sanchez for pointing out the relationship of our work with the Born rule in quantum physics. We acknowledge Chris Williams for highlighting the similarities between our work and the literature on estimating the square root of density functions. Finally, we are grateful to Patrick Tourniaire who carried out preliminary experiments on GMMs with negative parameters on 2D data.

References

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=o-mxIWAY1T8.

Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. Parameterized machine learning for high-energy physics. *ArXiv*, abs/1601.07913, 2016.

Pierre Blanchard, Desmond J Higham, and Nicholas J Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330, 2021.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. Technical report, University of California, Los Angeles (UCLA), 2020.

Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48:608–647, 2001.

Carl de Boor. Subroutine package for calculating with b-splines. 1971.

Alexis de Colnet and Stefan Mengel. A compilation of succinctness results for arithmetic circuits. *arXiv preprint arXiv:2110.13014*, 2021.

Ronald De Wolf. Nondeterministic quantum query and communication complexities. *SIAM Journal on Computing*, 32(3):681–699, 2003.

Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. *Advances in Neural Information Processing Systems*, 25, 2012.

Aaron W. Dennis. Algorithms for learning the structure of monotone and nonmonotone sum-product networks, 2016.

Nicola Di Mauro, Antonio Vergari, Teresa MA Basile, and Floriana Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10*, pages 203–219. Springer, 2017.

Laurent Dinh, Jascha Narain Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *ArXiv*, abs/1605.08803, 2016.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald De Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM (JACM)*, 62(2):1–23, 2015.

Jordi Fonollosa, Sadique Sheik, Ramón Huerta, and Santiago Marco. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B-chemical*, 215:618–629, 2015.

Mathieu Germain, Karol Gregor, Iain Murray, and H. Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, 2015.

Nicolas Gillis. *Nonnegative matrix factorization*. SIAM, Philadelphia, 2020.

Ivan Glasser, Ryan Sweke, Nicola Pancotti, Jens Eisert, and Juan Ignacio Cirac. Expressive power of tensor-network factorizations for probabilistic modeling, with applications from hidden Markov models to quantum machine learning. In *Neural Information Processing Systems*, 2019.

Georges Hebrail and Alice Berard. Individual household electric power consumption. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C58K54.

Xia Hong and Junbin Gao. Estimating the square root of probability density function on riemannian manifold. *Expert Systems - The Journal of Knowledge Engineering*, 38(7), 2021.

Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

R Jiang, MJ Zuo, and H-X Li. Weibull and inverse weibull mixture models allowing negative weights. *Reliability Engineering & System Safety*, 66(3):227–234, 1999.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Wenliang Li, Danica J. Sutherland, Heiko Strathmann, and Arthur Gretton. Learning deep kernels for exponential family densities. *ArXiv*, abs/1811.08357, 2018.

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. In *Neural Information Processing Systems*, 2021.

Ulysse Marteau-Ferey, Francis R. Bach, and Alessandro Rudi. Non-parametric models for non-negative functions. In *NeurIPS*, 2020.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014.

Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake. Finite mixture models. *Annual review of statistics and its application*, 6:355–378, 2019.

Knut Mørken. Some identities for products and degree raising of splines. *Constructive Approximation*, 7:195–208, 1991.

Georgii S. Novikov, Maxim E. Panov, and Ivan V. Oseledets. Tensor-train density estimation. In *UAI*, volume 161 of *Proceedings of Machine Learning Research*, pages 1321–1331. AUAI Press, 2021.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. *ArXiv*, abs/1705.07057, 2017.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Xiaoting Shao, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In Amir Globerson and Ricardo Silva, editors, *UAI*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. AUAI Press, 2019.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 7563–7574. PMLR, 2020.

Les A. Piegl and Wayne Tiller. The nurbs book. In *Monographs in Visual Communication*, 1995.

Aluisio Pinheiro and Brani Vidakovic. Estimating the square root of a density via compactly supported wavelets. *Computational Statistics and Data Analysis*, 25(4):399–415, 1997.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522, 2008.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

Lawrence Rabiner and Biinghwang Juang. An introduction to hidden Markov models. *ieee assp magazine*, 3(1):4–16, 1986.

Guillaume Rabusseau and François Denis. Learning negative mixture models by tensor decompositions. *arXiv preprint arXiv:1403.4224*, 2014.

Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian processes for machine learning (adaptive computation and machine learning). 2005.

Byron P. Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments & Methods in Physics Research Section A-accelerators Spectrometers Detectors and Associated Equipment*, 543:577–584, 2004.

Tim Roughgarden et al. Communication complexity (for algorithm designers). *Foundations and Trends® in Theoretical Computer Science*, 11(3–4):217–404, 2016.

Alessandro Rudi and Carlo Ciliberto. PSD representations for effective probability models. In *NeurIPS*, pages 19411–19422, 2021.

Matthias Seeger. Expectation propagation for exponential families. Technical report, 2005.

Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5:207–388, 2010.

Leslie G. Valiant. Negation can be exponentially powerful. In *Proceedings of the eleventh annual ACM symposium on theory of computing*, pages 189–196, 1979.

Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems*, volume 34, pages 13189–13201, 2021.

Allan H. Vermeulen, Richard H. Bartels, and Glenn R. Heppler. Integrating products of b-splines. *SIAM J. Sci. Comput.*, 13:1025–1038, 1992.

Baibo Zhang and Changshui Zhang. Finite mixture models with negative components. In *Machine Learning and Data Mining in Pattern Recognition: 4th International Conference, MLDM 2005, Leipzig, Germany, July 9-11, 2005. Proceedings 4*, pages 31–41. Springer, 2005.

A CIRCUITS

In Definition A.1 we formalize circuits.

