Tractable Regularization of Probabilistic Circuits

Anji Liu
Department of Computer Science
UCLA
Los Angeles, CA 90095
liuanji@cs.ucla.edu

Guy Van den Broeck
Department of Computer Science
UCLA
Los Angeles, CA 90095
guyvdb@cs.ucla.edu

Abstract

Probabilistic Circuits (PCs) are a promising avenue for probabilistic modeling. They combine advantages of probabilistic graphical models (PGMs) with those of neural networks (NNs). Crucially, however, they are tractable probabilistic models, supporting efficient and exact computation of many probabilistic inference queries, such as marginals and MAP. Further, since PCs are structured computation graphs, they can take advantage of deep-learning-style parameter updates, which greatly improves their scalability. However, this innovation also makes PCs prone to overfitting, which has been observed in many standard benchmarks. Despite the existence of abundant regularization techniques for both PGMs and NNs, they are not effective enough when applied to PCs. Instead, we re-think regularization for PCs and propose two intuitive techniques, *data softening* and *entropy regularization*, that both take advantage of PCs' tractability and still have an efficient implementation as a computation graph. Specifically, data softening provides a principled way to add uncertainty in datasets in closed form, which implicitly regularizes PC parameters. To learn parameters from a softened dataset, PCs only need linear time by virtue of their tractability. In entropy regularization, the exact entropy of the distribution encoded by a PC can be regularized directly, which is again infeasible for most other density estimation models. We show that both methods consistently improve the generalization performance of a wide variety of PCs. Moreover, when paired with a simple PC structure, we achieved state-of-the-art results on 10 out of 20 standard discrete density estimation benchmarks. Open-source code and experiments are available at https://github.com/UCLA-StarAI/Tractable-PC-Regularization.

1 Introduction

Probabilistic Circuits (PCs) [1, 2] are considered to be the lingua franca for Tractable Probabilistic Models (TPMs) as they offer a unified framework to abstract from a wide variety of TPM circuit representations, such as arithmetic circuits (ACs) [3], sum-product networks (SPNs) [4], and probabilistic sentential decision diagrams (PSDDs) [5]. PCs are a successful combination of classic probabilistic graphical models (PGMs) and neural networks (NNs). Moreover, by enforcing various structural properties, PCs permit efficient and exact computation of a large family of probabilistic inference queries [6, 7, 8]. The ability to answer these queries leads to successful applications in areas such as model compression [9] and model bias detection [10, 11]. At the same time, PCs are analogous to NNs since their evaluation is also carried out using computation graphs. By exploiting the parallel computation power of GPUs, dedicated implementations [2, 12] can train a complex PC with millions of parameters in minutes. These innovations have made PCs much more expressive and scalable to richer datasets that are beyond the reach of "older" TPMs [13].

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

However, such advances make PCs more prone to overfitting. Although parameter regularization has been extensively studied in both the PGM and NN communities [14, 15], we find that existing regularization techniques for PGMs and NNs are either not suitable or not effective enough when applied to PCs. For example, parameter priors or Laplace smoothing typically used in PGMs, and often used in PC learning as well [16, 17, 18], incur unwanted bias when learning PC parameters – we will illustrate this point in Sec. 3. Classic NN methods such as L1 and L2 regularization are not always suitable since PCs often use either closed-form or EM-based parameter updates.

This paper designs parameter regularization methods that are directly tailored for PCs. We propose two regularization techniques, *data softening* and *entropy regularization*. Both formulate the regularization objective in terms of distributions, regardless of their representation and parameterization. Yet, both leverage the tractability and structural properties of PCs. Specifically, data softening injects noise into the dataset by turning hard evidence in the samples into soft evidence [19, 20]. While learning with such softened datasets is infeasible even for simple machine learning models, with their tractability, a class of PCs (i.e., *deterministic* PCs) can learn the maximum-likelihood estimation (MLE) parameters given a softened dataset in $\mathcal{O}(|p| \cdot |\mathcal{D}|)$ time, where $|p|$ is the size of the PC and $|\mathcal{D}|$ is the size of the (original) dataset. For PCs that are not deterministic, every parameter update step can be done in $\mathcal{O}(|p| \cdot |\mathcal{D}|)$ time, still allowing efficient parameter learning. Additionally, the entropy of the distribution encoded by a PC can be tractably regularized. Although the entropy regularization objective for PC is multi-modal and a global optimum cannot be found in general, we propose an algorithm that is guaranteed to converge monotonically towards a stationary point.

We show that both proposed approaches consistently improve the test set performance over standard density estimation benchmarks. Furthermore, we observe that when data softening and entropy regularization are properly combined, even better generalization performance can be achieved. Specifically, when paired with a simple PC structure, this combined regularization method achieves state-of-the-art results on 10 out of 20 standard discrete density estimation benchmarks.

Notation We denote random variables by uppercase letters (e.g., X) and their assignments by lowercase letters (e.g., x). Analogously, we use bold uppercase letters (e.g., $\mathbf{X}$) and bold lowercase letters (e.g., $\boldsymbol{x}$) for sets of variables and their joint assignments, respectively.

2 Two Intuitive Ideas for Regularizing Distributions

A common way to prevent overfitting in machine learning models is to regularize the syntactic representation of the distribution. For example, L1 and L2 losses add mutually independent priors to all parameters of a model; other approaches such as Dropout [14], Bayesian Neural Networks (BNNs) [21], and Bayesian parameter smoothing [22] incorporate more complex and structured priors into the model [23]. In this section, we ask the question: how would we regularize an arbitrary distribution, regardless of the model at hand, and the way it is parameterized? Such global, model-agnostic regularizers appear to be under-explored. Next, we introduce two intuitive ideas for regularizing distributions, and study how they can be practically realized in the context of probabilistic circuits in the remainder of this paper.

Data softening Data augmentation is a common technique to improve the generalization performance of machine learning models [24, 25]. A simple yet effective type of data augmentation is to inject noise into the samples, for example by randomly corrupting bits or pixels [26]. This can greatly improve generalization as it renders the model more robust to such noise. While current noise injection methods are implemented as a sequence of sampled transformations, we stress that some noise injection can be done in closed form: we will be considering all possible corruptions, each with their own probability, as a function of how similar they are to a training data point.

Consider boolean variables[1] as an example: after noise injection, a sample $X=1$ is represented as a distribution over all possible assignments (i.e., $X=1$ and $X=0$), where the instance $X=1$, which is "similar" to the original sample, gets a higher probability: $P(X=1)=\beta$. Here $\beta \in (0.5, 1]$ is a hyperparameter that specifies the regularization strength — if $\beta=1$, no regularization is added; if β approaches 0.5, the regularized sample represents an (almost) uniform distribution. For a sample $\boldsymbol{x}$ with K variables $\mathbf{X} := \{X_i\}_{i=1}^{K}$, where the kth variable takes value x_k, we can similarly 'soften' $\boldsymbol{x}$

[1] We postpone the discussion on regularizing samples with non-boolean variables in Appendix B.1.

by independently injecting noise into each variable, resulting in a *softened distribution* $P_{\boldsymbol{x},\beta}$:

$$\forall \boldsymbol{x}' \in \mathsf{val}(\mathbf{X}), \quad P_{\boldsymbol{x},\beta}(\mathbf{X}=\boldsymbol{x}') := \prod_{i=1}^{K} P_{\boldsymbol{x},\beta}(X_i=x_i') = \prod_{i=1}^{K} \Big(\beta \cdot \mathbb{1}[x_i'=x_i] + (1-\beta)\cdot \mathbb{1}[x_i' \neq x_i]\Big).$$

For a full dataset $\mathcal{D} := \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$, this softening of the data can also be represented through a new, *softened dataset* $\mathcal{D}_\beta$. Its empirical distribution is the average softened distribution of its data. It is a weighted dataset, where $\texttt{weight}(\mathcal{D}_\beta, \boldsymbol{x})$ denotes the weight of sample $\boldsymbol{x}$ in $\mathcal{D}_\beta$:

$$\mathcal{D}_\beta := \{\boldsymbol{x} \,|\, \boldsymbol{x} \in \mathsf{val}(\mathbf{X})\} \quad \text{and} \quad \texttt{weight}(\mathcal{D}_\beta, \boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} P_{\boldsymbol{x}^{(i)},\beta}(\mathbf{X}=\boldsymbol{x}). \tag{1}$$

This softened dataset ensures that each possible assignment has a small but non-zero weight in the training data. Consequently, any distribution learned on the softened data must assign a small probability everywhere as well. Of course, materializing this dataset, which contains all possible training example, is not practical. Regardless, we will think of data softening as implicitly operating on this softened dataset. We remark that data softening is related to soft evidence [27] and virtual evidence [28], which both define a framework to incorporate uncertain evidence into a distribution.

Entropy regularization Shannon entropy is an effective indicator for overfitting. For a dataset $\mathcal{D}$ with N distinct samples, a perfectly overfitting model that learns the exact empirical distribution has entropy $\log(N)$. A distribution that generalizes well should have a much larger entropy, since it assigns positive probability to exponentially more assignments near the training samples. Concretely, for the protein sequence density estimation task [29] that we will experiment with in Sec. 4.3, the perfectly overfitting empirical distribution has entropy 3, a severely overfitting learned model has entropy 92, yet a model that generalizes well has entropy 177. Therefore, directly controlling the entropy of the learned distribution will help mitigate overfitting. Given a model $P_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$ and a dataset $\mathcal{D} := \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$, we define the following entropy regularization objective:

$$\texttt{LL}_{\text{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau) := \frac{1}{N}\sum_{i=1}^{N} \log P_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \tau \cdot \texttt{ENT}(P_{\boldsymbol{\theta}}), \tag{2}$$

where $\texttt{ENT}(P_{\boldsymbol{\theta}}) := -\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} P_{\boldsymbol{\theta}}(\boldsymbol{x}) \log P_{\boldsymbol{\theta}}(\boldsymbol{x})$ denotes the entropy of distribution $P_{\boldsymbol{\theta}}$, and τ is a hyperparameter that controls the regularization strength. Various forms of entropy regularization have been used in the training process of deep learning models. Different from Eq. (2), these methods regularize the entropy of a parametric [30, 31] or non-parametric [32] output space of the model.

Although both ideas for regularizing distributions are rather intuitive, it is surprisingly hard to implement them in practice since they are intractable even for the simplest machine learning models.

Theorem 1. *Computing the likelihood of a distribution represented as a exponentiated logistic regression (or equivalently, a single neuron) given softened data is #P-hard.*

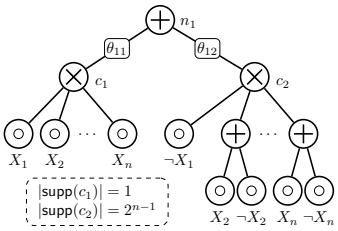
Theorem 2. *Computing the Shannon entropy of a normalized logistic regression model is #P-hard.*

Proof of Thm. 1 and 2 are provided in Appendices A.3 and A.4. Although data softening and entropy regularization are infeasible for many models, we will show in the following sections that they are tractable to use when applied to Probabilistic Circuits (PCs) [1], a class of expressive TPMs.

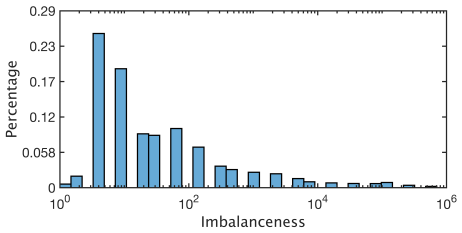
3 Background and Motivation

Probabilistic Circuits (PCs) are a collective term for a wide variety of TPMs. They present a unified set of notations that provides succinct representations for TPMs such as Probabilistic Sentential Decision Diagrams (PSDDs) [5], Sum-Product Networks (SPNs) [4], and Arithmetic Circuits (ACs) [3]. We proceed by introducing the syntax and semantics of a PC.

Definition 1 (Probabilistic Circuits)**.** A PC p that represents a probability distribution over variables $\mathbf{X}$ is defined by a parametrized directed acyclic graph (DAG) with a single root node, denoted n_r. The DAG comprises three kinds of units: *input*, *sum*, and *product*. Each leaf node n in the DAG corresponds to an input unit; each inner node n (i.e., sum and product units) receives inputs from its



(a) A PC with imbalanced sum unit n_1.



(b) Imbalanceness of the PCs learned by Strudel.

Figure 1: A Problem of Laplace smoothing. (a) Laplace smoothing cannot properly regularize this PC as the sum unit n_1 is imbalanced, i.e., its two children have drastically different support sizes. (b) A large fraction of sum units learned by a PC structure learning algorithm [17] are imbalanced.

children, denoted $\mathsf{in}(n)$. Each unit n encodes a probability distribution p_n, defined as follows:

$$p_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & \text{if } n \text{ is an input unit,} \\ \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) & \text{if } n \text{ is a sum unit,} \\ \prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) & \text{if } n \text{ is a product unit,} \end{cases}$$

where f_n is a univariate input distribution (e.g., boolean, categorical or Gaussian), and $\theta_{n,c}$ represents the parameter corresponds to edge (n, c). Intuitively, a sum unit models a weighted mixture distribution over its children, and a product unit encodes a factored distribution over its children. We assume w.l.o.g. that all parameters are positive and the parameters associated with any sum unit n sum up to 1 (i.e., $\sum_{c \in \mathsf{in}(n)} \theta_{n,c} = 1$). We further assume w.l.o.g. that a PC alternates between sum and product layers [33]. The size of a PC p, denoted $|p|$, is the number of edges in its DAG.

This paper focuses on two classes of PCs that support different types of queries: (i) PCs that allow linear-time computation of marginal (MAR) and maximum-a-posterior (MAP) inferences (e.g., PSDDs [5], selective SPNs [34]); (ii) PCs that only permit linear-time computation of MAR queries (e.g., SPNs [4]). The borders between these two types of PCs are defined by their *structural properties*, i.e., constraints imposed on a PC. First, in order to compute MAR queries in linear time, both classes of PCs should be decomposable (Def. 2) and smooth (Def. 3) [1]. These are properties of the (variable) scope $\phi(n)$ of PC units n, that is, the collection of variables defined by all its descendent input nodes.

