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.

1 Introduction

Probabilistic Circuits (PCs) (Choi et al., 2020c; Dang et al., 2021) 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) (Darwiche, 2003), sum-product networks (SPNs) (Poon & Domingos, 2011), and probabilistic sentential decision diagrams (PSDDs) (Kisa et al., 2014). 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 (Vergari et al., 2021; Khosravi et al., 2019; Shen et al., 2016). The ability to answer these queries leads to successful applications in areas such as model compression (Liang & Van den Broeck, 2017) and model bias detection (Choi et al., 2020a,b). 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 (Dang et al., 2021; Molina et al., 2019) 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 (Peharz et al., 2020a).

However, such advances make PCs more prone to overfitting. Although parameter regularization has been extensively studied in both the PGM and NN communities (Srivastava et al., 2014; Ioffe & Szegedy, 2015), 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 (Liang et al. 2017; Dang et al. 2020; Gens & Pedro 2013), 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 (Chan & Darwiche 2005; Pan et al. 2006). While learning with such softened datasets is infeasible even for simple machine learning models, with their tractability, PCs can learn the maximum-likelihood estimation (MLE) parameters given a softened dataset in $O(|p| \cdot |D|)$ time, where $|p|$ is the size of the PC and $|D|$ is the size of the (original) dataset. 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., $X$) and bold lowercase letters (e.g., $x$) 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 (Srivastava et al. 2014) and Bayesian Neural Networks (BNNs) (Goan & Fookes 2020) incorporate more complex and structured priors into the model (Gal & Ghahramani 2016). 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 (Perez & Wang 2017; Szegedy et al. 2016). A simple yet effective type of data augmentation is to inject noise into the samples, for example by randomly corrupting bits or pixels (Vincent et al. 2008). 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 $X$ 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 $\beta$ approaches 0.5, the regularized sample represents an (almost) uniform distribution. For a sample $x$ with $K$ variables $X := \{X_i\}_{i=1}^K$, where the $k$th variable takes value $x_k$, we can similarly ‘soften’ $x$ by independently injecting noise into each variable.

1We postpone the discussion on regularizing samples with non-boolean variables in Appendix B.1
resulting in a softened distribution \(P_{x,\beta} \): 

\[
\forall x' \in \text{val}(X), \quad P_{x,\beta}(X = x') := \prod_{i=1}^{K} P_{x,\beta}(X_i = x'_i) = \prod_{i=1}^{K} \left( \beta \cdot \mathbb{I}[x'_i = x_i] + (1 - \beta) \cdot \mathbb{I}[x'_i \neq x_i] \right). 
\]

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

\[
D_\beta := \{x \mid x \in \text{val}(X)\} \quad \text{and} \quad \text{weight}(D_\beta, x) = \frac{1}{N} \sum_{i=1}^{N} P_{x^{(i)},\beta}(X = x). \quad (1)
\]

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 (Jeffrey, 1990) and virtual evidence (Pearl, 2014), 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 \(D \) 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 (Russ et al., 2020) that we will experiment with in Sec. 4.3, the perfectly overfitting learned model has entropy 177. Therefore, directly controlling the entropy of the learned distribution will help mitigate overfitting. Given a model \(P_\theta \) parametrized by \(\theta \) and a dataset \(D := \{x^{(i)}\}_{i=1}^{N} \), we define the following entropy regularization objective:

\[
\text{LL}_\text{ent}(\theta; D, \tau) := \frac{1}{N} \sum_{i=1}^{N} \log P_\theta(x^{(i)}) + \tau \cdot \text{ENT}(P_\theta), \quad (2)
\]

where \(\text{ENT}(P_\theta) := -\sum_{x \in \text{val}(X)} P_\theta(x) \log P_\theta(x) \) denotes the entropy of distribution \(P_\theta \), and \(\tau \) 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 (Grandvalet & Bengio, 2006; Zhu et al., 2017) or non-parametric (Feng et al., 2017) 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 an 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) (Choi et al., 2020c), 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) (Kisa et al., 2014), Sum-Product Networks (SPNs) (Poon & Domingos, 2011), and Arithmetic Circuits (ACs) (Darwiche, 2003). 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 $X$ 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 children, denoted $\text{in}(n)$. Each unit $n$ encodes a probability distribution $p_n$, defined as follows:

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

where $f_n$ is a univariate input distribution (e.g., boolean, categorical or Gaussian), and $\theta_{n,c}$ represents the parameter that 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., $\sum_{c \in \text{in}(n)} \theta_{n,c} = 1$). We further assume w.l.o.g. that a PC alternates between sum and product layers (Vergari et al., 2015). 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-posteriori (MAP) inferences (e.g., PSDDs (Kis et al., 2014), selective SPNs (Peharz et al., 2014)); (ii) PCs that only permit linear-time computation of MAR queries (e.g., SPNs (Poon & Domingos, 2011)). 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) (Choi et al., 2020c). These are properties of the (variable) scope $\phi(n)$ of PC units $n$, that is, the collection of variables defined by all its descendant input nodes.