Definition A.1 (Circuit [Choi et al., 2020, Vergari et al., 2021])**.** A *circuit* c is a parametrized computational graph over variables $\mathbf{X}$ encoding a function $c(\mathbf{X})$ and comprising three kinds of computational units: *input*, *product*, and *sum*. Each product or sum unit n receives as inputs the outputs of other units, denoted with the set $\mathsf{in}(n)$. Each unit n encodes a function c_n defined as: (i) $l_n(\mathsf{sc}(n))$ if n is an input unit, where l_n is a function over variables $\mathsf{sc}(n) \subseteq \mathbf{X}$, called its *scope*, (ii) $\prod_{i \in \mathsf{in}(n)} c_i(\mathsf{sc}(n_i))$ if n is a product unit, and (iii) $\sum_{i \in \mathsf{in}(n)} \theta_i c_i(\mathsf{sc}(c_i))$ if n is a sum unit, with $\theta_i \in \mathbb{R}$ denoting the weighted sum parameters. The scope of a product or sum unit n is the union of the scopes of its inputs, i.e., $\mathsf{sc}(n) = \bigcup_{i \in \mathsf{in}(n)} \mathsf{sc}(i)$.

A *probabilistic circuit* (PC) is defined as a circuit Definition A.1 that encodes a non-negative function. PCs that are *smooth* and *decomposable* (Definition A.2) enable computing the partition function and, more in general, performing variable marginalization efficiently (Proposition A.1).

Definition A.2 (Smoothness and Decomposability)**.** A circuit is *smooth* if for every sum unit n, its input units depend all on the same variables, i.e, $\forall i, j \in \mathsf{in}(n) \colon \mathsf{sc}(i) = \mathsf{sc}(j)$. A circuit is *decomposable* if the inputs of every product unit n depend on disjoint sets of variables, i.e, $\forall i, j \in \mathsf{in}(n)\ i \neq j \colon \mathsf{sc}(i) \cap \mathsf{sc}(j) = \varnothing$.

Proposition A.1 (Tractability [Choi et al., 2020])**.** Let c be a smooth and decomposable circuit over variables $\mathbf{X}$ whose input units can be integrated efficiently. Then for any $\mathbf{Z} \subseteq \mathbf{X}$ and $\mathbf{y}$ an assignment to variables in $\mathbf{X} \setminus \mathbf{Z}$, the quantity $\int c(\mathbf{y}, \mathbf{z}) d\mathbf{z}$ can be computed exactly in time $\Theta(|c|)$, where $|c|$ denotes the size of the circuit.

As anticipated in Section 3, the expressive power of squared non-monotonic and *deterministic* PCs is the same of monotonic PCs. We prove it formally in Appendix B.2 by leveraging the definition of determinism that we show in Definition A.4.

Definition A.3 (Support [Choi et al., 2020])**.** In a circuit the *support* of a computational unit n over variables $\mathbf{X}$ is defined as the set of value assignments to variables in $\mathbf{X}$ such that the output of n is non-zero, i.e., $\mathsf{supp}(n) = \{\mathbf{x} \in \mathsf{val}(\mathbf{X}) \mid c_n(\mathbf{X}) \neq 0\}$.

Definition A.4 (Determinism [Darwiche, 2001])**.** A circuit c is *deterministic* if for any sum unit $n \in c$ its inputs have disjoint *support* (Definition A.3), i.e., $\forall i, j \in \mathsf{in}(n), i \neq j \colon \mathsf{supp}(i) \cap \mathsf{supp}(j) = \varnothing$.

B PROOFS

B.1 EXPONENTIAL SEPARATION

Theorem 3.1. There is a class of non-negative functions $\mathcal{F}$ over variables $\mathbf{X}$ that can be compactly represented as shallow squared NMMs (and hence squared non-monotonic PCs) but for which the smallest structured-decomposable monotonic PC computing any $F \in \mathcal{F}$ has size $2^{\Omega(|\mathbf{X}|)}$.

Proof. For the proof of Theorem 3.1, we start by constructing $\mathcal{F}$ by introducing a variant of the *unique disjointness* (UDISJ) problem, which seems to have first been introduced by De Wolf [2003]. The variant we consider here is defined over graphs, as detailed in the following definition.

Definition B.1 (Unique disjointness function)**.** Consider an undirected graph $G = (V, E)$, where V denotes its vertices and E its edges. To every vertex $v \in V$ we associate a Boolean variable X_v and let $\mathbf{X}_V = \{X_v \mid v \in V\}$ be the set of all these variables. The *unique disjointness function* of G is defined as

$$\mathsf{UDISJ}_G(\mathbf{X}_v) := \left(1 - \sum_{uv \in E} X_u X_v\right)^2. \tag{3}$$

The UDISJ function as a non-monotonic circuit. We will construct $\mathcal{F}$ as the class of functions UDISJ_G for graphs $G \in \mathcal{G}$, where $\mathcal{G}$ is a family of graphs that we will choose later. Regardless of the way the class $\mathcal{G}$ is picked, we can compactly represent UDISJ_G as a squared structured-decomposable (Definition 3.1) and non-monotonic circuit as follows. First, we represent the function $c(\mathbf{X}_V) = 1 - \sum_{uv \in E} X_u X_v$ as sum unit computing $1 \cdot a(\mathbf{X}_V) + (-1) \cdot b(\mathbf{X}_V)$ where

- a is a circuit gadget that realizes an unnormalized uniform distribution over the domain of variables in $\mathbf{X}_V$, i.e., $a(\mathbf{X}_V) = \prod_{v \in V} (\mathbb{1}\{X_v = 0\} + \mathbb{1}\{X_v = 1\})$ where $\mathbb{1}\{X_v = 0\}$ (resp. $\mathbb{1}\{X_v = 1\}$) is an indicator function that outputs 1 when X_v is set to 0 (resp. 1);
- b is another sum unit whose inputs are product units over the input units $\mathbb{1}\{X_u = 1\}, \mathbb{1}\{X_v = 1\}$ if there is an edge uv in G, i.e., $b(\mathbf{X}_V) = \sum_{uv \in E} \mathbb{1}\{X_u = 1\} \cdot \mathbb{1}\{X_v = 1\}$.