Definition 2 (Decomposability)**.** A PC is decomposable if for every product unit n, its children have disjoint scopes: $\forall c_1, c_2 \in \mathsf{in}(n)\ (c_1 \neq c_2), \phi(c_1) \cap \phi(c_2) = \varnothing$.

Definition 3 (Smoothness)**.** A PC is smooth if for every sum unit n, its children have the same scope: $\forall c_1, c_2 \in \mathsf{in}(n), \phi(c_1) = \phi(c_2)$.

Next, *determinism* is required to guarantee efficient computation of MAP inference [35].

Definition 4 (Determinism)**.** Define the support $\mathsf{supp}(n)$ of a PC unit n as the set of complete variable assignments $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ for which $p_n(\boldsymbol{x})$ has non-zero probability (density): $\mathsf{supp}(n) = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathsf{val}(\mathbf{X}), p_n(\boldsymbol{x}) > 0\}$. A PC is deterministic if for every sum unit n, its children have disjoint support: $\forall c_1, c_2 \in \mathsf{in}(n)\ (c_1 \neq c_2), \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \varnothing$.

Since the only difference in the structural properties of both PCs classes is determinism, we denote members in the first PC class as deterministic PCs, and members in the second PC class as non-deterministic PCs. Interestingly, both PC classes not only differ in their tractability, which is characterized by the set of queries that can be computed within $\mathtt{poly}(|p|)$ time [6], they also exhibit drastically different expressive efficiency. Specifically, abundant empirical [17, 13] and theoretical [36] evidences suggest that non-deterministic PCs are more expressive than their deterministic counterparts. Due to their differences in terms of tractability and expressive efficiency, this paper studies parameter regularization on deterministic and non-deterministic PCs separately.

Motivation Laplace smoothing is widely adopted as a PC regularizer [16, 17]. Since it is also the default regularizer for classical probabilistic models such as Bayesian Networks (BNs) [37] and Hierarchical Bayesian Models (HBMs) [38], this naturally raises the following question: *are there differences between a good regularizer for classical probabilistic models such as BNs and HBMs and effective regularizers for PCs?* The question can be answered affirmatively — while Laplace

Algorithm 1 Forward pass

Input: A deterministic PC p; sample $\boldsymbol{x}$
Output: $\texttt{value}[n]{:=}(\boldsymbol{x}\in\mathsf{supp}(n))$ for each unit n
foreach n traversed in postorder **do**
  if n **isa** input unit **then** $\texttt{value}[n]\leftarrow f_n(\boldsymbol{x})$
  elif n **isa** product unit **then**
    $\texttt{value}[n]\leftarrow\prod_{c\in\mathsf{in}(n)}\texttt{value}[c]$
  else //n is a sum unit
    $\texttt{value}[n]\leftarrow\sum_{c\in\mathsf{in}(n)}\texttt{value}[c]$

Algorithm 2 Backward pass

Input: A deterministic PC p; $\forall n, \texttt{value}[n]$
Output: $\texttt{flow}[n,c] := (\boldsymbol{x}\in(\gamma_n\cap\gamma_c))$ for each pair (n,c), where n is a sum unit and $c\in\mathsf{in}(n)$
$\forall n, \texttt{context}[n]\leftarrow 0$; $\texttt{context}[n_r]\leftarrow\texttt{value}[n_r]$
foreach sum unit n traversed in preorder **do**
  foreach $m\in\mathsf{pa}(n)$ **do** (denote $g\leftarrow\mathsf{pa}(m)$)
    $\texttt{f}\leftarrow\frac{\texttt{value}[m]}{\texttt{value}[g]}\cdot\texttt{context}[g]$
    $\texttt{context}[n]$ += $\texttt{f}$; $\texttt{flow}[g,m]=\texttt{f}$

smoothing provides good priors to BNs and HBMs, its uniform prior could add unwanted bias to PCs. Specifically, for every sum unit n, Laplace smoothing assigns the same prior to all its child parameters (i.e., $\{\theta_{n,c}\mid c\in\mathsf{in}(n)\}$), while in many practical PCs, these parameters should be given drastically different priors. For example, consider the PC shown in Fig. 1(a). Since c_2 has an exponentially larger support than c_1, it should be assumed as prior that θ_{12} will be much larger than θ_{11}.

We highlight the significance of the above issue by examining the fraction of sum units with imbalanced child support sizes in PCs learned by Strudel, a state-of-the-art structure learning algorithm for deterministic PCs [5]. We examine 20 PCs learned from the 20 density estimation benchmarks [39], respectively. All sum units with ≥ 3 children and with a support size ≥ 128 are recorded. We measure "imbalanceness" of a sum unit n by the fraction of the maximum and minimum support size of its children (i.e., $\frac{\max_{c_1\in\mathsf{in}(n)}|\mathsf{supp}(c_1)|}{\min_{c_2\in\mathsf{in}(n)}|\mathsf{supp}(c_2)|}$). As demonstrated in Fig. 1(b), more than 20% of the sum units have imbalanceness $\geq 10^2$, which suggests that the inability of Laplace smoothing to properly regularize PCs with imbalanced sum units could lead to severe performance degradation in practice.

4 How Is This Tractable And Practical?

In this section, we first provide additional background about the parameter learning algorithms for deterministic and non-deterministic PCs (Sec. 4.1). We then demonstrate how the two intuitive ideas for regularizing distributions (Sec. 2), i.e., data softening and entropy regularization, can be efficiently implemented for deterministic (Sec. 4.2) and non-deterministic (Sec. 4.3) PCs.

4.1 Learning the Parameters of PCs

Deterministic PCs Given a deterministic PC p defined on variables $\mathbf{X}$ and a dataset $\mathcal{D}=\{\boldsymbol{x}^{(i)}\}_{i=1}^N$, the maximum likelihood estimation (MLE) parameters $\boldsymbol{\theta}^*_{\mathcal{D}}:=\operatorname{argmax}_{\boldsymbol{\theta}}\sum_{i=1}^N\log p(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$ can be learned in closed-form. To formalize the MLE solution, we need a few extra definitions.

Definition 5 (Context)**.** The context γ_n of every unit n in a PC p is defined in a top-down manner: for the base case, context of the root node n_r is defined as its support: $\gamma_{n_r}:=\mathsf{supp}(n_r)$. For every other node n, its context is the intersection of its support and the union of its parents' ($\mathsf{pa}(n)$) contexts:

$$\gamma_n := \bigcup_{m\in\mathsf{pa}(n)}\gamma_m\cap\mathsf{supp}(n).$$

Intuitively, if an assignment $\boldsymbol{x}$ is in the context of unit n, then there exists a path on the PC's DAG from n to the root unit n_r such that for any unit m in the path, we have $\boldsymbol{x}\in\mathsf{supp}(m)$. Circuit flow extends the notation of context to indicate whether a sample $\boldsymbol{x}$ is in the context of an edge (n,c).

Definition 6 (Flows)**.** The flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ of any edge (n,c) in a PC given variable assignments $\boldsymbol{x}\in\mathsf{val}(\mathbf{X})$ is defined as $\mathbb{1}[\boldsymbol{x}\in\gamma_n\cap\gamma_c]$, where $\mathbb{1}[\cdot]$ is the indicator function. The flow $\mathrm{F}_{n,c}(\mathcal{D})$ w.r.t. dataset $\mathcal{D}=\{\boldsymbol{x}^{(i)}\}_{i=1}^N$ is the sum of the flows of all its samples: $\mathrm{F}_{n,c}(\mathcal{D}):=\sum_{i=1}^N\mathrm{F}_{n,c}(\boldsymbol{x}^{(i)})$.

The flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ for *all* edges (n,c) in a PC p w.r.t. sample $\boldsymbol{x}$ can be computed through a forward and backward path that both take $\mathcal{O}(|p|)$ time. The forward path, as shown in Alg. 1, starts from the leaf units and traverses the PC in postorder to compute $\forall n, \texttt{value}[n]:=\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)]$; afterwards, the backward path illustrated in Alg. 2 begins at the root unit n_r and traverses the PC in preorder to

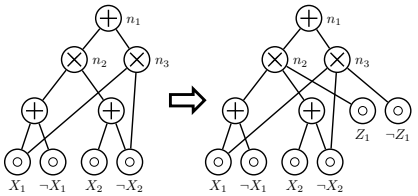


Figure 2: A non-deterministic PC can be modified as an equivalent deterministic PC with hidden variables.

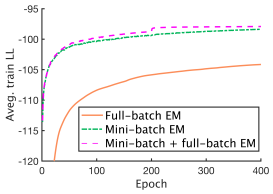


Figure 3: Average train LL on MNIST using different EM updates.

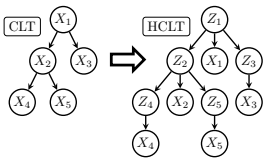


Figure 4: HCLT is constructed by adding hidden variables in a CLT [43].

compute $\forall n, \texttt{context}[n] := \mathbb{1}[\boldsymbol{x} \in \gamma_n]$ as well as $\forall (n,c), \texttt{flow}[n,c] := \mathrm{F}_{n,c}(\boldsymbol{x})$. By Def. 6, the time complexity for computing $\mathrm{F}_{n,c}(\mathcal{D})$ with respect to all edges (n,c) in p is $\mathcal{O}(|p| \cdot |\mathcal{D}|)$, where $|\mathcal{D}|$ is the size of dataset $\mathcal{D}$. The correctness of Alg. 1 and 2 are justified in Appendix A.6.

The MLE parameters $\boldsymbol{\theta}^*_{\mathcal{D}}$ given dataset $\mathcal{D}$ can be computed using the flows [5]:

$$\forall (n,c), \quad \theta^*_{n,c} = \mathrm{F}_{n,c}(\mathcal{D}) / \sum\nolimits_{c \in \mathsf{in}(n)} \mathrm{F}_{n,c}(\mathcal{D}). \tag{3}$$

Define hyperparameter α ($\alpha \geq 0$), for every sum unit n, Laplace smoothing regularizes its child parameters (i.e., $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$) by adding a *pseudocount* $\alpha / |\mathsf{in}(n)|$ to every child branch of n, which is equivalent to adding $\alpha / |\mathsf{in}(n)|$ to the numerator of Eq. (3) and α to its denominator.

Non-deterministic PCs As justified by Peharz et al. [40], every non-deterministic PC can be augmented as a deterministic PC with additional hidden variables. For example, in Fig. 2, the left PC is not deterministic since the support of both children of n_1 (i.e., n_2 and n_3) contains $x_1 \bar{x}_2$. The right PC augments the left one by adding input units correspond to hidden variable Z_1, which retains determinism by "dividing" the overlapping support $x_1 \bar{x}_2$ into $x_1 \bar{x}_2 z_1 \in \mathsf{supp}(n_2)$ and $x_1 \bar{x}_2 \bar{z}_1 \in \mathsf{supp}(n_3)$. Under this interpretation, parameter learning of non-deterministic PCs is equivalent to learning the parameters of deterministic PCs given incomplete data (we never observe the hidden variables), which can be solved by Expectation-Maximization (EM) [41, 42]. In fact, EM is the default parameter learning algorithm for non-deterministic PCs [13, 10].

Under the latent variable model view of a non-deterministic PC, its EM updates can be computed using *expected flows* [10]. Specifically, given observed variables $\mathbf{X}$ and (implicit) hidden variables $\mathbf{Z}$, the expected flow of edge (n,c) given dataset $\mathcal{D}$ is defined as

$$\mathrm{EF}_{n,c}(\mathcal{D}; \boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}, \boldsymbol{z} \sim p_c(\cdot \mid \boldsymbol{x}; \boldsymbol{\theta})} [\mathrm{F}_{n,c}(\boldsymbol{x}, \boldsymbol{z})],$$

where $\boldsymbol{\theta}$ is the set of parameters, and $p_c(\cdot \mid \boldsymbol{x}; \boldsymbol{\theta})$ is the conditional probability over hidden variables $\mathbf{Z}$ given $\boldsymbol{x}$ specified by the PC rooted at unit c. Similar to flows, the expected flows can be computed via a forward and backward pass of the PC (Alg. 5 and 6 in the Appendix). As shown by Choi et al. [10], for a non-deterministic PC, its parameters for the next EM iteration are given by

$$\theta^{(new)}_{n,c} = \mathrm{EF}_{n,c}(\mathcal{D}; \boldsymbol{\theta}) / \sum_{c \in \mathsf{in}(n)} \mathrm{EF}_{n,c}(\mathcal{D}; \boldsymbol{\theta}). \tag{4}$$

This paper uses a hybrid EM algorithm, which uses mini-batch EM updates to initiate the training process, and switch to full-batch EM updates afterwards. Specifically, in mini-batch EM, $\boldsymbol{\theta}^{(new)}$ are computed using mini-batches of samples, and the parameters are updated towards the taget with a step size η: $\boldsymbol{\theta}^{(k+1)} \leftarrow (1-\eta)\boldsymbol{\theta}^{(k)} + \eta \boldsymbol{\theta}^{(new)}$; when using full-batch EM, we iteratively compute the updated parameters $\boldsymbol{\theta}^{(new)}$ using the whole dataset. Fig. 3 demonstrates that this hybrid approach offers faster convergence speed compared to using full-batch or mini-batch EM only.

4.2 Regularizing Deterministic PCs

We demonstrate how the intuitive ideas for regularizing distributions presented in Sec. 2 (i.e., data softening and entropy regularization) can be efficiently applied to deterministic PCs.

