Encoding Negative Dependencies in Probabilistic Circuits

Aleksanteri M. Sladek[1] **Martin Trapp**[1] **Arno Solin**[1]

[1]Computer Science Department, Aalto University, Espoo, Finland

Abstract

Tractability is considered key to trustworthy decision-making under uncertainty, but it often comes at the expense of the ability to represent large families of probability distributions. Probabilistic circuits promise to remedy this by representing tractable yet expressive probabilistic models through hierarchical compositions of tractable distributions subject to certain structural and parameter constraints. A common parameter constraint enforced in these models is non-negativity of the weights, which has been shown by prior work to potentially hinder their *expressive efficiency*. In this work, we propose allowing for negative weights in probabilistic circuits by loosening the non-negativity constraint to a positive semidefinite constraint. We empirically show that probabilistic circuits with *positive semidefinite parameterized nodes* have increased expressive efficiency, whilst retaining tractability, and empirically outperform circuits with non-negative weight constraints.

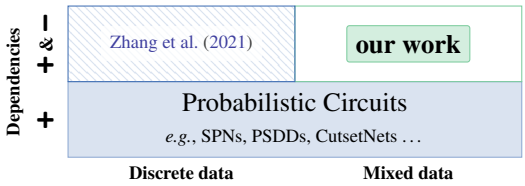

Figure 1: Our extends the realm of tractable probabilistic circuits encoding positive and negative dependencies to mixed (discrete and continuous) domains.

1 INTRODUCTION

Decision-making under uncertainty is a challenging task in modern machine learning (Ghahramani, 2015) in which models that allow for accurate *probabilistic inference* are required. Inference in large and flexible *probabilistic models* quickly becomes intractable, and often require approximate inference (Koller and Friedman, 2009). For example, in deep probabilistic models such as variational auto encoders (VAEs, Kingma and Welling, 2014) and diffusion models (Sohl-Dickstein et al., 2015), marginalisation and computation of moments can only be approximated. However, in safety-critical applications where accuracy of inferences are crucial (*e.g.* in the medical domain), employing approximations can be problematic. These concerns are at the heart of the study of *tractable probabilistic models* (TPMs), which aims to develop models that allow *efficient and exact* computation of probabilistic inferences, whilst simultaneously remaining capable of efficiently representing diverse classes of probabilistic distributions.

Probabilistic circuits (PCs, Choi et al., 2020), a unifying framework for a variety of TPMs (*e.g.*, Darwiche, 2000; Poon and Domingos, 2011; Trapp et al., 2019; Kisa et al., 2014; Rahman et al., 2014), presents a promising avenue in the study as they allow a concise analysis of tractable representations and their structural properties.

Constraints on the structure of PCs determine the tractable band (*i.e.*, which query classes can be answered exactly and efficiently), however, they come at the cost of *expressive efficiency* (Delalleau and Bengio, 2011; Martens and Medabalimi, 2014) (*i.e.*, which families of probability distributions can be represented efficiently). Moreover, recent work (Dennis, 2016; Zhang et al., 2020) has shown that *nonnegative* weight constraint in PCs hinder their ability to represent large families of distributions efficiently.

Allowing for negative parameterizations in TPM models is a promising avenue for pushing the frontier of tractability and expressive efficiency. It has been empirically (Dennis, 2016) and theoretically (Zhang et al., 2021) shown to improve expressive efficiency. In particular, Dennis (2016) proposed a structure and parameter learning approach, which aims to incorporate negative parameters while maintaining pos-

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

itivity of the output function. Further, Zhang et al. (2021) approached the problem differently by formalizing the circuit over *generating functions*, which resulted in a flexible model over binary data. However, neither of the approaches encompasses both continuous and discrete domains nor enable common gradient based optimization.

Independently of the work on TPMs, work in kernel methods introduced a novel family of tractable models defined as *positive semidefinite* (PSD) functions (*e.g.*, Marteau-Ferey et al., 2020; Rudi and Ciliberto, 2021; Muzellec et al., 2022; Tsuchida et al., 2023), which can equally be understood as kernel-sum-of-squares or families of squared neural models. Notably, those allow for negative parameterizations, have shown to improve expressive efficiency compared to a positive paramerization (Marteau-Ferey et al., 2020; Rudi and Ciliberto, 2021), and empirically outperform models with nonegative constraints. However, to obtain an efficient form of the normalization constant these models are chosen to be *shallow*, limiting their expressivity and applicability in relevant applications (Tsuchida et al., 2023).

In this work, we bridge between advancements in kernel methods and TPMs by bringing PSD parameterizations into the framework of PCs. Our work loosens the non-negative constraint in PCs, enabling more effective modelling of complex probability distributions. Moreover, by bringing PSD parameterizations into PCs and exploiting *compatibility* (Vergari et al., 2021) in structured-decomposable (Pipatsrisawat and Darwiche, 2008) PCs we extend the paradigm of squared neural families (Tsuchida et al., 2023) to *deep* probabilistic models. Hence, our work closes an important gap in the PC literature (*cf.*, Fig. 1) and bridges two worlds by integrating PSD parameterizations into structured-decomposable PCs. Lastly, we empirically show that our work outperforms PCs with nonnegativity constraints.

2 PRELIMINARIES

We use X to denote random variables and x to denote a realization. Further, we use $\mathsf{S}, \mathsf{P}, \mathsf{L}$ to denote sum, product and leaf nodes, respectively, and use N for a generic node. Further, we use $\mathrm{ch}(\cdot)$ to denote a function returning the children of a node. Parameters are denoted by θ, while we use $\boldsymbol{A}$ to indicate a PSD matrix ($\boldsymbol{A} \in \mathbb{S}^n_+$). Lastly, we use **bold font** to denote vectors/matrices and *calligraphic* letters for sets.

We briefly review probabilistic circuits and discuss relevant work on PSD parameterized models. Subsequently, we draw connections to PCs and introduce our main contribution.

2.1 PROBABILISTIC CIRCUITS

A probabilistic circuit (PC), is specified by a directed acyclic graph called the computational graph $\mathcal{G}$, the scope function ψ, and parameters $\boldsymbol{\theta}$ containing weights of internal nodes parameterizations of leaf nodes. The computational graph constitutes weighted sums $\mathsf{S}(\boldsymbol{x}) = \sum_{\mathsf{C}\in\mathrm{ch}(\mathsf{S})} \theta_{\mathsf{S},\mathsf{C}}\mathsf{C}(\boldsymbol{x})$, products $\mathsf{P}(\boldsymbol{x}) = \prod_{\mathsf{C}\in\mathrm{ch}(\mathsf{S})} \mathsf{C}(\boldsymbol{x})$, and leaf nodes associated with parametric functions, typically assumed to be density/mass functions of tractable distributions $\mathsf{L}(\boldsymbol{x}) = p(\boldsymbol{x} \mid \theta_{\mathsf{L}})$.

The output of the circuit is computed by evaluating the root node of $\mathcal{G}$. To ensure that the output of the circuit $\mathcal{C}$ is positive for any $\boldsymbol{x}$ in the support set of the PC, *i.e.*, $\mathcal{C}(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{X}$; we typically assume that $\theta_{\mathsf{S},\mathsf{C}} > 0$ for all $\mathsf{S}, \mathsf{C} \in \mathcal{G}$. In addition, each node $\mathsf{N} \in \mathcal{G}$ is associated with a scope $\psi(\mathsf{N}) \subseteq \boldsymbol{X}$ provided by a scope function $\psi\colon \mathsf{N} \to \mathcal{P}(\boldsymbol{X})$ (Trapp et al., 2019), where $\mathcal{P}(\boldsymbol{X})$ denotes the power set of $\boldsymbol{X}$, specifying the set of RVs the node represents a joint distribution over. Fig. 2(a) illustrates an PC over two RVs in which we use $\oplus$ to illustrate sum nodes and $\otimes$ for product nodes.