Note that b may not be smooth, but we can easily smooth it by adding to every product an additional input that is a circuit gadget similar to a that outputs 1 for any input $\mathbf{X}_{\overline{uv}}$, where $\mathbf{X}_{\overline{uv}} = \mathbf{X}_V \setminus \{X_u, X_v\}$. Since c is structured-decomposable (Definition 3.1), we can easily multiply it with itself to realize c^2 that would be still a structured-decomposable circuit whose size is polynomially bounded as $|c^2| \in \mathcal{O}(|c|^2)$ [Vergari et al., 2021]. In particular, in this case we have that $|c|$ is a polynomial in the number

of variables (or vertices) $|\mathbf{X}_V|$ by the construction above. Furthermore, note that c^2 is non-monotonic as one of its sum unit has negative parameters (i.e., -1) to encode the subtraction in Eq. (3).

The lower bound for monotonic circuits. To prove the exponential lower bound for monotonic circuits in Theorem 3.1, we will use an approach that has been used in several other works [Martens and Medabalimi, 2014, de Colnet and Mengel, 2021]. This approach is based on representing a decomposable circuit (and hence a structured-decomposable one) as a shallow mixture whose components are *balanced products*, as formalized next.

Definition B.2. Let $\mathbf{X}$ be a set of variables. A *balanced decomposable product* over $\mathbf{X}$ is a function from $\mathbf{X}$ to $\mathbb{R}$ that can be written as $f(\mathbf{Y}) \times h(\mathbf{Z})$ where $(\mathbf{Y}, \mathbf{Z})$ is a partitioning of $\mathbf{X}$, f and h are functions to $\mathbb{R}$ and $|\mathbf{X}|/3 \leq |\mathbf{Y}| \leq 2|\mathbf{X}|/3$.

Theorem B.1 ([Martens and Medabalimi, 2014])**.** Let F be a non-negative function over Boolean variables $\mathbf{X}$ computed by a smooth and decomposable circuit c. Then, F can be written as a sum of N balanced decomposable products (Definition B.2) over $\mathbf{X}$, with $N \leq |c|$ in the form[2]

$$F(\mathbf{X}) = \sum_{k=1}^{N} f_k(\mathbf{Y}_k) \times h_k(\mathbf{Z}_k),$$

where $(\mathbf{Y}_k, \mathbf{Z}_k)$ is partitioning of $\mathbf{X}$ for $1 \leq k \leq N$. If c is structured-decomposable, the N partitions $\{(\mathbf{Y}_k, \mathbf{Z}_k)\}_{k=1}^{N}$ are all identical. Moreover, if c is monotonic, then all f_k, h_k only compute non-negative values.

Intuitively, Theorem B.1 tells us that to lower bound the size of c we can lower bound N. To this end, we first encode the UDISJ function (Eq. (3)) as a sum of N balanced products and show the exponential growth of N for a family of graphs. We start with a special case for a representation in the following proposition.

Proposition B.1. Let G_n be a matching of size n, i.e., a graph consisting of n edges none of which share any vertices. Assume that the UDISJ function (Eq. (3)) for G_n is written as a product of balanced partitions

$$\mathsf{UDISJ}_{G_n}(\mathbf{Y}, \mathbf{Z}) = \sum_{k=1}^{N} f_k(\mathbf{Y}) \times h_k(\mathbf{Z}),$$

where for every edge uv in G_n we have that $X_u \in \mathbf{Y}$ and $X_v \in \mathbf{Z}$. Then $N = 2^{\Omega(n)}$.

[2] In Martens and Medabalimi [2014], Theorem 38, this result is stated with $N \leq |c|^2$. The square materializes from the fact that they reduce their circuits to have all their inner units to have exactly two inputs, as we already assume, following de Colnet and Mengel [2021].

To prove the above results, we will make an argument on the rank of the so-called *communication matrix*, also known as the *value matrix*, for a function F and a fixed partition $(\mathbf{Y}, \mathbf{Z})$.

Definition B.3. Let F be a function over $(\mathbf{Y}, \mathbf{Z})$, its communication matrix M_F is a $2^{|\mathbf{Y}|} \times 2^{|\mathbf{Z}|}$ matrix whose rows (resp. columns) are uniquely indexed by assignments to $\mathbf{Y}$ (resp. $\mathbf{Z}$) such that for a pair of index[3] $(i_{\mathbf{Y}}, j_{\mathbf{Z}})$, the entry at the row $i_{\mathbf{Y}}$ and column $j_{\mathbf{Z}}$ in M_F is $F(i_{\mathbf{Y}}, j_{\mathbf{Z}})$.