Data softening As hinted by Eq. (1), we need exponentially many samples to represent a softened dataset, which makes parameter learning intractable even for the simple logistic regression model (Thm. 1), let alone more complex probabilistic models such as VAEs [44] and GANs [45]. Despite

Algorithm 3 PC Entropy regularization

1: **Input:** A deterministic PC p; flow $\mathrm{F}_{n,c}(\mathcal{D})$ for every edge (n, c) in p; hyperparameter τ.
2: **Output:** A set of log-parameters, $\{\varphi_{n,c} : (n, c) \in p\}$, which are the solution of Eq. (2).
3: $\forall n,\ \texttt{node\_prob}[n] \leftarrow 0$; $\quad \texttt{node\_prob}[n_r] \leftarrow 1$ //n_r is the root node of p
4: **while** not converge **do**
5: $\quad \forall n,\ \texttt{entropy}[n] \leftarrow$ The entropy of the sub-PC rooted at n (see Alg. 4 in Appendix A.2)
6: $\quad$ **foreach** sum unit n traversed in preorder (parent before children) **do**
7: $\quad\quad d_i \leftarrow \mathrm{F}_{n,c_i}(\mathcal{D})/|\mathcal{D}|$; $\quad b = \tau \cdot \texttt{node\_prob}[n]$ //$\{c_i\}_{i=1}^{\mathsf{in}(n)}$ is the set of children of n
8: $\quad\quad$ Solve for $\{\varphi_{n,c_i}\}_{i=1}^{|\mathsf{in}(n)|}$ in the following set of equations (y is a variable):

$$\begin{cases} d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \texttt{entropy}[c_i] = y \quad (\forall i \in \{1, \dots, |\mathsf{in}(n)|\}) \\ \sum_{i=1}^{|\mathsf{in}(n)|} e^{\varphi_{n,c_i}} = 1 \end{cases} \tag{5}$$

9: $\quad\quad$ **for** each $c \in \mathsf{in}(n)$ and each $m \in \mathsf{in}(c)$ **do** //Update node_prob of grandchildren
10: $\quad\quad\quad \texttt{node\_prob}[m] \leftarrow \texttt{node\_prob}[m] + e^{\varphi_{n,c}} \cdot \texttt{node\_prob}[n]$

this negative result, the MLE parameters of a PC p w.r.t. $\mathcal{D}_\beta$ can be computed in time $\mathcal{O}(|p| \cdot |\mathcal{D}|)$, which is linear w.r.t. the model size as well as the size of the *original* dataset.

Theorem 3. *Let $f_n(\boldsymbol{x}) = \beta \cdot \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(n)] + (1-\beta) \cdot \mathbb{1}[\boldsymbol{x} \notin \mathsf{supp}(n)]$ in Alg. 1. Given a deterministic PC p, a boolean dataset $\mathcal{D}$, and hyperparameter $\beta \in (0.5, 1]$, the set of all flows $\{\mathrm{F}_{n,c}(\mathcal{D}_\beta) \mid \forall$ edge $(n, c)\}$ w.r.t. the softened dataset $\mathcal{D}_\beta$ can be computed by Alg. 1 and 2 within $\mathcal{O}(|p| \cdot |\mathcal{D}|)$ time.*

Proof of this theorem is provided in Appendix A.1. Since the MLE parameters (Eq. (3)) w.r.t. $\mathcal{D}_\beta$ can be computed in $\mathcal{O}(|p|)$ time using the flows, the overall time complexity to compute the MLE parameters is again $\mathcal{O}(|p| \cdot |\mathcal{D}|)$.

Entropy regularization The hope for tractable PC entropy regularization comes from the fact that the entropy of a deterministic PC p can be exactly computed in $\mathcal{O}(|p|)$ time [6, 46]. However, it is still unclear whether the entropy regularization objective $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$ (Eq. (2)) can be tractably maximized. We answer this question with a mixture of positive and negative results: while the objective is multi-modal and the global optimal is hard to find, we propose an efficient algorithm that (i) guarantees convergence to a stationary point, and (ii) achieves high convergence rate in practice. We start with the negative result.

Proposition 1. *There exists a deterministic PC p, a hyperparameter τ, and a dataset $\mathcal{D}$ such that $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$ (Eq. (2)) is non-concave and has multiple local maximas.*

Proof is given in Appendix A.7. Although global optimal solutions are generally infeasible, we propose an efficient algorithm that guarantees to find a stationary point of $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$. Specifically, Alg. 3 takes as input a deterministic PC p and all its edge flows w.r.t. $\mathcal{D}$, and returns a set of learned log-parameters that correspond to a stationary point of the objective.[2] In its main loop (lines 4-10), the algorithm alternates between two procedures: (i) compute the entropy of the distribution encoded by every node w.r.t. the current parameters (line 5),[3] and (ii) update PC parameters with regard to the computed entropies (lines 6-10). Specifically, in the parameter update phase (i.e., the second phase), the algorithm traverses every sum unit n in preorder and updates its child parameters by maximizing the entropy regularization objective ($\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$) with all other parameters fixed. This is done by solving the set of equations in Eq. (5) using Newton's method (lines 7-8).[4] In addition to the child nodes' entropy computed in the first phase, Eq. (5) uses the top-down probability of every unit n (i.e., $\texttt{node\_prob}[n]$), which is progressively updated in lines 9-10.

Theorem 4. *Alg. 3 converges monotonically to a stationary point of $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$ (Eq. (2)).*

Proof. The high-level idea of the proof is to show that the parameter update phase (lines 6-10) optimizes a concave surrogate objective of $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$, which is determined by the entropies computed

[2] We compute parameters in the logarithm space for numerical stability.

[3] This can be done by Alg. 4 shown in Appendix A.2. Lem. 1 proves that Alg. 4 takes $\mathcal{O}(|p|)$ time.

[4] Details for solving Eq. (5) is given in Appendix B.2.

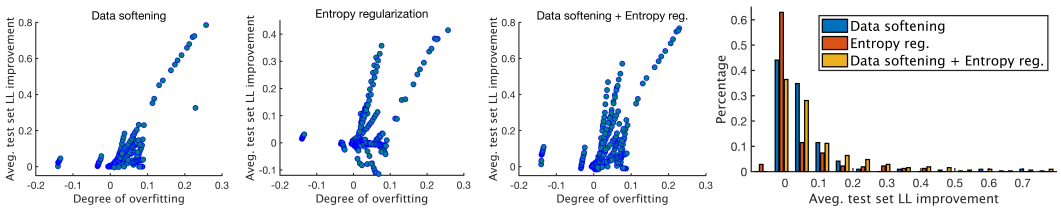


Figure 5: Both data softening and entropy regularization effectively improve the test set log-likelihood (LL) across various datasets [39] and PC structures [17]. LL improvement (higher is better) represents the gain of test set LL compared to Laplace smoothing. The test-set LLs are reported in Table 3.

in line 5. Specifically, we show that whenever the surrogate objective is improved, $\texttt{LL}_{\text{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$ is also improved. Since the surrogate objective is concave, it can be easily optimized. Therefore, Alg. 3 converges to a stationary point of $\texttt{LL}_{\text{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$. The detailed proof is in Appendix A.5. □

Alg. 3 can be regarded as a EM-like algorithm, where the E-step is the entropy computation phase (line 5) and the M-step is the parameter update phase (lines 6-10). Specifically, the E-step constructs a concave surrogate of the true objective ($\texttt{LL}_{\text{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$), and the M-step updates all parameters by maximizing the concave surrogate function. Although Thm. 4 provides no convergence rate analysis, the outer loop typically takes 3-5 iterations to converge in practice. Furthermore, Eq. (5) can be solved with high precision in a few (< 10) iterations. Therefore, compared to the computation of all flows w.r.t. $\mathcal{D}$, which takes $\mathcal{O}(|p| \cdot |\mathcal{D}|)$ time, Alg. 3 takes a negligible $\mathcal{O}(|p|)$ time.

In response to the motivation in Sec. 3, we show that both proposed methods can overcome the imbalanced regularization problem of Laplace smoothing. Again consider the example PC in Fig. 1(a), we conceptually demonstrate that both data softening and entropy regularization will not over-regularization θ_{11} compared to θ_{12}. First, data softening essentially add no prior to the parameters, and only soften the evidences in the dataset. Therefore, it will not over-regularize children with small support sizes. Second, entropy regularization will add a much higher prior to θ_{12}. Suppose $n = 10$, consider maximizing Eq. (2) with an empty dataset (i.e., we maximize $\texttt{ENT}(p_{n_1})$ directly), the optimal parameters would be $\theta_{11} \approx 0.002$ and $\theta_{12} \approx 0.998$. Therefore, entropy regularization will tend to add a higher prior to children with large support sizes. More fundamentally, the reason why both proposed approaches do not add biased priors to PCs is that they are designed to be model-agnostic, i.e., their definitions as shown in Sec. 2 are independent with the model they apply to.

Empirical evaluation We empirically evaluate both proposed regularization methods on the twenty density estimation datasets [39]. Since we are only concerned with parameter learning, we adopt PC structures (defined by its DAG) learned by Strudel [17]. 16 PCs with different sizes were selected for each of the 20 datasets. For all experiments, we performed a hyperparameter search for all three regularization approaches (Laplace smoothing, data softening, and entropy regularization)[5] using the validation set and report results on the test set. Please refer to Appendix B.3 for more details.

Results are summarized in Fig. 5. First look at the scatter plots on the left. The x-axis represents the degree of overfitting, which is computed as follows: denote $\texttt{LL}_{train}$ and $\texttt{LL}_{val}$ as the average train and validation log-likelihood under the MLE estimation with Laplace smoothing ($\alpha = 1.0$), the degree of overfitting is defined as $(\texttt{LL}_{val} - \texttt{LL}_{train})/\texttt{LL}_{val}$, which roughly captures how much the dataset/model pair suffers from overfitting. The y-axis represents the improvement on the average test set log-likelihood compared to Laplace smoothing. As demonstrated by the scatter plots, despite a few outliers, both proposed regularization methods steadily improve the test set LL over various datasets and PC structures, and the LL improvements are positively correlated with the degree of overfitting. Furthermore, as shown by the last scatter plot and the histogram plot, when combining data softening and entropy regularization, the LL improvement becomes much higher compared to using the two regularizers individually.

4.3 Regularizing Non-Deterministic PCs

By viewing every non-deterministic PC as a deterministic PC with additional hidden variables (Sec. 4.1), the regularization techniques developed in Sec. 4.2 can be directly adapted. Specifically,

[5] Specifically, $\alpha \in \{0.1, 0.4, 1.0, 2.0, 4.0, 10.0\}$, $\beta \in \{0.9996, 0.999, 0.996\}$, $\tau \in \{0.001, 0.01, 0.1\}$.

Table 1: Test set log-likelihood in 20 density estimation benchmarks. We compare our method (HCLT) with the best performance (Best PSDD) over 2 deterministic PC learner: Strudel [17] and LearnPSDD [16] as well as the best performance (Best SPN) over 4 SPN learning algorithms: EinSumNet [13], LearnSPN [18], ID-SPN [47], and RAT-SPN [48]. With the help of data softening and entropy regularization ($\alpha = 0.1$, $\beta = 0.998$, and $\tau = 0.001$), HCLT achieved the best performance over 10 out of 20 datasets. All experiments for HCLT were repeated 5 times, and the average and standard deviation are reported.

Dataset	HCLT	Best PSDD	Best SPN	Dataset	HCLT	Best PSDD	Best SPN
accidents	**-26.74**±0.03	-28.29	-26.98	jester	**-52.46**±0.01	-54.63	-52.56
ad	**-16.07**±0.06	-16.52	-19.00	kdd	-2.18±0.00	-2.17	**-2.12**
baudio	**-39.77**±0.01	-41.51	-39.79	kosarek	-10.66±0.01	-10.98	**-10.60**
bbc	-251.04±1.19	-258.96	**-248.33**	msnbc	-6.05±0.01	-6.04	**-6.03**
bnetflix	**-56.27**±0.01	-58.53	-56.36	msweb	-9.98±0.05	-9.93	**-9.73**
book	**-33.83**±0.01	-35.77	-34.14	nltcs	**-5.99**±0.01	-6.03	-6.01
c20ng	-153.40±3.83	-160.43	**-151.47**	plants	-14.26±0.16	-13.49	**-12.54**
cr52	-86.26±3.67	-92.38	**-83.35**	pumbs*	-23.64±0.25	-25.28	**-22.40**
cwebkb	-152.77±1.07	-160.5	**-151.84**	tmovie	**-50.81**±0.12	-55.41	-51.51
dna	**-79.05**±0.17	-82.03	-81.21	tretail	**-10.84**±0.01	-10.90	-10.85

data softening can be regarded as injecting noise in both observed and hidden variables. Since the dataset provides no information about the hidden variables anyway, data softening essentially still "perturbs" the observed variables only. On the other hand, entropy regularization will have different behaviors when applied to non-deterministic PCs. Specifically, since it is coNP-hard to compute the entropy of a non-deterministic PC [6], it is infeasible to optimize the entropy regularization objective $\mathrm{LL}_{\mathrm{ent}}(\boldsymbol{\theta}; \mathcal{D}, \tau)$ (Eq. (2)). However, we can still regularize the entropy of the distribution encoded by a non-deterministic PC over both of its observed and hidden variables, since explicitly representing the hidden variables renders the PC deterministic (Sec. 4.1).

On the implementation side, data softening is performed by modifying the forward pass of the algorithm used to compute expected flows (i.e., Alg. 5 and 6 in the Appendix). Entropy regularization is again performed by Alg. 3 at the M-step of each min-batch/full-batch EM update, except that the input flows (i.e., F) are replaced by the corresponding expected flows (i.e., EF).

Empirical evaluation We use a simple yet effective PC structure, hidden Chow-Liu Tree (HCLT), as demonstrated in Fig. 4. Specifically, on the left is a Bayesian network representation of a Chow-Liu Tree (CLT) [43] over 5 variables. For any CLT over variables $\{X_i\}_{i=1}^k$, we can modify it as a HCLT through the following steps. First, we introduce a set of k latent variables $\{Z_i\}_{i=1}^k$. Next, we replace all observed variables in the CLT with its corresponding latent variable (i.e., $\forall i, X_i$ is replaced by Z_i). Finally, we add an edge from every latent variable to its corresponding observed variable (i.e., $\forall i$, add an edge $Z_i \to X_i$). The HCLT structure is then compiled into a PC that encodes the same probability distribution. We used the hybrid mini-batch + full-batch EM as described in Sec. 4.1. For all experiments, we trained the PCs with 100 mini-batch EM epochs and 100 full-batch EM epochs. Please refer to Appendix B.4 for hyperparameters related to the HCLT structure and the parameter learning process. Similar to Sec. 4.2, we perform hyperparameter search for all methods using the validation set, and report results on the test set.