Next we provide a cursory review of structural properties of PCs relevant to our work. We review the *smoothness* and *decomposability* properties, as they ensure tractable integration. We also review *compatibility* and *structured-decomposability* (Pipatsrisawat and Darwiche, 2008) in PCs. We refer the reader to Choi et al. (2020) for further details.

Definition 2.1 (Smoothness & Decomposability)**.** *A sum node S is **smooth** if all children have the same scope, i.e., $\psi(\mathsf{C}) = \psi(\mathsf{C}'), \forall \mathsf{C}, \mathsf{C}' \in \mathrm{ch}(\mathsf{S})$. Further, a product node P is **decomposable** if all children have pairwise disjoint scopes, i.e., $\psi(\mathsf{C}) \cap \psi(\mathsf{C}') = \emptyset, \forall \mathsf{C}, \mathsf{C}' \in \mathrm{ch}(\mathsf{P})$. A PC is smooth if all sum nodes are smooth and decomposable if all product nodes are decomposable.*

Definition 2.2 (Compatibility)**.** *Two circuits $\mathcal{C}$ and $\mathcal{C}'$ over $\boldsymbol{X}$ are **compatible** if (i) they are smooth and decomposable and (2) any pair of products $\mathsf{P} \in \mathcal{C}$ and $\mathsf{P}' \in \mathcal{C}'$ with the same scope can be rearranged into binary products that are mutually compatible and decompose in the same way.*

*A circuit is **structured-decomposable** if it is **compatible** with itself.*

2.2 POSITIVE SEMIDEFINITE MODELS

Following Rudi and Ciliberto (2021), a probabilistic model expressed as a PSD function or sum of squared functions may be defined in terms of a non-negative function parametrized by a feature map $\phi_\eta\colon \mathbb{R}^d \to \mathcal{H}_\eta$ associated with a kernel function $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$, and a linear operator in the span of $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_n)$, *i.e.*, $\boldsymbol{A} \in \mathbb{S}^n_+$. The resulting model is the written as:

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{A}\, \phi(\boldsymbol{x}) = \textstyle\sum_{i,j}^n A_{i,j}\kappa(\boldsymbol{x}, \boldsymbol{x}_i)\kappa(\boldsymbol{x}, \boldsymbol{x}_j). \tag{1}$$

As Rudi and Ciliberto (2021) show, the resulting function resembles a mixture model (mixture of kernels) if $A_{i,j} = 0$

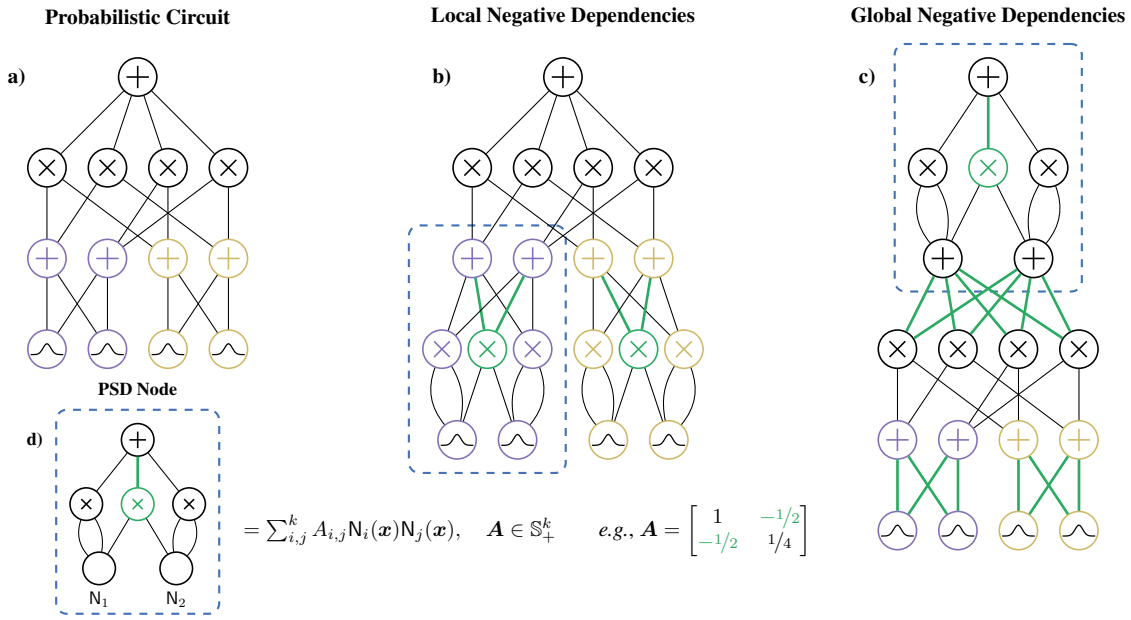

Figure 2: Illustrations of a PC (a) over two RVs coloured in purple and orange, alongside a PC with PSD parameterized nodes at the leaves (b) or the root (c) enabeling local or global negative dependencies, respectively. Note that weights along green edges can have negative parameters in the respective models.

for all $i \neq j$, and can be understood as a *mixture model with negative weights* when $A_{i,j} < 0$ for any $i \neq j$. Furthermore, they showed that the resulting model allows tractable integration, *i.e.*, the normalisation constant can be computed efficiently, discussed the expressive efficiency and optimality for function interpolation under smoothness assumptions, and later work (Muzellec et al., 2022) provided theoretical results for the class of functions.

Tsuchida et al. (2023) extended the paradigm using an equivalent but more general formulation of a squared 2-norm of a shallow neural network. Tractability of the normalisation constant for shallow architectures and various selections of activation functions (*e.g.*, $\sin(\cdot)$ in Meronen et al., 2021) are also discussed.

3 PSD PARAMETERIZED CIRCUITS

We propose bringing PSD parameterizations into PCs enabling: *(i)* negative parameters in PCs; *(ii)* modeling of negative dependencies in continuous and discrete domains; and *(iii)* more expressive efficient representation of probability distributions. To this end, we introduce PSD parameterized nodes which we also refer to as a PSD node.

Definition 3.1 (PSD Parameterized Node)**.** *Let* S *denote a sum node with* k *children, we say that* S *is a PSD parameterized node if it computes the quadratic form*

$$f(\boldsymbol{x}) = \mathbf{N}(\boldsymbol{x})^\top \boldsymbol{A}\, \mathbf{N}(\boldsymbol{x}) = \sum_{i,j}^{k} A_{ij} \mathsf{N}_i(\boldsymbol{x}) \mathsf{N}_j(\boldsymbol{x}), \quad (2)$$

where $\mathbf{N}(\boldsymbol{x}) = [\mathsf{N}_1(\boldsymbol{x}), \ldots, \mathsf{N}_k(\boldsymbol{x})]$ *with* $\mathsf{N}_i \in \mathrm{ch}(\mathsf{S})$ *and* $\boldsymbol{A} \in \mathbb{S}_+^k$*. Note that* $f(\cdot)$ *is not normalized.*

A PSD parameterized node computes by definition a sum of squares. Let $\boldsymbol{U}$ *denote a Cholesky decomposition of* $\boldsymbol{A}$*, i.e.,* $\boldsymbol{A} = \boldsymbol{U}^\top \boldsymbol{U}$*, then:*

$$\begin{aligned} f(\boldsymbol{x}) &= \mathbf{N}(\boldsymbol{x})^\top \boldsymbol{A}\, \mathbf{N}(\boldsymbol{x}) = (\boldsymbol{U}\, \mathbf{N}(\boldsymbol{x}))^\top (\boldsymbol{U}\, \mathbf{N}(\boldsymbol{x})) \qquad (3) \\ &= \left(\textstyle\sum_{i=1}^{k} U_{1,i} \mathsf{N}_i(\boldsymbol{x})\right)^2 + \left(\textstyle\sum_{i=2}^{k} U_{2,i} \mathsf{N}_i(\boldsymbol{x})\right)^2 + \\ &\quad \cdots + \left(U_{k,k} \mathsf{N}_k(\boldsymbol{x})\right)^2. \end{aligned}$$

We note that introducing a PSD parameterization into a node S results in: *(i)* potentially negative parameters, *i.e.*, A_{ij} with $i < j$ can be negative and $U_{ij} \in \mathbb{R}$ with $i < j$; *(ii)* allows tractable computation of the normalization constant if the sub-circuit at S is structured-decomposable (Vergari et al., 2021); and *(iii)* relaxes the nonnegativity constraint of sum nodes below S as the output of S is given as a sum of squares, *i.e.*, guaranteeing positivity of the output value (*cf.* Eq. (3)). We provide details on the computation of the normalization constant in Appendix A.1 alongside an algorithm to compute arbitrary marginals in Appendix A.2.