Example B.2. Let us consider a simple matching on 6 vertices, where $\mathbf{Y}$ correspond to the first 3 vertices, and $\mathbf{Z}$ to the last 3, and where there is an edge between the first, second and third vertices of $\mathbf{Y}$ and $\mathbf{Z}$. The matrix M_F is an 8-by-8 matrix, a row and a column for each assignment of the 3 binary variables associated to each vertex; it is given by

$\mathbf{Y}\backslash\mathbf{Z}$	000	100	010	001	110	101	011	111
000	1	1	1	1	1	1	1	1
100	1	0	1	1	0	0	1	0
010	1	1	0	1	0	1	0	0
001	1	1	1	0	1	0	0	0
110	1	0	0	1	1	0	0	1
101	1	0	1	0	0	1	0	1
011	1	1	0	0	0	0	1	1
111	1	0	0	0	1	1	1	4

Note that the name UDISJ comes from the fact that $M_F(i, j) = 0$ if and only if $\mathbf{Y}$ and $\mathbf{Z}$ share a single entry equal to 1.

In the following, we will rely on the following quantity.

Definition B.4 (Non-negative rank)**.** The non-negative rank of a non-negative matrix $A \in \mathbb{R}_+^{m \times n}$, denoted $\mathsf{rank}_+(A)$, is the smallest k such that there exist k nonnegative rank-one matrices $\{A_i\}_{i=1}^{k}$ such that $A = \sum_{i=1}^{k} A_i$. Equivalently, it is the smallest k such that there exists two non-negative matrices $B \in \mathbb{R}_+^{m \times k}$ and $C \in \mathbb{R}_+^{k \times n}$ such that $A = BC$.

Given a function F written as a sum over N decomposable products (see Theorem B.1) over a fixed partition $(\mathbf{Y}, \mathbf{Z})$, we now show that the non-negative rank of its communication matrix M_F (Definition B.3) is a lower bound of N.

Lemma B.1. Let $F(\mathbf{X}) = \sum_{k=1}^{N} f_k(\mathbf{Y}) \times h_k(\mathbf{Z})$ where f_k and h_k are non-negative functions and let M_F be the communication matrix (Definition B.3) of F for the partition $(\mathbf{Y}, \mathbf{Z})$, then it holds that

$$\mathsf{rank}_+(M_F) \leq N.$$

Proof. This proof is an easy extension of the proof of Lemma 13 from de Colnet and Mengel [2021]. Assume

[3] An index $i_{\mathbf{Y}}$ (resp. $j_{\mathbf{Z}}$) is a complete assignment to Boolean variables in $\mathbf{Y}$ (resp. $\mathbf{Z}$). See Theorem B.2.

w.l.o.g. that $f_k(\mathbf{Y}) \times h_k(\mathbf{Z}) \neq 0$ for any complete assignment to $\mathbf{Y}$ and $\mathbf{Z}$.[4] Let M_k denote the communication matrix of the function $f_k(\mathbf{Y}) \times h_k(\mathbf{Z})$. By construction, we have that $M_F = \sum_{k=1}^{N} M_k$. Furthermore, since all values in M_F are non-negative by definition, $\mathsf{rank}_+(M_k)$ is defined for all k and by sub-additivity of the non-negative rank we have that $\mathsf{rank}_+(M_F) \leq \sum_{k=1}^{N} \mathsf{rank}^+(M_k)$. To conclude the proof, it is sufficient to show that M_k are rank-1 matrices, i.e., $\mathsf{rank}_+(M_k) = 1$. To this end, consider an arbitrary k. Since $f_k(\mathbf{Y}) \times h_k(\mathbf{Z}) \neq 0$, there is a row in M_k that is not a row of zeros. Say it is indexed by $i_{\mathbf{Y}}$, then its entries are of the form $f_k(i_{\mathbf{Y}}) \times h_k(j_{\mathbf{Z}})$ for varying $j_{\mathbf{Z}}$. In any other rows indexed by $i'_{\mathbf{Y}}$ we have $f_k(i'_{\mathbf{Y}}) \times h_k(j_{\mathbf{Z}}) = (f_k(i'_{\mathbf{Y}})/f_k(i_{\mathbf{Z}})) \times f_k(i_{\mathbf{Y}}) \times h_k(j_{\mathbf{Z}})$ for varying $j_{\mathbf{Z}}$. Consequently, all rows are non-negative multiples of the $i_{\mathbf{Y}}$ row, and therefore $\mathsf{rank}_+(M_k) = 1$. □

To complete the proof of Proposition B.1, we leverage a known lower bound of the non-negative rank of the communication matrix of the UDISJ problem. The interested reader can find more information on this result in the books [Roughgarden et al., 2016, Gillis, 2020] and the references therein.

Theorem B.3 ([Fiorini et al., 2015])**.** Let a UDISJ function defined as in Proposition B.1, and M_{UDISJ} be its communication matrix over a partition $(\mathbf{Y}, \mathbf{Z})$, then it holds that

$$(3/2)^n \leq \mathsf{rank}_+(M_{\mathsf{UDISJ}}).$$

Using Theorem B.3 and Lemma B.1, we directly get Proposition B.1. So we have shown that, for a fixed partition of variables $(\mathbf{Y}, \mathbf{Z})$, every monotonic circuit c encoding the UDISJ function (Eq. (3)) of a matching of size n has size $|c| \geq 2^{\Omega(n)}$. However, the smallest non-monotonic circuit encoding the same function has polynomial size in n (see the construction of the UDISJ function as a circuit above). Now, to complete the proof for the exponential lower bound in Theorem 3.1, we need to find a function class $\mathcal{F}$ where this result holds for all possible partitions $(\mathbf{Y}, \mathbf{Z})$. Such function class consists of UDISJ functions over a particular family of graphs, as detailed in the following proposition.

Proposition B.2. There is a family of graphs $\mathcal{G}$ such that for every graph $G_n = (V_n, E_n) \in \mathcal{G}$ we have $|V_n| = |E_n| = \mathcal{O}(n)$, and any monotonic structured-decomposable circuit representation of UDISJ_{G_n} has size $2^{\Omega(n)}$.