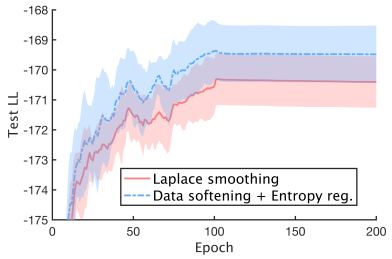


Figure 6: Average (±std) test LL over 5 trials on the protein dataset.

We first examine the performance on a protein sequence dataset [29] that suffers from severe overfitting. Specifically, the training LL is typically above -100 while the validation and test set LL are around -170. Fig. 6 shows the test LL for Laplace smoothing and the hybrid regularization approach as training progresses. With the help of data softening and entropy regularization, we were able to obtain consistently higher test set LL. Next, we compare our HCLT model (with regularization) with the state-of-the-art PSDD (Strudel [17] and LearnPSDD [16]) and SPN (EinSumNet [13], LearnSPN [18], ID-SPN [47], and RAT-SPN [48]) learning algorithms. Results are summarized in Table 1. With proper regularization, HCLT out-performed all baselines in 10 out of 20 datasets. Comparing with individual baselines, HCLT out-performs both PSDD learners on all datasets; HCLT

achieved higher log-likelihood on 18, 19, 10, and 17 datasets compared to EinSumNet, LearnSPN, ID-SPN, and RAT-SPN, respectively.

5 Conclusions

This paper proposes two model-agnostic distribution regularization techniques: data softening and entropy regularization. While both methods are infeasible for many machine learning models, we theoretically show that they can be efficiently implemented when applied to probabilistic circuits. On the empirical side, we show that both proposed regularizers consistently improve the generalization performance over a wide variety of PC structures and datasets.

Acknowledgement This work is partially supported by NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, DARPA grant #N66001-17-2-4032, a Sloan Fellowship, Intel, and Facebook.

References

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

[2] Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021.

[3] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

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

[5] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th international conference on principles of knowledge representation and reasoning (KR)*, pages 1–10, 2014.

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

[7] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, dec 2019.

[8] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3943–3951. Citeseer, 2016.

[9] Yitao Liang and Guy Van den Broeck. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, August 2017.

[10] YooJung Choi, Meihua Dang, and Guy Van den Broeck. Group fairness by probabilistic modeling with latent fair decisions. *arXiv preprint arXiv:2009.09031*, 2020.

[11] YooJung Choi, Golnoosh Farnadi, Behrouz Babaki, and Guy Van den Broeck. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10077–10084, 2020.

[12] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. *arXiv preprint arXiv:1901.03704*, 2019.

[13] 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 *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[16] Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[17] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. *arXiv preprint arXiv:2007.09331*, 2020.

[18] Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880. PMLR, 2013.

[19] Hei Chan and Adnan Darwiche. On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, 163(1):67–90, 2005.

[20] Rong Pan, Yun Peng, and Zhongli Ding. Belief update in bayesian networks using uncertain evidence. In *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 441–444. IEEE, 2006.

[21] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. In *Case Studies in Applied Bayesian Data Science*, pages 45–87. Springer, 2020.

[22] Nicola Di Mauro, Antonio Vergari, and Floriana Esposito. Learning accurate cutset networks by exploiting decomposability. In *Congress of the Italian Association for Artificial Intelligence*, pages 221–232. Springer, 2015.

[23] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[24] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[26] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[27] Richard C Jeffrey. *The logic of decision*. University of Chicago press, 1990.

[28] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[29] William P Russ, Matteo Figliuzzi, Christian Stocker, Pierre Barrat-Charlaix, Michael Socolich, Peter Kast, Donald Hilvert, Remi Monasson, Simona Cocco, Martin Weigt, et al. An evolution-based model for designing chorismate mutase enzymes. *Science*, 369(6502):440–445, 2020.

[30] Yves Grandvalet and Yoshua Bengio. Entropy regularization., 2006.

[31] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[32] Yihao Feng, Dilin Wang, and Qiang Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

[33] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

[34] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, volume 32, 2014.

[35] Jun Mei, Yong Jiang, and Kewei Tu. Maximum a posteriori inference in sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[36] Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In *International Conference on Machine Learning*, pages 825–833. PMLR, 2017.

[37] David Heckerman. A tutorial on learning with bayesian networks. *Innovations in Bayesian networks*, pages 33–82, 2008.

[38] Greg M Allenby and Peter E Rossi. Hierarchical bayes models. *The handbook of marketing research: Uses, misuses, and future advances*, pages 418–440, 2006.

[39] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[40] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016.

[41] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[42] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[43] CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[44] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[45] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[46] Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, december 2020.

[47] Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *International Conference on Machine Learning*, pages 710–718. PMLR, 2014.

[48] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020.

[49] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of SHAP explanations. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, Feb 2021.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section X.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Contributions are clearly stated in lines 42-52 and match the referred theorems and algorithms.
   (b) Did you describe the limitations of your work? [Yes] Sec. 4.2 discusses the limitation of Thm. 4; Sec. 4.3 discusses the limitation of applying entropy regularization to non-deterministic PCs.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] All theorems in the main text formally state all assumptions.
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs are included in the appendix. We added a reference to the corresponding proof after each theorem statement.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and instructions to reproduce the experimental results are included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All details for reproducibility are specified in Appendices B.3 and B.4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Error bars and standard deviations over 5 runs are reported in Fig. 6 and Table 1, respectively.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Details about computing resources can be found in Appendix B.3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited all PC learning algorithms as well as the datasets/benchmarks we adopted in Sec. 4.
   (b) Did you mention the license of the assets? [Yes] We specified both the used algorithm and data are publicly available in Sec. 4.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We included our code in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material

A Proofs

This section provides the full proof of the theorems stated in the main paper.

A.1 Proof of Theorem 3

High-level idea The high-level idea of this proof is by separately showing the correctness of the forward pass (Alg. 1) and the backward pass (Alg. 2). Specifically, for a "softened" sample $\boldsymbol{x}$, we aim to show that (i) in the forward pass, the value of $\boldsymbol{x}$ w.r.t. any PC unit n corresponds to the likelihood of $\boldsymbol{x}$ (note that since $\boldsymbol{x}$ can be represented as a weighted sum of exponentially many "hard" samples, the target likelihood is also the weighted sum of the respective likelihoods), and (ii) in the backward pass, the flow of $\boldsymbol{x}$ w.r.t. any PC unit corresponds to the weighted sum of the flows of the "hard" samples "contained" in $\boldsymbol{x}$. Both claims are proved by induction: for the forward pass, we first show that the base cases (leaf nodes) satisfy the claim, then by assuming all children of a PC unit satisfy the claim, we prove the inductive case of sum and product units; for the backward pass, induction is also applied in the preorder (parents before children).

As stated in the theorem, assume that we are given a deterministic PC p, a boolean dataset $\mathcal{D}$ containing N samples $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$, and hyperparameter $\beta \in (0.5, 1]$. Define K as the number of variables in $\mathbf{X}$, i.e., $\mathbf{X} = \{X_k\}_{k=1}^{K}$.

Correctness of the forward pass We show that the value of each node n w.r.t. sample $\boldsymbol{x}^{(i)}$ (by slightly abusing notation, denoted as $\texttt{value}_i[n]$) computed by Alg. 1 (with the specific choice of $f_n(\boldsymbol{x}) = \beta \cdot \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(n)] + (1-\beta) \cdot \mathbb{1}[\boldsymbol{x} \notin \mathsf{supp}(n)]$) is defined as

$$\texttt{value}_i[n] = \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta \cdot \mathbb{1}[x_k^{(i)} = x_k] + (1-\beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(n)], \tag{6}$$

where x_k denotes the kth feature of $\boldsymbol{x}$.

• Base case: input units. Suppose node n is a literal w.r.t. variable X_k. That is, $\boldsymbol{x} \in \mathsf{supp}(n)$ iff $x_k = \texttt{Lit}(n)$, where $\texttt{Lit}(n)$ is either $\texttt{true}$ or $\texttt{false}$ defined by the PC. Denote $\neg\texttt{Lit}(n)$ as the negation of $\texttt{Lit}(n)$. $\forall i \in \{1, \ldots, N\}$ we have

$$\begin{aligned}
\texttt{value}_i[n] =& \beta \cdot \mathbb{1}[\boldsymbol{x}^{(i)} \in \mathsf{supp}(n)] + (1-\beta) \cdot \mathbb{1}[\boldsymbol{x}^{(i)} \notin \mathsf{supp}(n)] \\
=& \beta \cdot \mathbb{1}[x_k^{(i)} = \texttt{Lit}(n)] + (1-\beta) \cdot \mathbb{1}[x_k^{(i)} = \neg\texttt{Lit}(n)] \\
\overset{(a)}{=}& \sum_{\boldsymbol{x} \in \{\boldsymbol{x} : \boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \wedge x_k = \texttt{Lit}(n)\}} \prod_{l=1, l\neq k}^{K} \Big(\beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1-\beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l]\Big) \\
& \cdot \Big(\beta \cdot \mathbb{1}[x_k^{(i)} = \texttt{Lit}(n)] + (1-\beta) \cdot \mathbb{1}[x_k^{(i)} = \neg\texttt{Lit}(n)]\Big) \\
=& \sum_{\boldsymbol{x} \in \{\boldsymbol{x} : \boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \wedge x_k = \texttt{Lit}(n)\}} \prod_{l=1, l\neq k}^{K} \Big(\beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1-\beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l]\Big) \\
& \cdot \Big(\beta \cdot \mathbb{1}[x_k^{(i)} = x_k] \cdot \mathbb{1}[x_k = \texttt{Lit}(n)] + (1-\beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k] \cdot \mathbb{1}[x_k = \texttt{Lit}(n)]\Big) \\
=& \sum_{\boldsymbol{x} \in \{\boldsymbol{x} : \boldsymbol{x} \in \mathsf{val}(\mathbf{X}) \wedge x_k = \texttt{Lit}(n)\}} \prod_{l=1, l\neq k}^{K} \Big(\beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1-\beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l]\Big) \\
& \cdot \Big(\beta \cdot \mathbb{1}[x_k^{(i)} = x_k] + (1-\beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[x_k = \texttt{Lit}(n)] \\
\overset{(b)}{=}& \sum_{\boldsymbol{x} \in \{\boldsymbol{x} : \boldsymbol{x} \in \mathsf{val}(\mathbf{X})\}} \prod_{l=1}^{K} \Big(\beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1-\beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l]\Big) \cdot \mathbb{1}[x_k = \texttt{Lit}(n)]
\end{aligned}$$

$$= \sum_{\boldsymbol{x}\in\{\boldsymbol{x}:\boldsymbol{x}\in\mathsf{val}(\mathbf{X})\}} \prod_{l=1}^{K} \Big(\beta\cdot\mathbb{1}[x_l^{(i)} = x_l] + (1-\beta)\cdot\mathbb{1}[x_l^{(i)} \neq x_l]\Big)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)],$$

where (a) holds because the added term

$$\sum_{\boldsymbol{x}\in\{\boldsymbol{x}:\boldsymbol{x}\in\mathsf{val}(\mathbf{X})\wedge x_k=\texttt{Lit}(n)\}} \prod_{l=1,l\neq k}^{K} \Big(\beta\cdot\mathbb{1}[x_l^{(i)} = x_l] + (1-\beta)\cdot\mathbb{1}[x_l^{(i)} \neq x_l]\Big) = 1;$$

the sum condition $\boldsymbol{x}_k = \texttt{Lit}(n)$ after (b) can be lifted thanks to the indicator $\mathbb{1}[x_k = \texttt{Lit}(n)]$.

• Inductive case: product units. Suppose n is a product unit with children $\{c_j\}_{j=1}^{|\mathsf{in}(n)|}$. Recall that the scope of the child c_j is denoted as $\phi(c_j)$. Since the PC is decomposable, the contexts of different children are non-overlapping. Suppose the value of any child unit c_j is defined according to Eq. (6), i.e.,

$$\texttt{value}_i[c_j] = \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_j)].$$

Denote K_{c_j} as the set of index for the variables in $\phi(c_j)$. We have

$$\begin{aligned}
\texttt{value}_i[n] &\overset{(a)}{=} \prod_{j=1}^{|\mathsf{in}(n)|} \texttt{value}_i[c_j] \\
&= \prod_{j=1}^{|\mathsf{in}(n)|} \Big\{ \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_j)]\Big\} \\
&= \prod_{j=1}^{|\mathsf{in}(n)|} \Big\{ \sum_{\boldsymbol{x}\in\mathsf{val}(\phi(c_j))} \prod_{k\in K_{c_j}} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_j)]\Big\} \\
&\overset{(b)}{=} \sum_{\boldsymbol{x}\in\mathsf{val}(\bigcup_{j=1}^{|\mathsf{in}(n)|}\phi(c_j))} \prod_{k\in\bigcup_{j=1}^{|\mathsf{in}(n)|}K_{c_j}} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \\
&\qquad\qquad \cdot \Big(\prod_{l=1}^{|\mathsf{in}(n)|} \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_l)]\Big) \\
&\overset{(c)}{=} \sum_{\boldsymbol{x}\in\mathsf{val}(\bigcup_{j=1}^{|\mathsf{in}(n)|}\phi(c_j))} \prod_{k\in\bigcup_{j=1}^{|\mathsf{in}(n)|}K_{c_j}} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)] \\
&\overset{(d)}{=} \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)],
\end{aligned}$$

where (a) holds by line 6 of Alg. 1; (b) holds since $\forall c_i, c_j \in \mathsf{in}(n)(c_i \neq c_j)$, we have $\phi(c_i)\cap\phi(c_j) = \varnothing$ and $K_{c_i}\cap K_{c_j} = \varnothing$ thanks to decomposability of the PC; (c) is satisfied by the definition of product units: $\mathsf{supp}(n) = \bigcap_{c\in\mathsf{in}(n)}\mathsf{supp}(c)$; (d) holds since $\bigcup_{j=1}^{|\mathsf{in}(n)|}\phi(c_j)$ is a subset of $\mathbf{X}$.