Depending on the position of the PSD parameterized node in the circuit we can balance computational cost and locality of the negative dependencies. Fig. 2 illustrates different placements of PSD nodes in a PC, resulting in local negative dependencies (if placed close to the leaves) or global negative dependencies (if placed at the root node).

4 EXPERIMENTS

The goal of our experiments is two-fold: *(i)* we aim to provide preliminary results evaluating whether the inclusion of PSD nodes results in greater expressive efficiency for a PC; and *(ii)* explore the effects that introducing PSD nodes in different parts of the PC has.

To this end, we compare three PC models: A PC without PSD nodes, a PC with PSD nodes at the leaves, and a PC with a PSD node at the root. The latter two models are referred to as the local negative dependency (LND) and global negative dependency (GND) PCs respectively, as they reflect introducing *local* or *global* negative dependencies. As such, all models share the same number of input nodes (K) and similar structures.

Experiments were performed over a space of configurations for the models discussed and evaluated on the synthetic data MOONS, WAVES, and RINGS and on toy data GEYSER, IRIS, and WINE (Azzalini and Bowman, 1990; Fisher, 1988; Aeberhard and Forina, 1991). We selected varying number of input nodes $K \in \{2, 4, 8, 16\}$ and five random seeds. As our preliminary implementation[1] is limited two RVs, higher dimensional data were projected to their 1st and 2nd principal components.

Training was performed via full-batch projected gradient descent with a negative log-likelihood (NLL) loss function and the Adam optimizer (Kingma and Ba, 2015). Data sets were split into train and test sets (80/20), with the training set further split into a train and validation set. Early stopping was performed on the validation set, and the final model's NLL was evaluated on the testing set. We note that special care has been taken when evaluating a PC with PSD nodes in log-space as terms in the log-sum-exp operation can be negative, see Appendix A.3 for details.

A curated table of results for the grid experiment can be found in Table 1, with best models highlighted in bold (lower is better). A full experiment results table (with additional baselines) can be found in Table 3. Results shown are based on five random seeds.

In all but one case, the PCs with PSD nodes outperformed their equivalent PCs. An increase in input nodes K generally lead to a better performing model (*cf.*, Table 2), except in few cases in which LND PC overfitted the training data. Results on Geyser are of particular interest as the data set is known to exhibit both negative and positive dependencies (the first and second dimensions are eruption duration at x_t and x_{t+1} respectively Scott, 1992). Overall, the experiment results suggest that PCs with PSD nodes have higher expressive efficiency than PCs with non-negativity constraints on their weights.

[1] Available at www.github.com/AaltoML/psd-pc

Table 1: Average test set NLL scores for the PC, LND PC and GND PC models with K input nodes. The GND and LND PCs outperform the PC in all but one case.

Data Set	K	PC	LND PC	GND PC
MOONS	2	$1.81 \pm .04$	$1.82 \pm .04$	$\mathbf{1.69 \pm .18}$
	4	$1.62 \pm .06$	$\mathbf{1.43 \pm .02}$	$1.46 \pm .13$
	8	$1.56 \pm .03$	$\mathbf{1.25 \pm .04}$	$1.34 \pm .08$
	16	$1.55 \pm .04$	$\mathbf{1.17 \pm .04}$	$1.46 \pm .10$
GEYSER	2	$1.85 \pm .12$	$1.84 \pm .13$	$\mathbf{1.60 \pm .23}$
	4	$1.77 \pm .08$	$1.78 \pm .08$	$\mathbf{1.63 \pm .21}$
	8	$1.83 \pm .15$	$1.81 \pm .08$	$\mathbf{1.62 \pm .13}$
	16	$1.90 \pm .22$	$2.31 \pm .43$	$\mathbf{1.59 \pm .23}$
RINGS	2	$2.71 \pm .01$	$2.70 \pm .01$	$\mathbf{2.60 \pm .12}$
	4	$2.57 \pm .01$	$2.55 \pm .02$	$\mathbf{2.35 \pm .18}$
	8	$2.34 \pm .03$	$2.21 \pm .03$	$\mathbf{1.99 \pm .26}$
	16	$2.29 \pm .01$	$1.95 \pm .05$	$\mathbf{1.93 \pm .14}$
WAVES	2	$3.72 \pm .03$	$3.73 \pm .03$	$\mathbf{3.51 \pm .29}$
	4	$3.40 \pm .01$	$3.38 \pm .02$	$\mathbf{3.15 \pm .17}$
	8	$3.40 \pm .05$	$\mathbf{3.23 \pm .05}$	$3.26 \pm .10$
	16	$3.38 \pm .02$	$\mathbf{3.26 \pm .06}$	$3.32 \pm .06$
IRIS	2	$2.47 \pm .19$	$2.50 \pm .22$	$\mathbf{2.45 \pm .22}$
	4	$2.41 \pm .22$	$2.40 \pm .18$	$\mathbf{2.01 \pm .20}$
	8	$2.59 \pm .21$	$2.56 \pm .18$	$\mathbf{2.47 \pm .24}$
	16	$2.57 \pm .14$	$2.99 \pm .52$	$\mathbf{2.47 \pm .29}$
WINE	2	$2.71 \pm .22$	$2.69 \pm .20$	$\mathbf{2.61 \pm .33}$
	4	$2.75 \pm .26$	$2.83 \pm .21$	$\mathbf{2.51 \pm .33}$
	8	$\mathbf{2.90 \pm .30}$	$3.47 \pm .58$	$3.07 \pm .45$
	16	$2.89 \pm .26$	$3.59 \pm .75$	$\mathbf{2.83 \pm .54}$

5 DISCUSSION & CONCLUSION

In this work, we established a bridge between recent advancements on positive semidefinite (PSD) constrained probabilistic models and probabilistic circuits (PCs) to loosen the non-negativity constraint of sum nodes in PCs and bring PSD parameterizations to deep probabilistic models. Incorporating PSD parameterizations in PCs empirically improves expressive efficiency of PCs with slight increase in computation time (from linear to quadratic), depending on the location of the PSD parameterization in the model. Moreover, our results suggest that leveraging PSD parameterizations are a promising direction to encode positive and negative dependencies on global or local levels and results in improved performance.

Due to the challenge of structure learning, our preliminary implementation is limited to two dimensions. We aim to leverage recent work on structure learning of structured-decomposable PCs in the future and investigate the PC structure with relation to the induced PSD matrix of the shallow mixture expansion of the PC.

Acknowledgements

AMS acknowledges funding from the Helsinki Institute for Information Technology (HIIT). MT acknowledges funding from the Academy of Finland (grant number 347279). AS acknowledges funding from the Academy of Finland (grant numbers 324345 and 339730). We acknowledge the computational resources provided by the Aalto Science-IT project. Finally, we would like to thank our colleagues at the Aalto Machine Learning (AaltoML) laboratory for their feedback and support for this paper.