Proof. We prove it by constructing a class of so-called *expander graphs*, which we introduce next. We say that a graph $G = (V, E)$ has expansion ε if, for every subset V' of V of size at most $|V|/2$, there are at least $\varepsilon|V'|$ edges from V' to $V \setminus V'$ in G. It is well-known, see e.g. Hoory et al. [2006], that there are constants $\varepsilon > 0$ and $d \in \mathbb{N}$ and a family $(G_n)_{n\in\mathbb{N}}$ of graphs such that G_n has at least n vertices, expansion ε and maximal degree d. We fix such a family of graphs in the remainder and denote by V_n, resp. E_n, the vertex set, resp. the edge set, of G_n.

Let c be a monotonic structured-decomposable circuit of size N computing UDISJ_{G_n}. Then, by using Theorem B.1, we can write it as

$$\mathsf{UDISJ}_{G_n}(\mathbf{Y}, \mathbf{Z}) = \sum_{k=1}^{N} f_k(\mathbf{Y}) \times h_k(\mathbf{Z}) \tag{4}$$

where $(\mathbf{Y}, \mathbf{Z})$ is a balanced partition of $\mathbf{X}_V$. Let $V_{\mathbf{Y}} = \{v \in V_n \mid X_v \in \mathbf{Y}\}$ and $V_{\mathbf{Z}} = \{v \in V_n \mid X_v \in \mathbf{Z}\}$. Then $(V_{\mathbf{Y}}, V_{\mathbf{Z}})$ form a balanced partition of V_n. By the expansion of G_n, it follows that there are $\Omega(n)$ edges from vertices in $V_{\mathbf{Y}}$ to vertices in $V_{\mathbf{Z}}$. By greedily choosing some of those edges and using the bounded degree of G_n, we can construct an edge set E'_n of size $\Omega(n)$ that is a matching between $\mathbf{Y}$ and $\mathbf{Z}$, i.e., all edges in E'_n go from $\mathbf{Y}$ to $\mathbf{Z}$ and every vertex in V_n is incident to only one edge in E'_n. Let V'_n be the set of endpoints in E'_n and $\mathbf{X}_{V'_n} \subseteq \mathbf{X}_V$ be the variables associated to them. We construct a new circuit c' from c by substituting all input units for variables X_v that are not in $\mathbf{X}_{V'_n}$ by 0. Clearly, $|c'| \leq |c|$ and hence all the lower bounds for $|c'|$ are lower bounds for $|c|$. Let $\overline{\mathbf{Y}} = \mathbf{X}_{V'_n} \cap \mathbf{Y}$ and $\overline{\mathbf{Z}} = \mathbf{X}_{V'_n} \cap \mathbf{Z}$. By construction c' computes the function

$$\mathsf{UDISJ}_{G'_n}(\overline{\mathbf{Y}}, \overline{\mathbf{Z}}) = \left(1 - \sum_{uv \in E'_n} X_u X_v\right)^2$$

which corresponds to solving the UDISJ problem over the graph $G'_n = (V'_n, E'_n)$. From Eq. (4) we get that

$$\mathsf{UDISJ}_{G'_n}(\overline{\mathbf{Y}}, \overline{\mathbf{Z}}) = \sum_{k=1}^{N} f'_k(\overline{\mathbf{Y}}) \times h'_k(\overline{\mathbf{Z}}),$$

where f'_k (resp. h'_k) are obtained from f_k (resp. h_k) by setting all the variables not in $\mathbf{X}_{V'_n}$ to 0. Since c' is monotonic by construction and $|E'_n| = \Omega(n)$, from Proposition B.1 it follows that $N = 2^{\Omega(n)}$. □

Proposition B.2 concludes the proof of Theorem 3.1, as we showed the existence of family of graphs for which the smallest structured-decomposable monotonic circuit computing the UDISJ function over n variables has size $2^{\Omega(n)}$. However, the smallest structured-decomposable non-monotonic circuit has size polynomial in n, whose construction has been detailed at the beginning of our proof.

□

[4] If this were not the case we could simply drop the term from the summation, which would clearly reduce the number of summands.

B.2 SQUARING DETERMINISTIC CIRCUITS

In Proposition B.3 we show that squaring any non-monotonic, smooth and decomposable (Definition A.2), and deterministic (Definition A.4) circuit translates to squaring its parameters and the functions modeled by its input units. Moreover, the sum units of the squared circuit would have the same number of components. As a consequence, there would be no advantage in terms of expressivity given by squaring deterministic circuits. Therefore, we are interested in squaring *non*-deterministic circuits.

Proposition B.3 (Squaring deterministic circuits)**.** Let c be a non-monotonic, smooth and decomposable (Definition A.2), and deterministic (Definition A.4) circuit over variables $\mathbf{X}$. The squared circuit c^2 can be obtained by squaring the parameters in c and the functions modeled by its input units. Moreover, the number of components of sum units does not increase and therefore $|c^2| = |c|$.