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

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

Next, determinism is required to guarantee efficient computation of MAP inference (Mei et al., 2018).

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

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 $\text{poly}(|p|)$ time (Vergari et al., 2021), they also exhibit drastically different expressive efficiency. Specifically, abundant empirical (Dang et al., 2020; Peharz et al., 2020a) and theoretical (Choi & Darwiche, 2017) 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 (Liang et al., 2017; Dang et al., 2020). Since it is also the default regularizer for classical probabilistic models such as Bayesian Networks (BNs) (Heckerman, 2008) and Hierarchical Bayesian Models (HBM) (Allenby & Rossi, 2006), this naturally raises the following question: *are there differences between a good regularizer for classical probabilistic models such as BNs and HBM and effective regularizers for PCs?* The question can be answered affirmatively — while Laplace smoothing provides good priors to BNs and HBM, 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.
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 [Dang et al., 2020] are imbalanced.

(i.e., \( \{\theta_{n,c} \mid c \in \text{in}(n)\} \)), 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 \( \theta_{11} \) will be much larger than \( \theta_{12} \).

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 [Kisa et al., 2014]. We examine 20 PCs learned from the 20 density estimation benchmarks [Van Haaren & Davis, 2012], respectively. All sum units with \( \geq 3 \) children and with a support size \( \geq 128 \) are recorded. We measure “imbalance” of a sum unit \( n \) by the fraction of the maximum and minimum support size of its children (i.e., \( \frac{\max_{c \in \text{in}(n)} |\text{supp}(c)|}{\min_{c \in \text{in}(n)} |\text{supp}(c)|} \)). As demonstrated in Fig. 1(b), more than 20% of the sum units have imbalance \( \geq 10^2 \), 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 \( X \) and a dataset \( D = \{x^{(i)}\}_{i=1}^N \), the maximum likelihood estimation (MLE) parameters \( \theta^*_D := \arg\max_{\theta} \sum_{i=1}^N \log p(x^{(i)}, \theta) \) can be learned in closed-form. To formalize the MLE solution, we need a few extra definitions.
Definition 5 (Context). The context \( \gamma_n \) 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: \( \gamma_{n_r} := \text{supp}(n_r) \). For every other node \( n \), its context is the intersection of its support and the union of its parents’ (\( \text{pa}(n) \)) contexts:

\[
\gamma_n := \bigcup_{m \in \text{pa}(n)} \gamma_m \cap \text{supp}(n).
\]

Intuitively, if an assignment \( x \) 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 \( x \in \text{supp}(m) \). Circuit flow extends the notation of context to indicate whether a sample \( x \) is in the context of an edge \((n,c)\).

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

The flow \( F_{n,c}(x) \) for all edges \((n,c)\) in a PC \( p \) w.r.t. sample \( \mathbf{x} \) 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 \( \forall n, \text{value} [n] := 1[\mathbf{x} \in \text{supp}(n)] \); afterwards, the backward path illustrated in Alg. 2 begins at the root unit \( n_r \) and traverses the PC in preorder to compute \( \forall n, \text{context} [n] := 1[\mathbf{x} \in \gamma_n] \) as well as \( \forall (n,c), \text{flow} [n,c] := F_{n,c}(x) \). By Def. 6, the time complexity for computing \( F_{n,c}(\mathbf{D}) \) with respect to all edges \((n,c)\) in \( p \) is \( \mathcal{O}(|p| \cdot |\mathbf{D}|) \), where \( |\mathbf{D}| \) is the size of dataset \( \mathbf{D} \). The correctness of Alg. 1 and 2 are justified in Appendix A.6.

The MLE parameters \( \theta^*_D \) given dataset \( \mathbf{D} \) can be computed using the flows (Kisa et al. 2014):

\[
\forall (n,c), \quad \theta^*_n,c = F_{n,c}(D)/\sum_{c \in \text{in}(n)} F_{n,c}(D).
\]

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

Non-deterministic PCs As justified by Peharz et al. (Peharz et al. 2016), every non-deterministic PC can be augmented as a deterministic PC with additional hidden variables. For example, in Fig. 2, the left PC is non-deterministic since the support of both children of \( n_1 \) (i.e., \( n_2 \) and \( n_3 \)) contains \( x_1 \). The right PC augments the left one by adding input units corresponding to hidden variable \( Z_1 \), which retains determinism by “dividing” the overlapping support \( x_1 \bar{x}_2 \) into \( x_1 \bar{x}_2 z_1 \in \text{supp}(n_2) \) and \( x_1 \bar{x}_2 z_1 \in \text{supp}(n_3) \). 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) (Darwiche 2009; Dempster et al. 1977). In fact, EM is the default parameter learning algorithm for non-deterministic PCs (Peharz et al. 2020a; Choi et al. 2020a).