References

Stefan Aeberhard and Michele Forina. Wine. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5PC7J.

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1027–1035. SIAM, 2007.

Adelchi Azzalini and Adrian W. Bowman. A look at some data on the Old Faithful geyser. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 39(3): 357–365, 1990. ISSN 00359254, 14679876.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic Circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, 2020.

Adnan Darwiche. A differential approach to inference in Bayesian Networks. In Craig Boutilier and Moisés Goldszmidt, editors, *16th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 123–132. Morgan Kaufmann, 2000.

George Datseris, Jonas Isensee, Sebastian Pech, and Tamás Gál. DrWatson: The perfect sidekick for your scientific inquiries. 5(54):2673, 2020. ISSN 2475-9066. doi: 10.21105/joss.02673.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24 (NeurIPS)*, pages 666–674, 2011.

Aaron Dennis. *Algorithms for Learning the Structure of Monotone and Nonmonotone Sum-Product Networks*. PhD thesis, 2016.

Ronald A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. 521(7553):452–459, 2015. doi: 10.1038/nature14541.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations (ICLR)*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations (ICLR)*, 2014.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic Sentential Decision Diagrams. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, KR'14, pages 558–567. AAAI Press, 2014. ISBN 1577356578.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009. ISBN 978-0-262-01319-2.

Ulysse Marteau-Ferey, Francis R. Bach, and Alessandro Rudi. Non-parametric models for non-negative functions. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *33rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 12816–12826, 2020.

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

Lassi Meronen, Martin Trapp, and Arno Solin. Periodic activation functions induce stationarity. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *34the Conference on Neural Information Processing Systems (NeurIPS)*, pages 1673–1685, 2021.

Boris Muzellec, Francis Bach, and Alessandro Rudi. Learning PSD-valued functions using kernel sums-of-squares. *ArXiv preprint arXiv:2111.11306*, 2022.

Knot Pipatsrisawat and Adnan Darwiche. New Compilation Languages Based on Structured Decomposability. In *23rd Conference on Artificial Intelligence (AAAI)*, pages 517–522, 2008.

Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In Fábio Gagliardi Cozman and Avi Pfeffer, editors, *27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 337–346. AUAI Press, 2011.

Tahrima Rahman, Prasanna V. Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, volume 8725 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2014. doi: 10.1007/978-3-662-44851-9_40.

Alessandro Rudi and Carlo Ciliberto. PSD representations for effective probability models. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 19411–19422, 2021.

David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in Probability and Statistics. Wiley, 1992. ISBN 978-0-47154770-9. doi: 10.1002/9780470316849.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis R. Bach and David M. Blei, editors, *32nd International Conference on Machine Learning (ICML)*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2256–2265. JMLR.org, 2015.

Martin Trapp. *Sum-Product Networks for Complex Modelling Scenarios*. PhD thesis, 2019.

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 6344–6355, 2019.

Russell Tsuchida, Cheng Soon Ong, and Dino Sejdinovic. Squared Neural Families: A New Class of Tractable Density Models. *ArXiv preprint arXiv:2305.13552*, 2023.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 13189–13201, 2021.

Honghua Zhang, Steven Holtzen, and Guy Van den Broeck. On the relationship between probabilistic circuits and determinantal point processes. In Ryan P. Adams and Vibhav Gogate, editors, *36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 1188–1197. AUAI Press, 2020.

Honghua Zhang, Brendan Juba, and Guy Van den Broeck. Probabilistic generating circuits. In Marina Meila and Tong Zhang, editors, *38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 12447–12457. PMLR, 2021.

A TECHNICAL DETAILS

A.1 DERIVATION OF NORMALIZATION CONSTANT

This section outlines the computation of the normalization constant z for a PSD parameterized node S in a structured-decomposable circuit, *i.e.*,

$$z = \int_{\mathbb{R}^d} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \tag{4}$$

Computing z recursively decomposes as follows:

Product of Compatible Product Nodes Let p_{P} and p'_{P} be compatible product nodes, we have that:

$$\begin{aligned}
&\int_{\mathbb{R}^{p_{\mathsf{P}}}} f_{\mathsf{P}}(\boldsymbol{x}) f_{\mathsf{P}'}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} && (5)\\
&= \int_{\mathbb{R}^{p_{\mathsf{P}}}} \left(\prod_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} f_{\mathsf{N}}(\hat{\boldsymbol{x}}) \prod_{\mathsf{N}' \in \mathrm{ch}(\mathsf{P}')} f_{\mathsf{N}'}(\hat{\boldsymbol{x}}) \right) \mathrm{d}\boldsymbol{x} && (6)\\
&= \int_{\mathbb{R}^{p_{\mathsf{P}}}} \left(\prod_{(\mathsf{N},\mathsf{N}') \in \Delta_{\mathsf{P}\times\mathsf{P}'}} f_{\mathsf{N}}(\hat{\boldsymbol{x}}) f_{\mathsf{N}'}(\hat{\boldsymbol{x}}) \right) \mathrm{d}\boldsymbol{x} && (7)\\
&= \prod_{(\mathsf{N},\mathsf{N}') \in \Delta_{\mathsf{P}\times\mathsf{P}'}} \int_{\mathbb{R}^{p_{\mathsf{N}}}} f_{\mathsf{N}}(\hat{\boldsymbol{x}}) f_{\mathsf{N}'}(\hat{\boldsymbol{x}}) \mathrm{d}\hat{\boldsymbol{x}} && (8)
\end{aligned}$$

where $\Delta_{\mathsf{P}\times\mathsf{P}'}$ denotes the diagonal of the Cartesian product of the children of P and P', *i.e.*, $\mathrm{diag}(\mathrm{ch}(\mathsf{P}) \times \mathrm{ch}(\mathsf{P}'))$, and compatibility ensures that both product nodes apply the same partition of the scope $\psi(\mathsf{P}) = \psi(\mathsf{P}')$ with parts in the same order.

Product of Sum Nodes Given two sum nodes, we have that:

$$\begin{aligned}
&\int f_{\mathsf{S}}(\boldsymbol{x}) f_{\mathsf{S}'}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} && (9)\\
&= \int \left(\left[\sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} f_{\mathsf{N}}(\hat{\boldsymbol{x}}) \right] \right. && (10)\\
&\qquad \left. \left[\sum_{\mathsf{N}' \in \mathrm{ch}(\mathsf{S}')} w_{\mathsf{S}',\mathsf{N}'} f_{\mathsf{N}'}(\boldsymbol{x}) \right] \right) \mathrm{d}\boldsymbol{x} && (11)\\
&= \int \left(\sum_{\substack{\mathsf{N} \in \mathrm{ch}(\mathsf{S}) \\ \mathsf{N}' \in \mathrm{ch}(\mathsf{S}')}} w_{\mathsf{S},\mathsf{N}} w_{\mathsf{S}',\mathsf{N}'} f_{\mathsf{N}}(\hat{\boldsymbol{x}}) f_{\mathsf{N}'}(\boldsymbol{x}) \right) \mathrm{d}\boldsymbol{x} && (12)\\
&= \sum_{\substack{\mathsf{N} \in \mathrm{ch}(\mathsf{S}) \\ \mathsf{N}' \in \mathrm{ch}(\mathsf{S}')}} w_{\mathsf{S},\mathsf{N}} w_{\mathsf{S}',\mathsf{N}'} \int f_{\mathsf{N}}(\hat{\boldsymbol{x}}) f_{\mathsf{N}'}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} && (13)
\end{aligned}$$

utilizing *smoothness* and *linearity* of sum nodes.