• Inductive case: sum units. Suppose n is a sum unit with children $\{c_j\}_{j=1}^{|\mathsf{in}(n)|}$. Suppose the value $\texttt{value}_i[c_j]$ of any child unit c_j is defined according to Eq. (6), we have

$$\begin{aligned}
\texttt{value}_i[n] &\overset{(a)}{=} \sum_{j=1}^{|\mathsf{in}(n)|} \texttt{value}_i[c_j] \\
&= \sum_{j=1}^{|\mathsf{in}(n)|} \Big\{ \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_j)]\Big\}
\end{aligned}$$

$$\overset{(b)}{=} \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \Big(\sum_{j=1}^{|\mathsf{in}(n)|} \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(c_j)]\Big)$$

$$\overset{(c)}{=} \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)],$$

where (a) follows line 8 of Alg. 1; (b) holds because the sum unit n is deterministic: $\forall c_i, c_j \in \mathsf{in}(n)(c_i \neq c_j), \mathsf{supp}(c_i)\cap\mathsf{supp}(c_j) = \varnothing$; (c) follows from the definition of sum units: $\mathsf{supp}(n) = \bigcup_{c\in\mathsf{in}(n)}\mathsf{supp}(c)$.

We have shown that for any unit n, the value stored in $\mathtt{value}_i[n]$ follows the definition in Eq. (6). We proceed to show the correctness of the backward pass.

Correctness of the backward pass Similar to the forward pass, we show that the context $\mathtt{context}_i[n]$ of each sum unit w.r.t. sample $\boldsymbol{x}^{(i)}$ computed by Alg. 2 is defined as

$$\mathtt{context}_i[n] = \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n], \tag{7}$$

and the flow $\mathtt{flow}_i[n, c]$ of each edge (n, c) s.t. n is a sum unit is:

$$\mathtt{flow}_i[n, c] = \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n \wedge \boldsymbol{x}\in\gamma_c]. \tag{8}$$

• Base case: root unit n_r. Without loss of generality, we assume the root node represents a sum unit.[6] According to Def. 5, the context of the root node n_r equals its support, i.e., $\gamma_{n_r} = \mathsf{supp}(n_r)$. Since in line 3 of Alg. 2, the value $\mathtt{context}_i[n]$ is set to $\mathtt{value}_i[n]$, we know that

$$\begin{aligned}\mathtt{context}_i[n] &= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(n)] \\ &= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n].\end{aligned}$$

• Inductive case: sum unit. Suppose n is a sum unit with parent product units $\{m_j\}_{j=1}^{|\mathsf{pa}(n)|}$. Denote the parent of product unit m_i as g_i.[7] Suppose the contexts of $\{g_j\}_{j=1}^{|\mathsf{pa}(n)|}$ satisfy Eq. (7). For ease of presentation, denote $H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) := \big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\big)$.

$$\begin{aligned}\mathtt{flow}_i[g_j, m_j] =& \frac{\mathtt{value}_i[m_j]}{\mathtt{value}_i[g_j]} \cdot \mathtt{context}_i[g_j] \\ =& \frac{\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_{g_j}]}{\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]} \\ & \cdot \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(m_j)]\end{aligned} \tag{9}$$

Define $\gamma'_{g_j} := \bigcup_{c\in\mathsf{pa}(g_j)} \gamma_c$, Def. 5 suggests that $\gamma_{g_j} = \gamma'_{g_j} \cap \mathsf{supp}(g_j)$. Thus,

$$\mathbb{1}[\boldsymbol{x}\in\gamma_{g_j}] = \mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}] \cdot \mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]. \tag{10}$$

[6] Note that if the root unit is not a sum, we can always add a sum unit as its parent and set the corresponding edge parameter to 1.

[7] W.l.o.g. we assume all product unit only have one parent.

Consider conditioning $\mathsf{supp}(g_j)$ and γ'_{g_j} on the variables $\phi(g_j)$ (i.e., the variable scope of g_j). For any partial variable assignment $\boldsymbol{e}$ over $\phi(g_j)$, if $\boldsymbol{e} \in \mathsf{supp}(g_j)$, then $\boldsymbol{e} \in \gamma'_{g_j}$. Denote K_{g_j} as the set of index for the variables in $\phi(g_j)$. We have

$$\begin{aligned}
&\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})}\prod_{k=1}^{K} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}]\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]\\
=&\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\phi(g_j))}\prod_{k\in K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]\Bigg)\\
&\cdot\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X}\setminus\phi(g_j))}\prod_{k\in\{1,\dots,K\}\setminus K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}]\Bigg) \qquad (11)
\end{aligned}$$

Plug Eqs. (11) and (10) into Eq. (9), we have

$$\begin{aligned}
\texttt{flow}_i[g_j,m_j] =& \frac{\sum_{\boldsymbol{x}\in\mathsf{val}(\phi(g_j))}\prod_{k\in K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]}{\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})}\prod_{k=1}^{K} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(g_j)]}\\
&\cdot\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X}\setminus\phi(g_j))}\prod_{k\in\{1,\dots,K\}\setminus K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}]\Bigg)\\
&\cdot\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})}\prod_{k=1}^{K} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(m_j)]\Bigg)\\
=&\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X}\setminus\phi(g_j))}\prod_{k\in\{1,\dots,K\}\setminus K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}]\Bigg)\\
&\cdot\Bigg(\sum_{\boldsymbol{x}\in\mathsf{val}(\phi(g_j))}\prod_{k\in K_{g_j}} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\mathsf{supp}(m_j)]\Bigg) \qquad (12)
\end{aligned}$$

Since m_j is a child of g_j, the support of m_j is a subset of g_j's support: $\mathsf{supp}(m_j) \subseteq \mathsf{supp}(g_j)$. Therefore, for any partial variable assignment $\boldsymbol{e}$ over $\phi(g_j)$, if $\boldsymbol{e} \in \mathsf{supp}(m_j)$, then $\boldsymbol{e} \in \mathsf{supp}(g_j)$. Since $\{\boldsymbol{e} \mid \boldsymbol{e} \in \mathsf{val}(\phi(g_j)) \wedge \boldsymbol{e} \in \mathsf{supp}(g_j)\} \subseteq \{\boldsymbol{e} \mid \boldsymbol{e} \in \mathsf{val}(\phi(g_j)) \wedge \boldsymbol{e} \in \gamma'_{g_j}\}$, we conclude that for any partial variable assignment $\boldsymbol{e}$ over $\phi(g_j)$, if $\boldsymbol{e} \in \mathsf{supp}(m_j)$, then $\boldsymbol{e} \in \gamma'_{g_j}$. Therefore, the two product terms in Eq. (12) can be joined with a Cartesian product:

$$\texttt{flow}_i[g_j,m_j] = \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})}\prod_{k=1}^{K} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma'_{g_j}\cap\mathsf{supp}(m_j)]. \qquad (13)$$

Note that $\gamma'_{g_j}\cap\mathsf{supp}(m_j) = \gamma'_{g_j}\cup\mathsf{supp}(g_j)\cap\mathsf{supp}(m_j) = \gamma_{g_j}\cap\mathsf{supp}(m_j)$. Since $\gamma_{m_j} = \gamma_{g_j}\cap\mathsf{supp}(m_j)$ (according to Def. 5), we have

$$\begin{aligned}
\gamma'_{g_j}\cap\mathsf{supp}(m_j) &= \gamma_{g_j}\cap\mathsf{supp}(m_j)\\
&= \gamma_{g_j}\cap\mathsf{supp}(g_j)\cap\mathsf{supp}(m_j)\\
&= \gamma_{g_j}\cap\gamma_{m_j}.
\end{aligned}$$

Plug the above equation into Eq. (13), we have

$$\texttt{flow}_i[g_j,m_j] = \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})}\prod_{k=1}^{K} H(\boldsymbol{x},\boldsymbol{x}^{(i)},k)\cdot\mathbb{1}[\boldsymbol{x}\in\gamma_{g_j}\cap\gamma_{m_j}],$$

which is equivalent to Eq. (7).

We proceed to show that the context of unit n follows Eq. (8). According to lines 6 and 7 of Alg. 2, $\texttt{context}_i[n]$ is computed as

$$\texttt{context}_i[n] = \sum_{j=1}^{|\mathsf{pa}(n)|}\texttt{flow}_i[g_j,m_j]$$

$$= \sum_{j=1}^{|\mathsf{pa}(n)|} \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \mathbb{1}[\boldsymbol{x} \in \gamma_{g_j} \cap \gamma_{m_j}]$$

$$= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \Big(\sum_{j=1}^{|\mathsf{pa}(n)|} \mathbb{1}[\boldsymbol{x} \in \gamma_{g_j} \cap \gamma_{m_j}] \Big). \tag{14}$$

Next, we show that $\forall m_i, m_j \in \mathsf{pa}(n)(m_i \neq m_j), \gamma_{m_i} \cap \gamma_{m_j} = \varnothing$. We prove this claim using its contrapositive form. Suppose there exists $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ such that $\boldsymbol{x} \in \gamma_{m_i}$ and $\boldsymbol{x} \in \gamma_{m_j}$. According to the definition of context, if $\boldsymbol{x} \in \gamma_{m_i}$, then there must be a path between m_i and the root node n_r where all nodes in the path are "activated", i.e., for any unit c in the path, $\boldsymbol{x} \in \gamma_c$. Similarly, there much exists a path of "activated" units between m_j and n_r. We note that the two paths must share a set of identical nodes since their terminal are both the root node n_r. Therefore, there must exist a sum unit n' along the intersection of the two path where at least two of its children are activated, i.e., $\exists c_1, c_2 \in \mathsf{in}(n')(c_1 \neq c_2)$, such that $\boldsymbol{x} \in \gamma_{c_1}$ and $\boldsymbol{x} \in \gamma_{c_2}$. This contradicts the assumption that the PC is deterministic. Therefore, the claim at the beginning of this paragraph holds. Thus,

$$\begin{aligned}
\sum_{j=1}^{|\mathsf{pa}(n)|} \mathbb{1}[\boldsymbol{x} \in \gamma_{g_j} \cap \gamma_{m_j}] &\overset{(a)}{=} \sum_{j=1}^{|\mathsf{pa}(n)|} \mathbb{1}[\boldsymbol{x} \in \gamma_{m_j}] \\
&\overset{(b)}{=} \mathbb{1}[\boldsymbol{x} \in \bigcup\nolimits_{j=1}^{|\mathsf{pa}(n)|} \gamma_{m_j}] \\
&\overset{(c)}{=} \mathbb{1}[\boldsymbol{x} \in \bigcup\nolimits_{j=1}^{|\mathsf{pa}(n)|} \gamma_{m_j} \cap \mathsf{supp}(n)] \\
&\overset{(d)}{=} \mathbb{1}[\boldsymbol{x} \in \gamma_n],
\end{aligned}$$

where (a) follows from $\gamma_{m_j} \subseteq \gamma_{g_j}$; (b) holds because the statement made in the previous paragraph (i.e., $\forall m_i, m_j \in \mathsf{pa}(n)(m_i \neq m_j), \gamma_{m_i} \cap \gamma_{m_j} = \varnothing$); (c) holds since $\mathsf{supp}(m_j) \subseteq \mathsf{supp}(n)$ and $\gamma_{m_j} \subseteq \mathsf{supp}(m_j)$; (d) directly applies the definition of context (i.e., Def. 5).

Plug in Eq. (14), we have

$$\begin{aligned}
\texttt{context}_i[n] &= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \Big(\sum_{j=1}^{|\mathsf{pa}(n)|} \mathbb{1}[\boldsymbol{x} \in \gamma_{g_j} \cap \gamma_{m_j}] \Big) \\
&= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} H(\boldsymbol{x}, \boldsymbol{x}^{(i)}, k) \cdot \mathbb{1}[\boldsymbol{x} \in \gamma_n].
\end{aligned}$$

Computing $\mathrm{F}_{n,c}(\mathcal{D}_\beta)$ Finally, we can compute $\mathrm{F}_{n,c}(\mathcal{D}_\beta)$ from the flows (i.e., $\texttt{flow}_i[n, c]$) computed by Alg. 2:

$$\begin{aligned}
\mathrm{F}_{n,c}(\mathcal{D}_\beta) &= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \texttt{weight}(\mathcal{D}_\beta, \boldsymbol{x}) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n \wedge \boldsymbol{x}\in\gamma_c] \\
&= \sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \underbrace{\sum_{i=1}^{N} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big)}_{\texttt{weight}(\mathcal{D}_\beta, \boldsymbol{x})} \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n \wedge \boldsymbol{x}\in\gamma_c] \\
&= \sum_{i=1}^{N} \underbrace{\sum_{\boldsymbol{x}\in\mathsf{val}(\mathbf{X})} \prod_{k=1}^{K} \Big(\beta\cdot\mathbb{1}[x_k^{(i)} = x_k] + (1-\beta)\cdot\mathbb{1}[x_k^{(i)} \neq x_k]\Big) \cdot \mathbb{1}[\boldsymbol{x}\in\gamma_n \wedge \boldsymbol{x}\in\gamma_c]}_{\texttt{flow}_i[n,c]} \\
&= \sum_{i=1}^{N} \texttt{flow}_i[n, c].
\end{aligned}$$

Finally, we note that Alg. 1 and 2 both run in time $\mathcal{O}(|p|\cdot|\mathcal{D}|)$.

Algorithm 4 PC entropy

Input: A deterministic PC p
Output: $\texttt{entropy}[n] := \texttt{ENT}(p_n)$ for every unit n
foreach n traversed in postorder **do**
  if n **isa** input unit **then** $\texttt{entropy}[n] = \texttt{ENT}(p_n)$ //entropy of the input distribution
  elif n **isa** product unit **then** $\texttt{entropy}[n] = \sum_{c \in \mathsf{in}(n)} \texttt{entropy}[c]$
  else //n is a sum unit **then**

$$\texttt{entropy}[n] = -\sum_{c \in \mathsf{in}(n)} \theta_{n,c} \log \theta_{n,c} + \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot \texttt{entropy}[c]$$

A.2 Useful Lemmas

This section provides several useful lemmas that are later used in the proof of Thm. 4.