Proof. The proof is by induction. Let $n \in c$ be a product unit that computes $c_n(\mathbf{Z}) = \prod_{i \in \mathsf{in}(n)} c_n(\mathbf{Z}_i)$, with $\mathbf{Z} \subseteq \mathbf{X}$ and $\mathbf{Z}_1, \dots, \mathbf{Z}_{|\mathsf{in}(n)|}$ forming a partitioning of $\mathbf{Z}$. Then its squaring computes $c_n(\mathbf{Z})^2 = \prod_{i \in \mathsf{in}(n)} c_n(\mathbf{Z}_i)^2$. Now consider a sum unit $n \in c$ that computes $c_n(\mathbf{Z}) = \sum_{i \in \mathsf{in}(n)} \theta_i c_i(\mathbf{Z})$ with $\mathbf{Z} \subseteq \mathbf{X}$ and $\theta_i \in \mathbb{R}$. Then its squaring computes $c_n(\mathbf{Z})^2 = \sum_{i \in \mathsf{in}(n)} \sum_{j \in \mathsf{in}(n)} \theta_i \theta_j c_i(\mathbf{Z}) c_j(\mathbf{Z})$. Since c is deterministic (Definition A.4), for any i, j with $i \neq j$ either $c_i(\mathbf{Z})$ or $c_j(\mathbf{Z})$ is zero for any assignment to $\mathbf{Z}$. Therefore, we have that

$$c_n(\mathbf{Z})^2 = \sum_{i \in \mathsf{in}(n)} \theta_i^2 c_i(\mathbf{Z})^2. \tag{5}$$

The base case is defined on an input unit n that models a function l_n, and hence its squaring is an input unit that models l_n^2. By induction c^2 is constructed from c by squaring the parameters of sum units θ_i and squaring the functions l_n modeled by input units. Moreover, the number of components of the squared sum units remain the same, as we observe in Eq. (5), and therefore $|c^2| = |c|$. □

C RELATED WORKS

Squared circuits and the Born rule. In this work we present a generic framework to derive a class of NMMs and non-monotonic PCs without imposing constraints on the parameters: by *squaring* (Section 3). Squaring a circuit in order to retrieve a probability distribution is related to the Born rule in quantum mechanics [Novikov et al., 2021], which is applied in Born machines [Glasser et al., 2019]. Born machines can be seen as squared circuits in which product and sum units encode tensor contractions and operate on real (or complex) numbers. However, while Born machines require specifying a total ordering of variables, the framework of circuits permits to model possibly more complex interactions between variables via the specification of a region graph (Section 3), which can also be learned from data [Poon and Domingos, 2011].

Shallow negative MMs. The problem of ensuring that a negative MM still encodes a valid probability distribution has been investigated for some particular parametric distributions, and this usually involve imposing constraints on the parameters. Jiang et al. [1999] derived sufficient conditions on the shape and scale of Weibull distributions to obtain a valid MM, whilst allowing for negative mixture parameters. An analogous result for Gaussian components was given by Zhang and Zhang [2005], who showed how to learn a valid negative GMM by enforcing closed form constraints on the component covariance matrices. In the case of spherical Gaussian components, Rabusseau and Denis [2014] proposed a technique based on the tensor power method to estimate the parameters of a valid negative GMM from data. Rather than estimating the probability density function p, some works investigated the problem of estimating its square root $\sqrt{p}$ with a (possibly negative) function f such that $p(\mathbf{X}) \propto f^2(\mathbf{X})$, with f being a linear combination of basis functions [Pinheiro and Vidakovic, 1997]. Moreover, Hong and Gao [2021] showed that the parameters optimization problem for estimating $\sqrt{p}$ can be formulated as a Riemannian optimization for computational advantage. All the probabilistic models mentioned above are *shallow*, and scaling them to high-dimensional data represents a major challenge. By contrast, our framework provides a recipe for building *deep* non-monotonic PCs. In addition, we show their increased expressivity with respect to traditional MMs both theoretically (Theorem 3.1) and empirically (Section 4) on high-dimensional data.

Other non-monotonic PCs. Dennis [2016] proposed a non-monotonic PC architecture consisting of a subtraction of two monotonic PCs (implemented with a single sum unit with 1 and -1 as parameters) sharing the same structure. Pairwise constraints defined on the non-negative parameters ensure that the overall PC encodes a non-negative function. Similarly to the aforementioned NMMs above, learning while ensuring such constraints are satisfied is not straightforward. That is, it requires ad-hoc methods based on parametrization choices of the input units. By contrast, our framework enables us to learn non-monotonic PCs via unconstrained gradient descent.

Relationship with PSD models. Positive semi-definite (PSD) models generalize monotonic mixture models and sport several advantages including tractable marginalization and allowing negative parameters [Marteau-Ferey et al., 2020, Rudi and Ciliberto, 2021]. Similarly to NMMs mentioned above, they represent a class of *shallow* mixture models. The relationship between PSD models and the proposed framework is that PSD models can be compactly repre-

sented as a mixture of squared non-monotonic PCs. This means that non-monotonic PCs must be at least as expressive as PSD models. More formally, a PSD model typically defines a non-negative function f over variables $\mathbf{X}$ of the form $f(\mathbf{x}; \mathbf{W}, \phi) = \phi(\mathbf{x})^T \mathbf{W} \phi(\mathbf{x})$, where $\mathbf{W} \in \mathbb{R}^{k \times k}$ is a PSD matrix and $\phi : \mathsf{val}(\mathbf{X}) \to \mathbb{R}^k$ is a feature map from the domain $\mathsf{val}(\mathbf{X})$ of variables $\mathbf{X}$ to $\mathbb{R}^k$. Let $\mathbf{W} = \sum_{i=1}^{m} \lambda_i \mathbf{w}_i \mathbf{w}_i^T$ be the eigendecomposition of $\mathbf{W}$, with $m = \mathsf{rank}(\mathbf{W})$. Then, f can be rewritten as

$$f(\mathbf{x}; \mathbf{W}, \phi) = \phi(\mathbf{x})^T \left(\sum_{i=1}^{m} \lambda_i \mathbf{w}_i \mathbf{w}_i^T \right) \phi(\mathbf{x})$$
$$= \sum_{i=1}^{m} \lambda_i (\mathbf{w}_i^T \phi(\mathbf{x}))^2$$

where $\lambda_i > 0$ for $1 \leq i \leq m$. As such, f can be seen as a mixture having m squared NMMs as components, which can be compactly expressed within the framework of squared non-monotonic PCs (Section 3).