Remark A.1. *In the special case of univariate Gaussian leaf nodes, f_{L} is in the form*

$$f_{\mathsf{L}}(x) = k_{\eta_{\mathsf{L}}}(x, x_{\mathsf{L}}) = exp\left(-\eta_{\mathsf{L}}[x - x_{\mathsf{L}}]^2\right) \tag{14}$$

In this case Equation the product of two leaf nodes has a closed form solution as follows:

$$\begin{aligned}
\int f_{\mathsf{L}}(x) f_{\mathsf{L}'}(x) &= \int k_{\eta_{\mathsf{L}}}(x, x_{\mathsf{L}}) k_{\eta_{\mathsf{L}'}}(x, x_{\mathsf{L}'}) \mathrm{d}x && (15)\\
&= k_{\frac{\eta_{\mathsf{L}}\eta_{\mathsf{L}'}}{\eta_{\mathsf{L}}+\eta_{\mathsf{L}'}}}(x_{\mathsf{L}}, x_{\mathsf{L}'}) \int k_{\eta_{\mathsf{L}}+\eta'_{\mathsf{L}}}(x, x_{\mathsf{L}\mathsf{L}'}) \mathrm{d}x && (16)\\
&= k_{\frac{\eta_{\mathsf{L}}\eta_{\mathsf{L}'}}{\eta_{\mathsf{L}}+\eta_{\mathsf{L}'}}}(x_{\mathsf{L}}, x_{\mathsf{L}'}) \pi^{\frac{1}{2}} (\eta_{\mathsf{L}} + \eta_{\mathsf{L}'})^{-\frac{1}{2}} && (17)
\end{aligned}$$

where $x_{\mathsf{L}\mathsf{L}'} = \frac{x_{\mathsf{L}}\eta_{\mathsf{L}} + x_{\mathsf{L}'}\eta_{\mathsf{L}'}}{\eta_{\mathsf{L}}+\eta_{\mathsf{L}'}}$. Note that $x_{\mathsf{L}\mathsf{L}'}$ need not be computed as it is a translation term with no effect on the integral evaluation. The last equation utilizes the closed form solution for the integral of a Gaussian kernel:

$$\int k_{\eta}(x, x') \mathrm{d}x = \left(\frac{\pi}{\eta}\right)^{\frac{1}{2}}$$

◁

Integral of the Product of m 1D Gaussian Kernels In general, we need to compute the integral of the product of m leaves. We will focus our derivation on Gaussian kernel functions, but similar results can be obtain for cases discussed in Tsuchida et al. (2023). The following derivation is a generalization of derivation F.1 shown in Rudi and Ciliberto (2021)

We wish to find the integral:

$$\begin{aligned}
\int \prod_{i=1}^{m} f_{\mathsf{L}}(x) \mathrm{d}x &= \int \prod_{i=1}^{m} k_{\eta_i}(x, x_i) \mathrm{d}x && (18)\\
&= \int e^{-\sum_{i=1}^{m} \eta_i (x - x_i)^2} \mathrm{d}x && (19)
\end{aligned}$$

Pulling out and expanding the exponent (highlighted in blue):

$$= x^2 \left(\sum_{i=1}^{m} \eta_i \right) - 2x \left(\sum_{i=1}^{m} \eta_i x_i \right) + \sum_{i=1}^{m} \eta_i x_i^2 \tag{20}$$

Taking $\sum_{i=1}^{m} \eta_i$ as a common factor of the first two terms:

$$= \left(\sum_{i=1}^{m} \eta_i \right) \left(x^2 - 2x \left[\frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i} \right] \right) + \sum_{i=1}^{m} \eta_i x_i^2 \tag{21}$$

Completing the square in the rightmost parenthesis

$$\begin{aligned}
&= \left(\sum_{i=1}^{m} \eta_i \right) \left(x^2 - 2x \left[\frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i} \right] \right. && (22)\\
&\quad \left. - \left[\frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i} \right]^2 + \left[\frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i} \right]^2 \right) + \sum_{i=1}^{m} \eta_i x_i^2 && (23)
\end{aligned}$$

Expanding out the product and cancelling

$$= \left(\sum_{i=1}^{m} \eta_i\right) \left(\left[x - \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right]^2 - \left[\frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right]^2\right) \tag{24}$$

$$+ \sum_{i=1}^{m} \eta_i x_i^2 \tag{25}$$

Expanding out the product of the parenthesis, resulting in cancellation of the square of one of the denominators

$$= \left(\sum_{i=1}^{m} \eta_i\right) \left(x - \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right)^2 \tag{26}$$

$$+ \sum_{i=1}^{m} \eta_i x_i^2 - \frac{\left(\sum_{i=1}^{m} \eta_i x_i\right)^2}{\sum_{i=1}^{m} \eta_i} \tag{27}$$

Pulling out and simplifying the last two terms by unifying the denominators (highlighted in orange):

$$= \frac{\left(\sum_{i=1}^{m} \eta_i\right) \sum_{i=1}^{m} \eta_i x_i^2}{\sum_{i=1}^{m} \eta_i} - \frac{\left(\sum_{i=1}^{m} \eta_i x_i\right)^2}{\sum_{i=1}^{m} \eta_i} \tag{28}$$

$$= \frac{1}{\sum_{i=1}^{m} \eta_i} \left(\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \eta_i \eta_j x_i^2 - \sum_{\substack{i,j=1 \\ i \neq j}}^{m} \eta_i \eta_j x_i x_j\right) \tag{29}$$

Grouping together pairs of squares and pairwise products and combining them into differences of two squares:

$$= \frac{1}{\sum_{i=1}^{m} \eta_i} \left[\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \eta_i \eta_j (x_i - x_j)^2 * \mathbb{1}(j > i)\right] \tag{30}$$

Noting that the last equation's summation represents a summation over elements with indices i, j contained within the triangular matrix above the diagonal of the Cartesian product of index set $\mathcal{I} = \{1, \ldots, m\}$ with itself. In other words, a summation over elements indexed by all two-element subsets in set $\mathcal{I}$.

Adding Eq. (30) back into Eq. (27):

$$= \left(\sum_{i=1}^{m} \eta_i\right) \left(x - \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right)^2 \tag{31}$$

$$+ \frac{1}{\sum_{i=1}^{m} \eta_i} \left(\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \eta_i \eta_j (x_i - x_j)^2\right) \tag{32}$$

Adding the exponent back and splitting into a product of two exponentials:

$$= \exp\left\{\left(-\sum_{i=1}^{m} \eta_i\right) \left(x - \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right)^2\right\} \tag{33}$$

$$* \exp\left\{-\frac{1}{\sum_{i=1}^{m} \eta_i} \left(\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \eta_i \eta_j (x_i - x_j)^2\right)\right\} \tag{34}$$

Formulating as new Gaussian kernels:

$$= k_{\sum_{i=1}^{m} \eta_i}\left(x, \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right) \prod_{\substack{i,j=1 \\ i \neq j}}^{m} \left[k_{\frac{\eta_i \eta_j}{\sum_{k=1}^{m} \eta_k}}(x_i, x_j)\right] \tag{35}$$

Adding the integral back in:

$$= \int \prod_{\substack{i,j=1 \\ i \neq j}}^{m} \left[k_{\sum_{i=1}^{m} \eta_i}\left(x, \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right) k_{\frac{\eta_i \eta_j}{\sum_{k=1}^{m} \eta_k}}(x_i, x_j)\right] \mathrm{d}x \tag{36}$$

$$= \prod_{\substack{i,j=1 \\ i \neq j}}^{m} \left[k_{\frac{\eta_i \eta_j}{\sum_{k=1}^{m} \eta_k}}(x_i, x_j)\right] \tag{37}$$

$$* \int k_{\sum_{i=1}^{m} \eta_i}\left(x, \frac{\sum_{i=1}^{m} \eta_i x_i}{\sum_{i=1}^{m} \eta_i}\right) \mathrm{d}x \tag{38}$$