Lemma 1. *Given a deterministic PC p whose root node is n_r, its entropy $\texttt{ENT}(p) \overset{def}{=} \texttt{ENT}(p_{n_r})$ can be decomposed recursively as follows:*

$$\texttt{ENT}(p_n) = \begin{cases} \sum_{c \in \mathsf{in}(n)} \big(-\theta_{n,c} \log \theta_{n,c} + \theta_{n,c} \cdot \texttt{ENT}(p_c) \big) & \text{if } n \text{ is a sum unit}, \\ \sum_{c \in \mathsf{in}(n)} \texttt{ENT}(p_c) & \text{if } n \text{ is a product unit}, \end{cases}$$

where the entropy of an input unit is defined by the entropy of the corresponding univariate distribution. Following this decomposition, we construct Alg. 4 that computes the entropy of every nodes in a deterministic PC in $\mathcal{O}(|p|)$ time.

Proof. We show the correctness of the entropy decomposition over a sum unit and a product unit respectively.

• Sum units. If n is a sum unit:

$$\begin{aligned}
\texttt{ENT}(p_n) &= -\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} p_c(\boldsymbol{x}) \Big) \log \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} p_c(\boldsymbol{x}) \Big) \\
&= -\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} p_c(\boldsymbol{x}) \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] \Big) \log \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} p_c(\boldsymbol{x}) \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] \Big) \\
&\overset{(a)}{=} -\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} \sum_{c \in \mathsf{in}(n)} \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] \cdot \theta_{n,c} \cdot p_c(\boldsymbol{x}) \cdot \Big(\log \theta_{n,c} + \log p_c(\boldsymbol{x}) \Big) \\
&= -\sum_{c \in \mathsf{in}(n)} \theta_{n,c} \log \theta_{n,c} \underbrace{\Big(\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] p_c(\boldsymbol{x}) \Big)}_{=1} \\
&\qquad + \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \underbrace{\Big(-\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} p_c(\boldsymbol{x}) \log p_c(\boldsymbol{x}) \Big)}_{=\texttt{ENT}(p_c)} \\
&= \sum_{c \in \mathsf{in}(n)} \Big(-\theta_{n,c} \log \theta_{n,c} + \theta_{n,c} \cdot \texttt{ENT}(p_c) \Big), \qquad (15)
\end{aligned}$$

where (a) uses the assumption that the sum unit is deterministic, i.e., $\forall c_1, c_2 \in \mathsf{in}(n)\ (c_1 \neq c_2), \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \varnothing$.

• Product units. If n is a product unit:

$$\begin{aligned}
\texttt{ENT}(p_n) &= -\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(n))} \Big(\prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) \Big) \log \Big(\prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) \Big) \\
&= -\sum_{c \in \mathsf{in}(n)} \Big(\sum_{\boldsymbol{x} \in \mathsf{val}(\phi(c))} p_c(\boldsymbol{x}) \log p_c(\boldsymbol{x}) \Big)
\end{aligned}$$

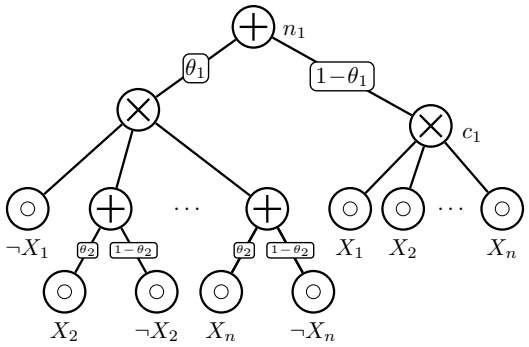


Figure 7: An example PC to show that PC entropy is neither convex nor concave.

$$= \sum_{c \in \mathsf{in}(n)} \mathtt{ENT}(p_c). \tag{16}$$

□

Lemma 2. *The entropy of a deterministic PC p is neither convex nor concave w.r.t. its parameters.*

Proof. Consider the example PC in Fig. 7. Assume $n = 20$ and define parameters $\boldsymbol{\theta}_a = \{\theta_1 = 0.1, \theta_2 = 0.1\}$ and $\boldsymbol{\theta}_b = \{\theta_1 = 0.12, \theta_2 = 0.12\}$. Denote $\boldsymbol{\theta}_c = (\boldsymbol{\theta}_a + \boldsymbol{\theta}_b)/2$, we have

$$2 \cdot \mathtt{ENT}(p; \boldsymbol{\theta}_c) - \mathtt{ENT}(p; \boldsymbol{\theta}_a) - \mathtt{ENT}(p; \boldsymbol{\theta}_b) \approx -0.0047898 < 0.$$

Hence the entropy is not concave.

Define parameters $\boldsymbol{\theta}_d = \{\theta_1 = 0.4, \theta_2 = 0.8\}$ and $\boldsymbol{\theta}_e = \{\theta_1 = 0.42, \theta_2 = 0.82\}$. Denote $\boldsymbol{\theta}_f = (\boldsymbol{\theta}_d + \boldsymbol{\theta}_e)/2$, we have

$$2 \cdot \mathtt{ENT}(p; \boldsymbol{\theta}_f) - \mathtt{ENT}(p; \boldsymbol{\theta}_d) - \mathtt{ENT}(p; \boldsymbol{\theta}_e) \approx 0.0056294 > 0.$$

Hence the entropy is not convex. □

Lemma 3. *For any dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ and any deterministic PC p with parameters $\boldsymbol{\theta}$, the following formula is concave w.r.t. $\boldsymbol{\theta}$:*

$$\sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}). \tag{17}$$

Proof. For any input $\boldsymbol{x}$, $\log p(\boldsymbol{x}; \boldsymbol{\theta})$ can be decomposed over sum and product units:

• Sum units. Suppose n is a sum unit, then

$$\begin{aligned}
\log p_n(\boldsymbol{x}; \boldsymbol{\theta}) &= \log \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) \Big) \\
&= \log \Big(\sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] \Big) \\
&= \sum_{c \in \mathsf{in}(n)} \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(c)] \big(\log \theta_{n,c} + \log p_c(\boldsymbol{x}) \big),
\end{aligned} \tag{18}$$

where the last equation holds because unit n is deterministic: $\forall c_i, c_j \in \mathsf{in}(n)(c_i \neq c_j), \mathsf{supp}(c_i) \cap \mathsf{supp}(c_j) = \varnothing$.

• Product units. Suppose n is a product unit, then

$$\log p_n(\boldsymbol{x}; \boldsymbol{\theta}) = \log \Big(\prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) \Big) = \sum_{c \in \mathsf{in}(n)} \log p_c(\boldsymbol{x}). \tag{19}$$

According to Eqs. (18) and (19), for any $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$, $\log p(\boldsymbol{x}; \boldsymbol{\theta})$ can be decomposed into the sum over a set of log-parameters (e.g., $\log \theta_{n,c}$). Therefore, Eq. (17) is concave. □

Lemma 4. *Given a deterministic PC p with root node n_r, its entropy $\texttt{ENT}(p)$ can be decomposed as follows:*

$$\texttt{ENT}(p_{n_r}) = -\sum_{(n,c)\in\texttt{edges}(p_{n_r})} P_{n_r}(n)\cdot\theta_{n,c}\log\theta_{n,c},$$

where $\texttt{edges}(p)$ denotes all edges (n,c) in the PC with sum unit n; $P_{n_r}(n)$ is defined in Eq. (21)*.*

Proof. We prove the lemma by induction.

• Base case. Suppose m is a sum unit such that all its decendents are either input units or product unit. By definition, we have $P_m(m)=1$, and $\texttt{edges}(p_m)=\{(m,c)\mid c\in\mathsf{in}(m)\}$. Thus,

$$-\sum_{(n,c)\in\texttt{edges}(p_m)} P_m(n)\cdot\theta_{n,c}\log\theta_{n,c} = -\sum_{c\in\mathsf{in}(m)}\theta_{m,c}\log\theta_{m,c} = \texttt{ENT}(p_m).$$

• Inductive case: product units. Suppose m is a product unit such that for each of its children $c\in\mathsf{in}(m)$, we have

$$\texttt{ENT}(p_c) = -\sum_{(n',c')\in\texttt{edges}(p_c)} P_c(n')\cdot\theta_{n',c'}\log\theta_{n',c'}.$$

Then by Lem. 1 we know that

$$\begin{aligned}
\texttt{ENT}(p_m) &= \sum_{c\in\mathsf{in}(m)}\texttt{ENT}(p_m)\\
&= -\sum_{c\in\mathsf{in}(m)}\sum_{(n',c')\in\texttt{edges}(p_c)} P_c(n')\cdot\theta_{n',c'}\log\theta_{n',c'}\\
&\overset{(a)}{=} -\sum_{c\in\mathsf{in}(m)}\sum_{(n',c')\in\texttt{edges}(p_c)} P_m(n')\cdot\theta_{n',c'}\log\theta_{n',c'}\\
&\overset{(b)}{=} -\sum_{(n',c')\in\texttt{edges}(p_m)} P_m(n')\cdot\theta_{n',c'}\log\theta_{n',c'},
\end{aligned}$$

where (a) holds since for any sum unit n', $P_c(n')=P_m(n')$, and (b) follows from the fact that $\texttt{edges}(p_m)=\bigcup_{c\in\mathsf{in}(m)}\texttt{edges}(p_c)$.

• Inductive case: sum units. Suppose m is a sum unit such that for each of its children $c\in\mathsf{in}(m)$, we have

$$\texttt{ENT}(p_c) = -\sum_{(n',c')\in\texttt{edges}(p_c)} P_c(n')\cdot\theta_{n',c'}\log\theta_{n',c'}.$$

Then by Lem. 1 we have

$$\begin{aligned}
\texttt{ENT}(p_m) &= \sum_{c\in\mathsf{in}(m)}\big(-\theta_{n,c}\log\theta_{n,c}+\theta_{n,c}\cdot\texttt{ENT}(p_c)\big)\\
&= \sum_{c\in\mathsf{in}(m)} -\theta_{n,c}\log\theta_{n,c} - \sum_{c\in\mathsf{in}(n)}\sum_{(n',c')\in\texttt{edges}(p_c)}\underbrace{\theta_{m,c}\cdot P_c(n')}_{P_m(n')}\cdot\theta_{n',c'}\log\theta_{n',c'}\\
&= \sum_{c\in\mathsf{in}(m)} -P_m(m)\theta_{n,c}\log\theta_{n,c} - \sum_{c\in\mathsf{in}(n)}\sum_{(n',c')\in\texttt{edges}(p_c)} P_m(n')\cdot\theta_{n',c'}\log\theta_{n',c'}\\
&\overset{(a)}{=} -\sum_{(n',c')\in\texttt{edges}(p_m)} P_m(n')\cdot\theta_{n',c'}\log\theta_{n',c'},
\end{aligned}$$

where (a) holds because $\texttt{edges}(p_m)=\big(\bigcup_{c\in\mathsf{in}(m)}\texttt{edges}(p_c)\big)\bigcup\big(\{(m,c)\mid c\in\mathsf{in}(m)\}\big)$. □

Lemma 5. *The entropy regularization objective in Eq.* (2) *w.r.t. a deterministic PC p and a dataset $\mathcal{D}$ could have multiple local maximas.*

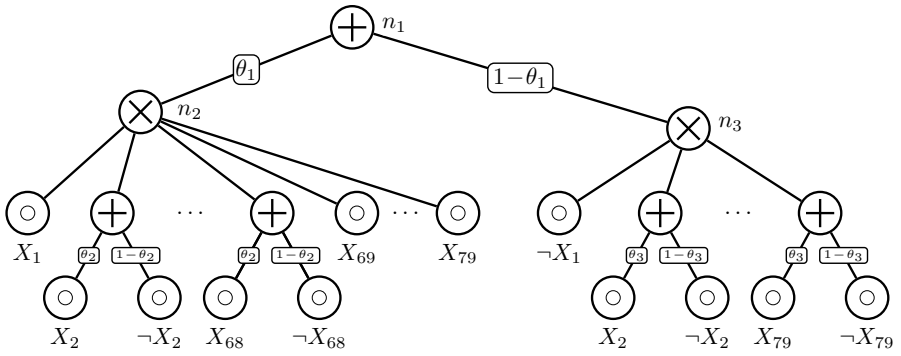


Figure 8: An example PC to show that Eq. (2) could have multiple stationary points.

Proof. Consider the deterministic PC p in Fig. 8 and dataset $\mathcal{D}$ with a single sample $\boldsymbol{x} = (\texttt{true}, \ldots, \texttt{true})$. The objective in Eq. (2) can be re-written as follows

$$\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau) := \log p_{n_1}(\boldsymbol{x}) + \tau \cdot \texttt{ENT}(p_{n_1}),$$

where n_1 is the root node of the PC as denoted in Fig. 8. We further decompose $\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau)$:

$$\begin{aligned}\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau) &= \log\theta_1 + 67 \cdot \log\theta_2 + \tau \cdot \texttt{ENT}(p_{n_1}) \\ &= \log\theta_1 + 67 \cdot \log\theta_2 + \tau \cdot \big(-\theta_1 \log\theta_1 - (1-\theta_1)\log(1-\theta_1)\big) \\ &\quad + \tau \cdot \theta_1 \cdot \texttt{ENT}(p_{n_2}) + \tau \cdot (1-\theta_1) \cdot \texttt{ENT}(p_{n_3}).\end{aligned}$$

First, we observe that to maximize $\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau)$, θ_3 should always be 0.5 since the only term that depends on θ_3 is $(1-\theta_1) \cdot \texttt{ENT}(p_{n_3})$ and $1-\theta_1 > 0$. Therefore, we have

$$\texttt{ENT}(p_{n_3}) = 78 \cdot \log 2 \approx 54.065.$$

Next, for any fixed $\theta_1 \in (0, 1]$, the objective $\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau)$ is concave w.r.t. θ_2:

$$\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau) = 67 \cdot \big(\log\theta_2 - \tau \cdot \theta_1 \cdot (\theta_2 \log\theta_2 + (1-\theta_2)\log(1-\theta_2))\big) + \text{const}, \tag{20}$$

where the constant term does not depend on θ_2. Therefore, for any θ_1, we can uniquely compute the optimal value of θ_2. We are left with determining the optimal value of θ_1. Choose $\tau = 1.5$, the derivative of $\mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau)$ w.r.t. θ_1 is (denote $\texttt{ent}_0 := -(\theta_2 \log\theta_2 + (1-\theta_2)\log(1-\theta_2))$)

$$\begin{aligned}g(\theta_1) := \frac{\partial \mathcal{L}_{\text{ent}}(\boldsymbol{\theta}; n_1, \tau)}{\partial \theta_1} &= \frac{1}{\theta_1} + 1.5 \cdot \Big(\log(1-\theta_1) - \log\theta_1 + \texttt{ENT}(p_{n_2}) - \texttt{ENT}(p_{n_3})\Big) \\ &= \frac{1}{\theta_1} + 1.5 \cdot \Big(\log(1-\theta_1) - \log\theta_1 + 67 \cdot \texttt{ent}_0 - \texttt{ENT}(p_{n_3})\Big)\end{aligned}$$

where $\texttt{ENT}(p_{n_3})$ can be viewed as a constant and $\texttt{ent}_0$ depends on θ_1. Specifically, for any θ_1, we compute θ_2 and hence $\texttt{ent}_0$ by maximizing Eq. (20). Putting everything together, we have

$$\begin{cases} g(0.02) \approx 1.772730 > 0, \\ g(0.7) \approx -0.190743 < 0, \\ g(0.9) \approx 2.055231 > 0, \\ g(0.99) \approx -0.216938 < 0. \end{cases}$$

Since g is continuous in range $(0, 1]$, there exists a local maxima of θ_1 between 0.02 and 0.7 as well as between 0.9 and 0.99. Therefore, the entropy regularization objective could have multiple local maximas.