D SQUARING TENSORIZED LAYERS

Algorithm D.1 squareAndConnectLayer$(\mathbf{c}, \mathbf{c}_l^2, \mathbf{c}_r^2)$

1: **Input:** A tensorized circuit $\mathbf{c}$ and two squared tensorized circuits $\mathbf{c}_l^2, \mathbf{c}_r^2$.
2: **Output:** The tensorized squared circuit $\mathbf{c}^2$.
3: **if** $\mathbf{c} = \mathbf{W}(\mathbf{c}_l \odot \mathbf{c}_r)$, with $\mathbf{W} \in \mathbb{R}^{k \times k}$ **then**
4: Let $\mathbf{c}^2 = \mathbf{W}(\mathbf{c}_l^2 \odot \mathbf{c}_r^2)\mathbf{W}^T$
5: **else if** $\mathbf{c} = \mathbf{W}(\mathrm{vec}(\mathbf{c}_l \otimes \mathbf{c}_r))$, with $\mathbf{W} \in \mathbb{R}^{k \times k^2}$ **then**
6: Let $\mathbf{c}^2 = \mathbf{W}(\mathbf{c}_l^2 \otimes \mathbf{c}_r^2)\mathbf{W}^T$
7: **return** $\mathbf{c}^2$

Given a structured-decomposable and tensorized circuit $\mathbf{c}$ defined on a region graph [Poon and Domingos, 2011], Algorithm 1 illustrates how $\mathbf{c}^2$ is constructed. This algorithm depends on how layers of sum and product units are connected in $\mathbf{c}$. Algorithm D.1 shows how squared tensorized layers are constructed for common deep PC architectures.

In deep PC architectures such as RAT-SPNs [Peharz et al., 2019] and EiNets [Peharz et al., 2020], a tensorized sum-product layer firstly computes *cross-wise products* of paired tensorized inputs, and then computes multiple weighted sums (as in a mixture). More formally, let $\mathbf{c}_l, \mathbf{c}_r \in \mathbb{R}^k$ be the *left* and *right* tensorized inputs respectively, then the layer computes $\mathbf{W}(\mathrm{vec}(\mathbf{c}_l \otimes \mathbf{c}_r))$ where $\mathbf{W} \in \mathbb{R}^{k \times k^2}$ denotes the parameters of the sum unit and vec the vectorization operator. On the other hand, in HCLTs [Liu and Van den Broeck, 2021] a tensorized sum-product layer computes *element-wise products* of their inputs instead. More formally, using the same notation above, it computes $\mathbf{W}(\mathbf{c}_l \odot \mathbf{c}_r)$ where $\mathbf{W} \in \mathbb{R}^{k \times k}$.

In our experiments we make use of element-wise products, as squaring tensorized layers that compute cross-wise products would be more computationally expensive. In particular, the time complexity to compute the matrix multiplications in Algorithm D.1 in case of element-wise products is $\mathcal{O}(k^3)$, while for cross-wise products is $\mathcal{O}(k^5)$.

E SPLINES AS EXPRESSIVE AND TRACTABLE COMPONENTS

Polynomials defined on fixed intervals are candidate functions to be modeled by input units of squared non-monotonic PCs, due to the fact they can be negative and their product can be tractably integrated. In particular, we experiment with piecewise polynomials, also called *splines*, which are briefly introduced below.

A spline function of order k is a piecewise polynomial defined on a variable X, and the n values of X where polynomials meet are called *knots*. *B-splines* of order k are basis functions for continuous spline functions of the same degree. In practice, we can represent any spline function f of order k defined over n knots inside an interval $[a, b)$ as a linear combination of $k + n$ basis functions, i.e.,

$$f(X) = \sum_{i=1}^{k+n} \alpha_i B_{i,k}(X) \tag{6}$$

where $\alpha_i \in \mathbb{R}$ are the parameters of the spline and $B_{i,k}(X)$ are polynomials of order k (i.e., the basis of f), which are unequivocally determined by the choice of the n knots. In particular, every $B_{i,k}(X)$ is a non-negative function that is recursively defined with the Cox-de-Boor formula [de Boor, 1971, Piegl and Tiller, 1995].

Given two splines f, g of order k defined over n knots and represented in terms of basis functions as in Eq. (6), we can write the integral of their product as follows.

$$\int_a^b f(X) g(X) dX = \sum_{i=1}^{k+n} \sum_{j=1}^{k+n} \alpha_i \beta_j \int_a^b B_{i,k}(X) B_{j,k}(X) \tag{7}$$

where $\alpha_i \in \mathbb{R}$ (resp. $\beta_j \in \mathbb{R}$) denote the parameters of f (resp. g). Therefore, integrating product of splines requires integrating products of their basis functions, which are still polynomials. Among the various way of computing Eq. (7) exactly [Vermeulen et al., 1992], we can do it in time $\mathcal{O}(n^2 \cdot k^2)$ by representing the product $B_{i,k}(X) B_{j,k}(X)$ as another polynomial of order $2k + 1$ and then integrating it in the interval $[a, b)$.