Substituting the closed form solution for an arbitrary Gaussian function integral (highlighted in blue, see Remark A.1):

$$= \sqrt{\frac{\pi}{\sum_{i=1}^{m} \eta_i}} \prod_{\substack{i,j=1 \\ i \neq j}}^{m} \left[k_{\frac{\eta_i \eta_j}{\sum_{k=1}^{m} \eta_k}}(x_i, x_j)\right] \tag{39}$$

Hence the integral of a product of m Gaussian leaf nodes is:

$$\int \prod_{i=1}^{m} f_{\mathsf{L}}(x) \mathrm{d}x = \sqrt{\frac{\pi}{\sum_{i=1}^{m} \eta_i}} \prod_{\substack{i,j=1 \\ i \neq j}}^{m} \left[k_{\frac{\eta_i \eta_j}{\sum_{k=1}^{m} \eta_k}}(x_i, x_j)\right] \tag{40}$$

A.2 INTEGRATION ALGORITHM

In practice the integration dervivations outlined in Appendix A.1 for computing the normalization constant $z_{\mathcal{C}}$ of a PSD parameterized PC $\mathcal{C}$ can be expressed in a recursive algorithm. One such algorithm is proposed in Algorithm 1.

Note that this algorithm takes two arguments as its input: the circuit $\mathcal{C}$ to be integrated and $\boldsymbol{\lambda}$ defining the set of RVs to be marginalized. For calculating the normalization constant we set $\boldsymbol{\lambda} = \psi(\mathcal{C})$, *i.e.*, all RVs $\mathcal{C}$ operates over. For marginalizing out a specific subset of RVs, $\boldsymbol{\lambda}$ reflects this set and the algorithm refers to it to return the correct marginalization.

The key working principle of Algorithm 1 is modifying the computational graph $\mathcal{G}$ of $\mathcal{C}$ into a new computational graph $\mathcal{G}_z$ which computes $z_\mathcal{C}$, *i.e.*, $f_{\mathcal{G}_z}(\mathcal{C}) = z_\mathcal{C}$. The algorithm begins at the root node of $\mathcal{G}$ and traverses it in a depth-first manner. Its operation can be thought of as a 'squishing' of the computational graph through operations that reduce depth and increase the width of the graph. This is achieved via three key operations, denoted in Algorithm 1 as sub-routines `product-of-sums`, `swap-order` and `product-kernel`.

For example, let the root node of $\mathcal{C}$ be a product node P. If the children of P are sum nodes S and S', then sub-routine `product-of-sums` is called. Following Eq. (9), this computational sub-graph is explicitly expanded into a new computational sub-graph $\mathcal{G}'$, with a sum node S'' as its root. Its children are new product nodes $\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_k$ computing the k pairwise products of the children of S and S'. Consequently, the weights of S'' are the pairwise products of the weights of S and S'. The algorithm then replaces the product node P and its children S and S' in $\mathcal{G}$ with S''.

Note: In Algorithm 1, *PSD parameterized nodes are considered as sum nodes* as they are just a weighted summation operation with specific product node configurations as their children. As such, `product-of-sums` processes them in a similar way to sum nodes. The only difference is, the weights of the resulting new sum node S'' created are a *Kroenecker* product of the weight matrices of the two PSD nodes being multiplied.

Since Algorithm 1 is recursive, the processing of $\mathcal{G}'$ kicks off the processing of sub-graphs in $\mathcal{G}'$. For example, the pairwise products $\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_k$ constructed by `product-of-sums` are passed back to Algorithm 1 for evaluation. If the children of P_i are sum or PSD parameterized nodes, `product-of-sums` is called again. If, however, the children are products P'_i and P''_i, then the algorithm determines via the `decomposable` and `compatible` operators whether further modification of the graph is needed.

In the case of decomposable product nodes, the algorithm need not take further action, as the integrals of products operating over different variables can be computed separately and multiplied. If, however, the products are not decomposable but still compatible, then the algorithm calls the `swap-order` sub-routine. Using commutativity of products, this routine rearranges the children of the product nodes into products over the same RV such that their integrals can be separated. At this point, Algorithm 1 is once more called and the integral computed using the decomposable branch.

As suggested by Appendix A.1, the algorithm eventually winds up with products of input distributions that cannot be further separated. In this case, the sub-routine `product-kernel` is called to merge the products of input nodes into a single node which can be integrated in closed form. For the Gaussian case, this was shown in Eq. (18).

The end result of Algorithm 1 is a graph with integrals at its leaves, which can be passed up the graph to compute the final normalization constant or arbitrary marginals, depending on $\boldsymbol{\lambda}$.

A.3 LOG-SUM-EXP TRICK WITH NEGATIVE WEIGHTS

Computations in probabilistic circuits are typically performed in the log-domain, rather than the linear domain, requiring the application of the log-sum-exp trick for numerical stability. Numerical instability occurs if exponentiating an inner term results in an under- or overflow, which, in case of the floating point number system, is reflected in values being too large/small to be representable given the number of exponent and mantissa bits available. The log-sum-exp trick is typically given as

$$\log \mathsf{S} = c + \log \left(\sum_i^k \exp\left(\log\left(w_i \mathsf{N}_i\right) - c\right) \right), \qquad (41)$$

with $c = \max_i\{\log\left(w_i \mathsf{N}_i\right)\}$. However, in order to perform the log-sum-exp trick every $w_i \mathsf{N}_i$ term has to be positive. For PSD parameterized sum nodes, this is not necessarily the case. Conveniently, however, one can express Eq. (41) in the following form assuming positivity of the result

$$c + \log \left(\sum_i^k (-1)^{s_i} \exp\left(\log\left((-1)^{s_i} w_i \mathsf{N}_i\right) - c\right) \right), \qquad (42)$$

where s_i denotes the sign-bit of the ith term and $c = \max_i\{\log\left(|w_i \mathsf{N}_i|\right)\}$. The above ensures that the exponentiated terms can still be represented given the number of exponent and mantissa bits available. Note that if the resulting sum is negative, the result can be represented as the log of its absolute value and the sign bit may be carried along for further computations.

B EXPERIMENTAL DETAILS

All of our models were defined and all experiments run using the Julia programming language. We also utilized the DrWatson experiment assistant package (Datseris et al., 2020), which provides many useful functionalities for managing a standardized codebase and experiment setup. A total of 700 models were trained on CPUs on a High Performance Computing cluster available for our research.

B.1 MODELS

The models we defined were restricted to operate over two RVs, given the preliminary nature of our experiments. The

Algorithm 1 Integration graph constructor algorithm

Let a PSD parameterized PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\boldsymbol{\lambda}$ denotes the set of RVs to marginalize.