□

A.3 Proof of Theorem 1

This theorem is a direct corollary of Theorem 5 in [49], which has the following statement:

Computing the expectation of a logistic regression model w.r.t. a uniform data distribution is #-hard.

Note that with $\beta = 0.5$, the distribution $\mathcal{D}_\beta$ is essentially uniform, Thm. 1 follows directly from [49].

A.4 Proof of Theorem 2

This proof largely follows the proof of Theorem 5 in [49]. The proof is by reduction from #NUMPAR, which is defined as follows. Given n positive integers $k_1, \ldots, k_n$, we want to count the number of subset $S \subseteq [n]$ that satisfies $\sum_{i \in S} k_i = \sum_{i \notin S} k_i$. #NUMPAR is known to be #P-hard.

Fix an instance of #NUMPAR, $k_1, \ldots, k_n$, and assume w.l.o.g. that the sum of the numbers is even, i.e., $\sum_i k_i = 2c$ for some natural number c. Define $P := \{S \mid S \subseteq [n], \sum_{i \in S} k_i = c\}$. By definition $|P|$ is the solution to the #NUMPAR problem. Note that for each $S \in P$, its complement $\bar{S}$ should also be a member of P, and hence $|P|$ is even.

Define a logistic regression model as $F(x_1, \ldots, x_n) := \sigma(w_0 + \sum_{i=1}^n w_i \cdot x_i)$, where σ is the sigmoid function. Define the normalized model of F as $G(x_1, \ldots, x_n) := F(x_1, \ldots, x_n)/Z$, where $Z := \sum_{\boldsymbol{x} \in \mathsf{val}(X)} F(x_1, \ldots, x_n)$. Denote the entropy of a normalized logistic regressor G as $\texttt{ENT}(G) := -\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} G(\boldsymbol{x}) \log G(\boldsymbol{x})$.

We now describe an algorithm that computes $|P|$ using an oracle for $\texttt{ENT}(G)$, where G is a normalized logistic regression model. Denote m as a large natural number to be chosen later, and define the following weights

$$w_0 := -\frac{m}{2} - mc, \quad w_i := mk_i (\forall i \in [n]).$$

Let F be the logistic regressor corresponds to the above weights and G the normalized model of F. We can represent $\texttt{ENT}(G)$ as follows:

$$\begin{aligned}\texttt{ENT}(G) = -\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \frac{F(\boldsymbol{x})}{Z} \log \frac{F(\boldsymbol{x})}{Z} &= -\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \Big(\frac{F(\boldsymbol{x}) \log F(\boldsymbol{x})}{Z} - \frac{F(\boldsymbol{x})}{Z} \log Z \Big) \\ &= -\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \frac{F(\boldsymbol{x}) \log F(\boldsymbol{x})}{Z} + \log Z.\end{aligned}$$

For large enough m, $F(\boldsymbol{x})$ will approach either 0 or 1. Therefore, the first term in the above equation will approach 0. Therefore, for large enough m, we have

$$\texttt{ENT}(G) \approx \log Z = \log \Big(\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \sigma(w_0 + \sum_{i=1}^n w_i \cdot x_i) \Big) = \log \Big(\sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \sigma(w_0 + \sum_{i=1}^n w_i \cdot x_i) \Big).$$

For each $S \subseteq [n]$, we define $\text{weight}(S) := -\frac{m}{2} - mc + m(\sum_{i \in S} k_i)$. Then,

$$\begin{aligned}\exp(\texttt{ENT}(G)) &\approx \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \sigma(-\frac{m}{2} - mc + m(\sum_{i \in [n]} k_i x_i)) \\ &= \sum_{\boldsymbol{x} \in \mathsf{val}(\mathbf{X})} \sigma(-\frac{m}{2} - mc + m(\sum_{i : x_i = 1} k_i)) \\ &= \sum_{S \subseteq [n]} \sigma(\text{weight}(S)) \\ &= \frac{1}{2} \sum_{S \subseteq [n]} \big(\sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \big).\end{aligned}$$

If S is a solution to #NUMPAR, then

$$\sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) = 2\sigma(-m/2).$$

Othervise, one of $\text{weight}(S)$ and $\text{weight}(\bar{S})$ is $\geq m/2$ and the other is $\leq -3m/2$, and hence

$$\sigma(m/2) \leq \sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \leq 1 + \sigma(-3m/2).$$

For a large enough m such that $2\sigma(-m/2) < \epsilon$ and $1 - \sigma(m/2) < \epsilon$, we have

$$\begin{aligned}S \in P: &\quad 0 \leq \sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \leq \epsilon, \\ S \notin P: &\quad 1 - \epsilon \leq \sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \leq 1 + \epsilon.\end{aligned}$$

Therefore, we have

$$\frac{2^n - |P|}{2}(1-\epsilon) \leq \exp(\texttt{ENT}(G)) \leq \frac{|P|}{2}\epsilon + \frac{2^n - |P|}{2}(1+\epsilon)$$

$$|P| \geq 2^n - \frac{2\exp(\texttt{ENT}(G))}{1-\epsilon}$$

$$|P| \leq 2^n(1+\epsilon) - 2\exp(\texttt{ENT}(G))$$

This gives a lower and upper bound for $|P|$. For small enough ϵ (governed by large enough m), the difference between the lower and upper bound is less than 1, and hence $|P|$ can be uniquely determined, which proves the theorem.

A.5 Proof of Theorem 4

First note that according to Lem. 2, Eq. (2) is not a convex optimization problem. The key idea of Alg. 3 is to propose a set of *surrogate objective* functions, and maximize the objective function Eq. (2) by iteratively maximizing the surrogate objective. Concretely, we show the monotonic convergence property of Alg. 3 by checking the correctness of the following three statements:

• **Statement #1:** The surrogate objective is easy to maximize as it is a concave function w.r.t. the parameters.

• **Statement #2:** The surrogate objective is consistent with the original objective Eq. (2). That is, whenever a set of surrogate objectives are improved, the true objective is also improved.

• **Statement #3:** The surrogate objectives can always be improved unless the original objective Eq. (2) has zero first-order derivative.

• **Statement #4:** Solving Eq. (5) is equivalent to maximizing the surrogate objective.

Before verifying the statements, we first formally define the surrogate. Denote $\texttt{ENT}(p_n; \boldsymbol{\theta})$ as the entropy of the PC rooted at n and with parameters $\boldsymbol{\theta}$; the top-down probability of n, denoted $P_{n_r}(n)$, is recursively defined as follows:

$$P_{n_r}(n) := \begin{cases} 1 & \text{if n is the root node } n_r, \\ \sum_{m \in \mathsf{pa}(n)} P_{n_r}(m) & \text{if n is a sum unit,} \\ \sum_{m \in \mathsf{pa}(n)} \theta_{m,n} \cdot P_{n_r}(m) & \text{if n is a product unit.} \end{cases} \tag{21}$$

Given a set of reference parameters $\boldsymbol{\theta}^{\text{ref}}$, we define the surrogate objective w.r.t. parameter $\theta_{n,c}$ as

$$\begin{aligned} \mathcal{L}_{\text{surr}}(\theta_{n,c}; \boldsymbol{\theta}^{\text{ref}}) := & \underbrace{\frac{1}{N}\sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}^{\text{ref}} \backslash \{\theta^{\text{ref}}_{n,c}\}, \theta_{n,c})}_{\text{Term 1}} \\ & + \underbrace{\tau \cdot P_{n_r}(n; \boldsymbol{\theta}^{\text{ref}}) \cdot \Big(-\theta_{n,c}\log\theta_{n,c} + \theta_{n,c} \cdot \texttt{ENT}(p_c; \boldsymbol{\theta}^{\text{ref}})\Big)}_{\text{Term 2}}. \end{aligned} \tag{22}$$

Given parameters $\boldsymbol{\theta}^{\text{old}}$, we now describe an update procedure to obtain a set of new parameters $\boldsymbol{\theta}^{\text{new}}$.

Parameter update procedure We start with an empty set of parameters $\boldsymbol{\theta}^{\text{update}} := \boldsymbol{\theta}^{\text{old}}$ and iteratively update its entries with updated parameters $\theta^{\text{new}}_{n,c}$. For every sum unit n traversed in pre-order, we update the parameters $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$ by maximizing the sum of surrogate objectives:

$$\sum_{c \in \mathsf{in}(n)} \mathcal{L}_{\text{surr}}(\theta_{n,c}; \boldsymbol{\theta}^{\text{update}}). \tag{23}$$

After solving the above equation, the updated parameters $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$ replace the corresponding original parameters in $\boldsymbol{\theta}^{\text{update}}$. As we will proceed to show in statement #4, maximizing Eq. (23) is done in Lines 7 to 7 in Alg. 3.

Given the formal definition of the surrogate objective and the corresponding update process, we re-state the three statements and prove their validity in the following.

• **Statement #1:** The surrogate objective Eq. (23) is concave w.r.t. parameters $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$.

Proof. This statement can be proved by showing that $\forall(n,c), \forall\boldsymbol{\theta}, \mathcal{L}_{\text{surr}}(\theta_{n,c};\boldsymbol{\theta})$ is concave. Specifically, according to Lem. 3, the first term of Eq. (22) is concave; the second term of Eq. (22) is concave since (i) $-x\log x$ is concave w.r.t. x, and (ii) $P_{n_r}(n;\boldsymbol{\theta}^{\text{ref}})$ and $\texttt{ENT}(p_c;\boldsymbol{\theta}^{\text{ref}})$ are independent of $\{\theta_{n,c'} \mid c' \in \mathsf{in}(n)\}$. □

• **Statement #2:** For any sum unit n and any parameters $\boldsymbol{\theta}$, if we update n's parameters (i.e., $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$) by maximizing Eq. (23), the true objective Eq. (2) will also improve.

Proof. Consider updating the parameters correspond to sum unit n (i.e., $\{\theta_{n,c} \mid c \in \mathsf{in}(n)\}$) by maximizing Eq. (23). We can re-arrange the entropy $\texttt{ENT}(p_{n_r})$ as follows:

$$
\begin{aligned}
\texttt{ENT}(p_{n_r}) &\overset{(a)}{=} -\sum_{(n',c')\in\mathsf{edges}(p_{n_r})} P_{n_r}(n')\cdot\theta_{n',c'}\log\theta_{n',c'} \\
&= -\sum_{(n',c')\in\mathsf{edges}(p_n)} P_{n_r}(n')\cdot\theta_{n',c'}\log\theta_{n',c'} + \text{const} \\
&= -\sum_{(n',c')\in\mathsf{edges}(p_n)} \Big(\sum_{m\in\mathsf{pa}(n)} P_{n_r}(m)\Big)\cdot P_n(n')\cdot\theta_{n',c'}\log\theta_{n',c'} + \text{const} \\
&= -\sum_{(n',c')\in\mathsf{edges}(p_n)} P_{n_r}(n)\cdot P_n(n')\cdot\theta_{n',c'}\log\theta_{n',c'} + \text{const} \\
&= P_{n_r}(n)\cdot\texttt{ENT}(p_n) + \text{const} \\
&\overset{(b)}{=} P_{n_r}(n)\cdot\sum_{c\in\mathsf{in}(n)}\Big(-\theta_{n,c}\log\theta_{n,c}+\theta_{n,c}\cdot\texttt{ENT}(p_c)\Big) + \text{const},
\end{aligned}
$$

where const denotes terms that do not depend on $\{\theta_{n,c'} \mid c' \in \mathsf{in}(n)\}$; (a) and (b) directly apply Lem. 4 and Lem. 1, respectively.

Thus, the true objective Eq. (2) can be written as follows:

$$
\begin{aligned}
&\frac{1}{N}\sum_{i=1}^{N}\log p(\boldsymbol{x}^{(i)}) + \tau\cdot\texttt{ENT}(p) \\
=&\frac{1}{N}\sum_{i=1}^{N}\log p(\boldsymbol{x}^{(i)}) + \tau\cdot P_{n_r}(n)\cdot\sum_{c\in\mathsf{in}(n)}\Big(-\theta_{n,c}\log\theta_{n,c}+\theta_{n,c}\cdot\texttt{ENT}(p_c)\Big) + \text{const}
\end{aligned}
\tag{24}
$$

Compare Eq. (24) and Eq. (22), we can see that they only differs in some constant terms. Therefore, maximizing Eq. (22) w.r.t. $\{\theta_{n,c'} \mid c' \in \mathsf{in}(n)\}$ will lead to an increase in the true objective Eq. (2). □

• **Statement #3:** The surrogate objectives can always be improved unless the original objective Eq. (2) has zero first-order derivative.