F PARTITION FUNCTION OVERFLOW

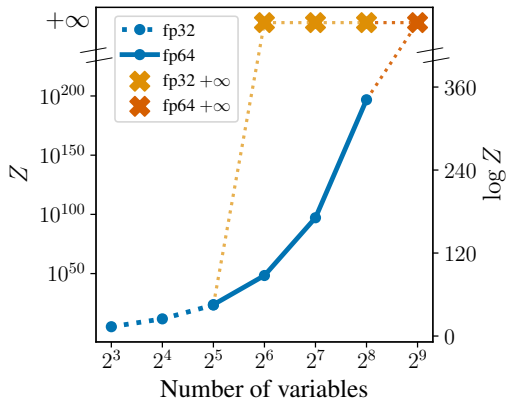


Figure F.1: **Squared non-monotonic PCs cannot scale without performing computations in log-space**. Partition functions (and their *natural* logarithm) of squared non-monotonic PCs with increasing number of variables V computed using 32-bit and 64-bit floating point arithmetic. The number of layers of the circuits are $\lfloor \log_2 V \rfloor$.

In Fig. F.1 we show the partition function values Z of squared non-monotonic PCs with Gaussian distributions as input units and with increasing number of variables. The parameters of the sum units are initialized by sampling from a normal distribution $\mathcal{N}(0, 1)$. Scaling squared non-monotonic PCs to more than a few tens (resp. hundreds) of variables without performing computations in log-space is infeasible in 32-bit (resp. 64-bit) floating point arithmetic. Even if we initialized all parameters to be < 1 in absolute value to make Z smaller, we would encounter underflows during circuit evaluation. The bottom line is that we have to perform computations in log-space even in presence of negative values, and this can be done with the *signed log-sum-exp trick* (see Section 3.1).

G EXPERIMENTS

G.1 RING SYNTHETIC DATASET

For the experiments showed in Fig. 1, we sample $10'000/2'000/2'000$ ring-shaped synthetic training/validation/test examples. Then, we show the density functions estimated by two Gaussian Mixture Models (GMM) with 2 and 16 components, respectively, and a negative GMM (NGMM) with two components. All mixture models are learned by maximizing the log-likelihood and by gradient ascent with Adam as optimizer [Kingma and Ba, 2015], with learning rate $5 \cdot 10^{-3}$ and batch size 16.

G.2 SYNTHETIC AND UCI DATASETS

Following [Li et al., 2018] we demonstrate the behavior of PCs on both synthetic and UCI datasets (see Table G.1). The four synthetic data sets showed in Fig. 2 are *Rings*, *Banana*, *Spiral* and *Wave* (from left to right). We generate each synthetic data set by sampling $10\_000/2000/2000$ training/validation/test examples.

Table G.1: **Data set statistics.** Dimensionality D and number of examples N for each data set after the preprocessing by [Papamakarios et al., 2017].

		N		
	D	train	validation	test
POWER	6	1,659,917	184,435	204,928
GAS	8	852,174	94,685	105,206
HEPMASS	21	315,123	35,013	174,987
MINIBOONE	43	29,556	3,284	3,648

G.3 MODEL DETAILS

For the experiments on UCI data sets we follow Peharz et al. [2019] and randomly derive a structured-decomposable and balanced region graph for PCs. We do so by randomly and evenly splitting sets of variables into binary partitions recursively, and until further splitting is not possible or would make the region graph unbalanced. Then, monotonic PCs and squared non-monotonic PCs are constructed by parametrizing such region graph with layers of sum and product units. We construct multiple PCs on different random region graphs, and then build a monotonic mixture with them as components. See Appendix G.4 for details about the hyperparameters.

The tensorized layers in monotonic PCs and squared non-monotonic PCs compute element-wise products, due to their efficiency as detailed in Appendix D.

G.4 HYPERPARAMETERS

In this section we detail the hyperparameters used for our experiments (Section 4). We train PCs by maximizing the log-likelihood and by gradient ascent with Adam as optimizer [Kingma and Ba, 2015]. We represent the parameters of sum units in monotonic PCs in log-space and initialize them by sampling from a Gaussian $\mathcal{N}(0, 1)$. By contrast, the parameters of sum units in squared non-monotonic PCs are initialized by sampling from a Log-Normal distribution $\mathcal{LN}(0, 1)$. While the parameters of squared non-monotonic PCs are initialized to be non-negative, they become negative during training.

Hyperparameters for synthetic data sets. We fix the number of input units for each feature (i.e., the components) to

16 for all PCs, and use univariate B-splines of degree 2 (i.e., we use quadratic splines) as input units defined on 32 knots that are chosen uniformly in the domain space. For all PCs we fix the batch size to 512 and learning rate to 10^{-2}.

Hyperparameters for UCI data sets. For the experiments on UCI data sets we sample region graphs randomly and instantiate tensorized PCs whose sum units have 32 components (for both monotonic and non-monotonic PCs). Following [Peharz et al., 2019], for POWER, GAS, HEPMASS, MINIBOONE we repeat this process 2, 4, 8, 16 times, and put the resulting PCs as components in a monotonic mixture. For PCs with B-splines as input units (Appendix E) we fix the degree to 2 and we search for the number of knots in $\{128, 256, 512\}$, which are uniformly chosen in the domain space. For all PCs we search for the batch size in $\{512, 1024, 2048\}$ and learning rate in $\{10^{-3}, 10^{-2}\}$.

G.5 LOG-LIKELIHOODS ON SYNTHETIC DATASETS

In Table G.2 we show the average log-likelihoods computed on unseen samples of the synthetic data sets showed in Fig. 2 (from left to right).

Table G.2: **Negative parameters make squared non-monotonic PCs expressive.** Test average log-likelihoods on synthetic data sets of monotonic PCs ($+$), squared monotonic PCs ($+^2$) and squared non-monotonic PCs ($\pm^2$).

Model	Rings	Banana	Spiral	Wave
PC $+$	-1.959	-2.489	-2.402	-2.045
PC $+^2$	-1.823	-2.463	-2.314	-1.922
PC $\pm^2$	**-1.729**	**-2.395**	**-2.189**	**-1.869**