function INTEGRAL-GRAPH($\mathsf{N}, \boldsymbol{\lambda}$)
  $\mathrm{R} \leftarrow \emptyset$
  if $\mathsf{N} : \mathsf{S}$ **then**
    for $\mathsf{C} \in \mathrm{ch}(\mathsf{N})$ **do**
      $\mathrm{R} \leftarrow \mathrm{R} + w_{\mathsf{N},\mathsf{C}}$ INTEGRAL-GRAPH($\mathsf{C}, \boldsymbol{\lambda}$)
    end for
  else if $\mathsf{N} : \mathsf{P}$ **then**
    $\mathsf{C}, \mathsf{C}' \leftarrow \mathrm{ch}(\mathsf{N})$
    if $(\mathsf{C} : \mathsf{S}) \wedge (\mathsf{C}' : \mathsf{S})$ **then**
      $\mathsf{N}' \leftarrow$ PRODUCT-OF-SUMS(N)
      $\mathrm{R} \leftarrow$ INTEGRAL-GRAPH($\mathsf{N}', \boldsymbol{\lambda}$)
    else if $(\mathsf{C} : \mathsf{P}) \wedge (\mathsf{C}' : \mathsf{P})$ **then**
      if $\mathrm{compatible}(\mathsf{C}, \mathsf{C}')$ **then**
        $\mathsf{N}' \leftarrow$ SWAP-ORDER(N)
        for $\mathsf{C} \in \mathrm{ch}(\mathsf{N}')$ **do**
          $\mathrm{R} \leftarrow \mathrm{R} \times$ INTEGRAL-GRAPH($\mathsf{C}, \boldsymbol{\lambda}$)
        end for
      else if $\mathrm{decomposable}(\mathsf{C}, \mathsf{C}')$ **then**
        $\mathsf{N}' \leftarrow \mathsf{N}$
        for $\mathsf{C} \in \mathrm{ch}(\mathsf{N}')$ **do**
          $\mathrm{R} \leftarrow \mathrm{R} \times$ INTEGRAL-GRAPH($\mathsf{C}, \boldsymbol{\lambda}$)
        end for
      else
        $\mathsf{N}' \leftarrow$ PRODUCT-KERNEL(N)
        if $\psi(\mathsf{N}') \in \boldsymbol{\lambda}$ **then**
          $\mathrm{R} \leftarrow \int \mathsf{N}' \, \mathrm{d}\psi(\mathsf{N}')$
        else
          $\mathrm{R} \leftarrow \mathsf{N}'$
        end if
      end if
    else if $(\mathsf{C} : \mathsf{L}) \wedge (\mathsf{C}' : \mathsf{L})$ **then**
      $\mathsf{N}' \leftarrow$ PRODUCT-KERNEL(N)
      if $\psi(\mathsf{N}') \in \boldsymbol{\lambda}$ **then**
        $\mathrm{R} \leftarrow \int \mathsf{N}' \, \mathrm{d}\psi(\mathsf{N}')$
      else
        $\mathrm{R} \leftarrow \mathsf{N}'$
      end if
    end if
  else
    $\mathrm{R} \leftarrow \mathsf{N}$
  end if
  return R
end function

full set of models considered were a *Gaussian Mixture Model* (GMM) and the PSD model defined by Rudi and Ciliberto (2021) as baselines, and the PC, *local negative dependency* (LND) PC and *global negative dependency* (GND) PC as the models of interest. The model architectures were formulated as such:

B.2 GMM & PSD MODELS

The GMM model was defined as a common mixture model operating over K input nodes, each of which was a 2D Gaussian kernel. Hence, the GMM was simply a sum node computing a mixture of K 2D kernel functions.

The PSD model was exactly the same, with the exception that it replaces the sum node with a PSD node. Hence, it computed the PSD node function over the K 2D Gaussian kernels. Rather than computing a mixture of K like the GMM, it computed a mixture of K^2 kernel squares and pairwise products.

B.2.1 PC

The PC model consisted of K univariate Gaussian kernels as input distributions (for each RV, i.e $2K$ input nodes in total). The parents of the input nodes were two sum nodes, computing mixtures of the leaves. All pairwise products of mixtures between the two dimensions were computed by K^2 product nodes, over which a further mixture was computed by a sum node at the root.

B.2.2 LND PC

The LND PC model is equivalent to the PC model, with the exception that instead of computing mixtures between univariate leaves via sum nodes, the sum nodes were replaced with PSD nodes. Each PSD node was assigned to operate over a pair of the K input nodes, with PSD nodes not sharing children with one another. Alternatively, the formulation could be seen as one PSD node parameterized with a *block diagonal matrix*, which determines weights for only a subset of all possible pairwise products of the input nodes. As the PSD nodes operated over pairs, each PSD node was parameterized with a 2×2 parameter matrix.

B.2.3 GND PC

The GND PC model is equivalent to the PC model, with an additional sum node added as an additional root. Then, a PSD node was added as a parent to the two sum nodes. Hence, the GND PC computes a PSD function over two mixtures of the children of the PC model's root node.

B.3 DATA SETS

Models were trained on well-known toy data sets in machine learning: MOONS, RINGS, WAVES, GEYSER, IRIS and WINE. The WAVES dataset ($N = 1000$) was simulated via generating 2D points from a uniform circular distribution with radius $r = 3$. Rejection sampling was used to reject points located within $1 < r < 2$, *i.e.* a central blob and a ring surrounding it was generated. The MOONS dataset ($N = 1000$) was generated using the SyntheticDatasets.jl package's `make_moons` function with noise parameter `noise` $= 0.1$. The RINGS dataset ($N = 3999$) was generated using a similar method as the WAVES dataset, except it consists of thinner rings and contains an additional outer ring. The GEYSER data set ($N = 271$) is a pre-processed version of Azzalini and Bowman (1990), processed following Trapp (2019) and Scott (1992). The IRIS data set ($N = 149$) and WINE data set ($N = 177$) were retrieved from the University of California Irvine (UCI) machine learning data set repository (Fisher, 1988; Aeberhard and Forina, 1991). All data sets used had two features, either by default or through projection into 1st and 2nd principal components in higher dimensional data sets (IRIS, WINE) to accomodate our model architecture.

B.4 TRAINING PROCEDURE

The parameters optimized in our experiments included sum node, PSD node and input node parameters. Parameter initialization for each node type was performed as follows: *(i)* sum node weight vector elements were sampled from a uniform distribution, and the vector was normalized to sum to one; *(ii)* PSD node parameter matrices $\boldsymbol{A}$ were initialized with an $m \times m$ identity matrix (m denotes the number of inputs to the node); and *(iii)* input nodes were initialized as 1D Gaussian kernels (see Eq. (14)) with length-scale parameter η initialized to 10, and the offset (or center) parameter x' initialized to a point in the data set using the *k-means++* seeding algorithm (Arthur and Vassilvitskii, 2007).

At the beginning of an experiment, data sets were randomly split into a train, validation and test set. Each model was trained for 1500 epochs using full-batch projected gradient descent with a negative log-likelihood loss function. Projected gradient descent was used to ensure parameters for PSD nodes remained within the space of PSD matrices, and parameters of sum nodes remained nonnegative. For PSD node parameters, projection was done via a Cholesky decomposition of parameter matrix $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{U}^\top$ and taking the log of diagonal elements of $\boldsymbol{U}$ to ensure non-negativity. Hence the optimized parameters of a PSD node consisted of an upper triangular matrix of values with a diagonal of log-scale parameters.

Parameter optimization was done via the Adam optimizer Kingma and Ba (2015), and each experiment was repeated for five random seeds. The validation set was used to save the best performing model during training, and the resulting model was evaluated on the test set.

Experiment results were calculated as the average test set NLL achieved by the models trained over the five different seeds. The full results table is shown in Table 3, and graphs comparing the effect of increasing the number of input distributions to the NLL score achieved can be seen in Table 2.

C EXPERIMENTAL RESULTS

See next page for graphs visualizing the experiment results for the PC, LND PC and GND PC models. The following page provides the full experimental result table with baseline models (GMM and PSD model) included.

Table 2: Graphs showing number of input nodes the model has vs. average NLL test set score achieved for each data set experimented with. Averages computed across the five random seeds. An alternative visualization of results in Table 3.