Proof. Recall from Eq. (24) that for any sum unit n, the true objective Eq. (2) can be written as the sum of Eq. (22) and terms that are independent with the parameters of n (i.e., $\{\theta_{n,c'} \mid c' \in \mathsf{in}(n)\}$). Therefore, the true objective can always be improved by maximizing the surrogate objective Eq. (23) as long as the true objective has non-zero first-order derivative w.r.t. the parameters. □

• **Statement #4:** Solving Eq. (5) is equivalent to maximizing the surrogate objective.

Proof. We want to maximize the surrogate objective given the assumption that the parameters w.r.t. a sum unit sum up to 1:

$$
\underset{\theta_{n,c}}{\text{maximize}}\,\mathcal{L}_{\text{surr}}(\theta_{n,c};\boldsymbol{\theta}^{\text{ref}}),\ \text{such that}\ \sum_{c\in\mathsf{in}(n)}\theta_{n,c} = 1. \tag{25}
$$

Algorithm 5 Forward pass (expected flows)

Input: A non-deterministic PC p; sample $\boldsymbol{x}$
Output: $\texttt{value}[n] := (\boldsymbol{x} \in \mathsf{supp}(n))$ for each unit n
foreach n traversed in postorder **do**
  if n **isa** input unit **then** $\texttt{value}[n] \leftarrow f_n(\boldsymbol{x})$
  elif n **isa** product unit **then**
    $\texttt{value}[n] \leftarrow \prod_{c \in \mathsf{in}(n)} \texttt{value}[c]$
  else //n is a sum unit
    $\texttt{value}[n] \leftarrow \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot \texttt{value}[c]$

Algorithm 6 Backward pass (expected flows)

Input: A non-deterministic PC p; $\forall n, \texttt{value}[n]$
Output: $\texttt{eflow}[n, c] := \mathbb{E}_{\boldsymbol{z} \in p_c(\cdot|\boldsymbol{x};\boldsymbol{\theta})}((\boldsymbol{x}, \boldsymbol{z}) \in (\gamma_n \cap \gamma_c))$ for each pair (n, c), where n is a sum unit and $c \in \mathsf{in}(n)$
$\forall n, \texttt{context}[n] \leftarrow 0$; $\texttt{context}[n_r] \leftarrow \texttt{value}[n_r]$
foreach sum unit n traversed in preorder **do**
  foreach $m \in \mathsf{pa}(n)$ **do** (denote $g \leftarrow \mathsf{pa}(m)$)
    $\texttt{f} \leftarrow \frac{\texttt{value}[m]}{\texttt{value}[g]} \cdot \texttt{context}[g] \cdot \theta_{g,m}$
    $\texttt{context}[n] \mathrel{+}= \texttt{f}$; $\texttt{flow}[g, m] = \texttt{f}$

Since the surrogate objective $\mathcal{L}_{\text{surr}}(\theta_{n,c}; \boldsymbol{\theta}^{\text{ref}})$ is concave, maximizing the surrogate objective is equivalent to finding its stationary point. Specifically, we solve Eq. (25) with the Lagrange multiplier method (variable λ corresponds to the constraint):

$$\underset{\theta_{n,c}}{\text{maximize}} \, \underset{\lambda}{\text{minimize}} \, \mathcal{L}_{\text{surr}}(\theta_{n,c}; \boldsymbol{\theta}^{\text{ref}}) - \lambda(1 - \sum_{c \in \mathsf{in}(n)} \theta_{n,c})$$

Its KKT conditions can be written as:

$$\begin{cases} \frac{\mathrm{F}_{n,c_i}(\mathcal{D})}{|\mathcal{D}| \cdot \theta_{n,c_i}} - \tau \cdot P_{n_r}(n; \boldsymbol{\theta}^{\text{ref}})(\log \theta_{n,c_i} + 1 + \texttt{ENT}(p_{c_i}; \boldsymbol{\theta}^{\text{ref}})) + \lambda = 0 \quad (\forall 1 \le i \le |\mathsf{in}(n)|), \\ \sum_{c \in \mathsf{in}(n)} \theta_{n,c} = 1. \end{cases}$$

It is easy to verify that the above equation is equivalent to Eq. (5) by substituting the definitions in Lines 7-8 in Alg. 3.

□

Therefore, by following the parameter update procedure, we can always make progress since the surrogate objective is concave (statement #1) and the true objective improves as long as the surrogate objective increases (statement #2). Finally, the learning procedure will not terminate unless a local maximum is achieved (statement #3).

A.6 Correctness of Algorithms 1 and 2

The correctness of Alg. 1 and 2 can be justified directly by the proof of Thm. 3. Specifically, since with $\beta = 1$, the softened dataset $\mathcal{D}_\beta$ is equivalent to $\mathcal{D}$, we can use the proof in Appendix A.1 and set $\beta = 1$ (the proof holds for any $\beta \in (0.5, 1]$).

A.7 Proof of Proposition 1

The first statement (i.e., Eq. (2)) could be non-concave) is proved in Lem. 2. The second statement (i.e., Eq. (2) could have multiple local maximas) is proved in Lem. 5.

B Method or Experiment Details

B.1 Soften non-boolean datasets

As a direct extension of softening boolean datasets, datasets with categorical variables can be similarly softened. Suppose X is a categorical variable with k categories. For an assignment $x = j$, we can soften it as follows

$$\begin{cases} P(x = i) = \frac{1-\beta}{k} \quad (i \neq j), \\ P(x = j) = \beta. \end{cases}$$

To compute the flow $F_{n,c}(\mathcal{D}_\beta)$ w.r.t. a softened categorical dataset, we can again adopt Alg. 1 and 2 by choosing

$$f_n(\boldsymbol{x}) = \beta \cdot \mathbb{1}[\boldsymbol{x} \in \mathsf{supp}(n)] + \frac{1-\beta}{k} \cdot \mathbb{1}[\boldsymbol{x} \notin \mathsf{supp}(n)].$$

B.2 Solving Equation 5

Denote $\gamma_{c_i} := \texttt{entropy}[c_i]$, our goal is to solve the following set of equations:

$$\begin{cases} d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \gamma_{c_i} = y \quad (\forall i \in \{1, \dots, |\mathsf{in}(n)|\}), \\ \sum_{i=1}^{|\mathsf{in}(n)|} e^{\varphi_{n,c_i}} = 1. \end{cases}$$

We break down the problem by iteratively solve for $\{\varphi_{n,c_i}\}_{i=1}^{|\mathsf{in}(n)|}$ and y, respectively.

• Solve for y. Given variables $\{\varphi_{n,c_i}\}_{i=1}^{|\mathsf{in}(n)|}$, we update y as

$$y = \frac{1}{|\mathsf{in}(n)|} \sum_{i=1}^{|\mathsf{in}(n)|} d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \gamma_{c_i}.$$

• Solve for $\{\varphi_{n,c_i}\}_{i=1}^{|\mathsf{in}(n)|}$. Given y, we first update each φ_{n,c_i} individually by solving the equation

$$d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \gamma_{c_i} = y.$$

Specifically, this is done by iterative Newton method update:

$$\varphi_{n,c_i} \mathrel{+}= \frac{\frac{d_i}{\varphi_{n,c_i}} + b \cdot (\gamma_{c_i} - \varphi_{n,c_i}) + y}{\frac{d_i}{\varphi_{n,c_i}} + b}$$

After one Newton method update step for every parameter in $\{\varphi_{n,c_i}\}_{i=1}^{|\mathsf{in}(n)|}$, we enforce the constraint $\sum_{i=1}^{|\mathsf{in}(n)|} e^{\varphi_{n,c_i}} = 1$ by

$$\varphi_{n,c_i} \mathrel{-}= \log\Big(\sum_{i=1}^{|\mathsf{in}(n)|} e^{\varphi_{n,c_i}} \Big).$$

B.3 Details of the Experiments on Deterministic PCs

PC structures For each dataset, we adopt 16 PCs by running Strudel [17] for $\{1000, 1200, 1400, \dots, 4000\}$ iterations except for the dataset "dna", which we ran Strudel for $\{50, 100, 150, \dots, 800\}$ iterations since the learning algorithm takes significantly longer for this dataset.

Hyperparameters We always perform hyperparameter search using the validation set, and report the final performance on the test set. Whenever we use data softening or entropy regularization, we also add pseudocount $\alpha = 1$ since it yields better performance.

Server specifications All our experiments were run on a server with 72 CPUs, 512G Memory, and 2 TITAN RTX GPUs.

Detailed results See Table 3 for extended numerical results.

B.4 Details of the Experiments on Non-Deterministic PCs

The HCLT structure For the experiments on the twenty datasets, we set the hidden size of the HCLT structure as 12, i.e., every latent variable Z is a categorical variable with 12 categories. Additionally, following [17, 16], we learn a mixture of 4 HCLTs to achieve better performance. For the protein sequence dataset, we adopted a mixture of 2 HCLTs with hidden size 32.

Hyperparameters Due to the complexity of running EM iteratively, we were not able to perform a grid-search for hyperparameters since that would take too long. In our experiments, we tried the following sets of hyperparameters (for α, β, and τ): $(0.1, 1.0, 0.0)$, $(0.2, 1.0, 0.0)$, $(0.4, 1.0, 0.0)$, $(0.1, 0.998, 0.0)$, $(0.1, 1.0, 0.001)$, and $(0.1, 0.998, 0.001)$. Among these hyperparameter choices, $(0.1, 0.998, 0.001)$ achieved the best validation LL in most datasets, and thus we reported this set of results. Therefore, for non-deterministic PCs, it is also beneficial to combine both proposed regularization techniques.

Table 2: Full results on the 20 density estimation benchmarks. As an extension of Table 1, we report the average test-set log-likelihood of all baselines: Strudel [17], LearnPSDD [16], EinSumNet [13], LearnSPN [18], ID-SPN [47], and RAT-SPN [48].

Dataset	HCLT	EiNet	LearnSPN	ID-SPN	RAT-SPN	Strudel	LearnPSDD
accidents	-26.78	-35.59	-40.50	-26.98	-35.48	-29.46	-28.29
ad	-16.04	-26.27	-19.73	-19.00	-48.47	-16.52	-20.13
baudio	-39.77	-39.87	-40.53	-39.79	-39.95	-42.26	-41.51
bbc	-250.07	-248.33	-250.68	-248.93	-252.13	-258.96	-260.24
bnetflix	-56.28	-56.54	-57.32	-56.36	-56.85	-58.68	-58.53
book	-33.84	-34.73	-35.88	-34.14	-34.68	-35.77	-36.06
c20ng	-151.92	-153.93	-155.92	-151.47	-152.06	-160.77	-160.43
cr52	-84.67	-87.36	-85.06	-83.35	-87.36	-92.38	-93.30
cwebkb	-153.18	-157.28	-158.20	-151.84	-157.53	-160.50	-161.42
dna	-79.33	-96.08	-82.52	-81.21	-97.23	-87.10	-83.02
jester	-52.45	-52.56	-75.98	-52.86	-52.97	-55.30	-54.63
kdd	-2.18	-2.18	-2.18	-2.13	-2.12	-2.17	-2.17
kosarek	-10.66	-11.02	-10.98	-10.60	-10.88	-10.98	-10.99
msnbc	-6.05	-6.11	-6.11	-6.04	-6.03	-6.05	-6.04
msweb	-9.90	-10.02	-10.25	-9.73	-10.11	-10.19	-9.93
nltcs	-6.00	-6.01	-6.11	-6.02	-6.01	-6.06	-6.03
plants	-14.31	-13.67	-12.97	-12.54	-13.43	-13.72	-13.49
pumbs*	-23.32	-31.95	-24.78	-22.40	-32.53	-25.28	-25.40
tmovie	-50.69	-51.70	-52.48	-51.51	-53.63	-59.47	-55.41
tretail	-10.84	-10.91	-11.04	-10.85	-10.91	-10.90	-10.92

Table 3: Results comparing different regularization approaches using the 20 density estimation benchmarks. This table contains the part of the results summarized in Fig. 5. Specifically, we report performance of the PC generated by running Strudel [17] for 4,000 steps, except for dna, where we ran the learner for 1,000 steps.

Dataset	Laplace smoothing	Data softening	Entropy reg.	Data softening + Entropy reg.
accidents	-29.37	-29.37	-29.39	-29.37
ad	-16.39	-16.39	-16.55	-16.39
baudio	-42.89	-42.75	-42.78	-42.59
bbc	-258.82	-258.64	-258.71	-258.35
bnetflix	-59.51	-59.34	-59.19	-59.07
book	-36.93	-36.81	-37.05	-36.69
c20ng	-160.84	-160.80	-160.81	-160.73
cr52	-91.97	-91.91	-91.99	-91.86
cwebkb	-159.93	-159.78	-159.97	-159.67
dna	-95.63	-94.90	-95.24	-94.87
jester	-56.19	-55.95	-55.83	-55.62
kdd	-2.19	-2.18	-2.19	-2.17
kosarek	-11.03	-11.00	-11.04	-10.97
msnbc	-6.04	-6.04	-6.04	-6.04
msweb	-10.11	-10.08	-10.12	-10.06
nltcs	-6.18	-6.10	-6.17	-6.09
plants	-13.56	-13.42	-13.56	-13.41
pumbs*	-25.66	-25.66	-25.69	-25.68
tmovie	-59.56	-59.44	-59.53	-59.35
tretail	-11.34	-11.30	-11.35	-11.27

Detailed results As an extension of Table 1, Table 2 provides the average test set log-likelihood for all adopted baselines.

Hyperparameters of RAT-SPN We took the RAT-SPN results on the twenty density estimation benchmarks from the original paper. Therefore, the hyperparameter settings for RAT-SPN are the same as reported in the original paper: cross-validate the split-depth $D \in \{1, 2, 3, 4\}$ and the number of sum-weights $W_s \in \{1e3, 1e4, 1e5\}$, and used Eq. (1) in [48] to select R, S, and I. Following the original paper, dropout is not used for training the generative models.