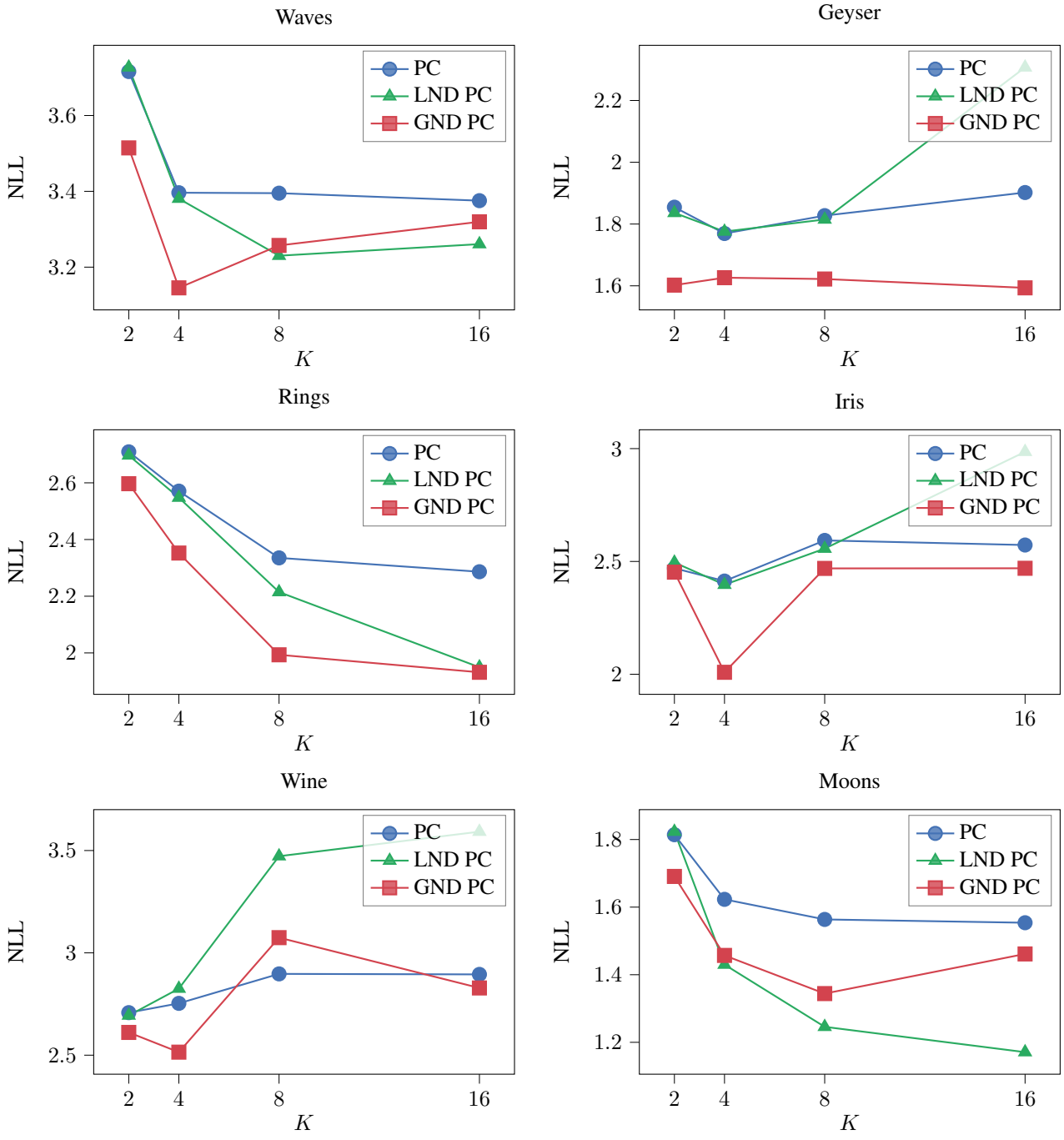

Table 3: Full experiment results table. Table shows the average test set NLL score $\pm$ standard deviation for experiments run over five random seeds. Best performing model for a given basepoint configuration is highlighted in bold (lower is better).

Data Set	K	GMM	PSD-MODEL	PC	LND PC	GND PC
MOONS	2	$1.802 \pm .027$	$1.828 \pm .027$	$1.814 \pm .038$	$1.823 \pm .043$	$\mathbf{1.691 \pm .184}$
	4	$1.550 \pm .039$	$1.554 \pm .022$	$1.623 \pm .058$	$\mathbf{1.430 \pm .020}$	$1.457 \pm .134$
	8	$1.232 \pm .014$	$\mathbf{1.215 \pm .021}$	$1.564 \pm .028$	$1.246 \pm .037$	$1.344 \pm .080$
	16	$\mathbf{1.125 \pm .045}$	$1.130 \pm .037$	$1.554 \pm .040$	$1.171 \pm .044$	$1.461 \pm .096$
GEYSER	2	$2.371 \pm .189$	$2.387 \pm .193$	$1.855 \pm .120$	$1.836 \pm .126$	$\mathbf{1.602 \pm .228}$
	4	$1.842 \pm .213$	$1.841 \pm .221$	$1.769 \pm .083$	$1.776 \pm .080$	$\mathbf{1.626 \pm .212}$
	8	$1.865 \pm .176$	$1.962 \pm .241$	$1.827 \pm .150$	$1.815 \pm .076$	$\mathbf{1.622 \pm .129}$
	16	$1.970 \pm .240$	$2.180 \pm .341$	$1.902 \pm .220$	$2.307 \pm .427$	$\mathbf{1.593 \pm .231}$
RINGS	2	$2.738 \pm .004$	$2.728 \pm .008$	$2.709 \pm .012$	$2.697 \pm .010$	$\mathbf{2.597 \pm .124}$
	4	$2.671 \pm .016$	$2.657 \pm .022$	$2.571 \pm .008$	$2.548 \pm .018$	$\mathbf{2.352 \pm .183}$
	8	$2.477 \pm .034$	$2.328 \pm .062$	$2.335 \pm .029$	$2.215 \pm .033$	$\mathbf{1.993 \pm .260}$
	16	$2.158 \pm .042$	$2.042 \pm .045$	$2.286 \pm .010$	$1.949 \pm .049$	$\mathbf{1.932 \pm .142}$
WAVES	2	$3.883 \pm .068$	$4.046 \pm .183$	$3.716 \pm .025$	$3.727 \pm .034$	$\mathbf{3.515 \pm .293}$
	4	$3.647 \pm .094$	$3.698 \pm .057$	$3.396 \pm .009$	$3.381 \pm .018$	$\mathbf{3.145 \pm .165}$
	8	$3.300 \pm .048$	$3.256 \pm .034$	$3.395 \pm .055$	$\mathbf{3.230 \pm .045}$	$3.257 \pm .101$
	16	$3.224 \pm .033$	$\mathbf{3.216 \pm .032}$	$3.375 \pm .021$	$3.261 \pm .057$	$3.320 \pm .063$
IRIS	2	$2.632 \pm .180$	$2.650 \pm .180$	$2.470 \pm .186$	$2.495 \pm .223$	$\mathbf{2.453 \pm .223}$
	4	$2.309 \pm .104$	$2.347 \pm .114$	$2.413 \pm .224$	$2.397 \pm .183$	$\mathbf{2.009 \pm .204}$
	8	$\mathbf{2.331 \pm .086}$	$2.413 \pm .157$	$2.593 \pm .213$	$2.557 \pm .177$	$2.469 \pm .242$
	16	$\mathbf{2.349 \pm .083}$	$2.715 \pm .366$	$2.573 \pm .143$	$2.986 \pm .525$	$2.470 \pm .292$
WINE	2	$2.806 \pm .270$	$2.823 \pm .246$	$2.708 \pm .217$	$2.694 \pm .200$	$\mathbf{2.611 \pm .335}$
	4	$2.669 \pm .199$	$2.689 \pm .190$	$2.753 \pm .260$	$2.825 \pm .210$	$\mathbf{2.515 \pm .329}$
	8	$\mathbf{2.801 \pm .277}$	$2.841 \pm .403$	$2.897 \pm .300$	$3.472 \pm .581$	$3.074 \pm .450$
	16	$3.119 \pm .334$	$3.484 \pm .565$	$2.895 \pm .261$	$3.592 \pm .747$	$\mathbf{2.828 \pm .541}$