Under the latent variable model view of a non-deterministic PC, its EM updates can be computed using expected flows (Choi et al. 2020a). Specifically, given observed variables \( \mathbf{X} \) and (implicit) hidden variables \( \mathbf{Z} \), the expected flow of edge \((n,c)\) given dataset \( \mathbf{D} \) is defined as

\[
\text{EF}_{n,c}(\mathbf{D}; \theta) := \mathbb{E}_{\mathbf{x} \sim \mathbf{D}, \mathbf{z} \sim p_{\cdot}(\cdot | \mathbf{x} ; \theta)}[F_{n,c}(\mathbf{x}, \mathbf{z})],
\]

where \( \theta \) is the set of parameters, and \( p_{\cdot}(\cdot | \mathbf{x}; \theta) \) is the conditional probability over hidden variables \( \mathbf{Z} \) given \( \mathbf{x} \) 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. (Choi et al. 2020a), for a non-deterministic PC, its parameters for the next EM iteration are given by

\[
\theta_n,c^{(\text{new})} = \frac{\text{EF}_{n,c}(\mathbf{D}; \theta)}{\sum_{c \in \text{in}(n)} \text{EF}_{n,c}(\mathbf{D}; \theta)}.
\]

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, \( \theta^{(\text{new})} \) are computed...
using mini-batches of samples, and the parameters are updated towards the target with a step size $\eta$: $\theta^{(k+1)} \leftarrow (1 - \eta)\theta^{(k)} + \eta\theta^{(\text{new})}$; when using full-batch EM, we iteratively compute the updated parameters $\theta^{(\text{new})}$ 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 (Kingma & Welling, 2013) and GANs (Goodfellow et al., 2014). Despite this negative result, the MLE parameters of a PC w.r.t. $\mathcal{D}_\beta$ can be exactly computed in time $O(|\mathcal{D}||\beta|)$, which is linear w.r.t. the model size as well as the size of the original dataset.

**Theorem 3.** Let $f_n(x) = \beta \cdot 1[x \in \text{supp}(n)] + (1 - \beta) \cdot 1[x \notin \text{supp}(n)]$ in Alg. 1. Given a deterministic PC $p$, a boolean dataset $\mathcal{D}$, and hyperparameter $\beta \in (0, 1]$, the set of all flows $\{F_{n,c}(\mathcal{D}_\beta) | \forall \text{edge } (n,c)\}$ w.r.t. the softened dataset $\mathcal{D}_\beta$ can be computed by Alg. 1 and 3 within $O(|\mathcal{D}||\beta|)$ 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 $O(|\mathcal{D}|)$ time using the flows, the overall time complexity to compute the MLE parameters is again $O(|\mathcal{D}||\beta|)$.

**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 $O(|p|)$ time (Vergari et al., 2021; Shih & Ermon, 2020). However, it is still unclear whether the entropy regularization objective $\text{LL}_{\text{ent}}(\theta; \mathcal{D}, \tau)$ (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 $\tau$, and a dataset $\mathcal{D}$ such that $\text{LL}_{\text{ent}}(\theta; \mathcal{D}, \tau)$ (Eq. (2)) is non-concave and has multiple local maxima.

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 $\text{LL}_{\text{ent}}(\theta; \mathcal{D}, \tau)$. 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.

We compute parameters in the logarithm space for numerical stability.
Algorithm 3 PC Entropy regularization

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

$$
\begin{align}
&d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \text{entropy}[c_i] = y \quad (\forall i \in \{1, \ldots, |\text{in}(n)|\}) \\
&\sum_{i=1}^{|\text{in}(n)|} e^{\varphi_{n,c_i}} = 1
\end{align}
$$

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

**Theorem 4.** Alg. 3 converges monotonically to a stationary point of $\text{LL}_{\text{ent}}(\theta; D, \tau)$ 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 $\text{LL}_{\text{ent}}(\theta; D, \tau)$, which is determined by the entropies computed in line 5. Specifically, we show that whenever the surrogate objective is improved, $\text{LL}_{\text{ent}}(\theta; D, \tau)$ is also improved. Since the surrogate objective is concave, it can be easily optimized. Therefore, Alg. 3 converges to a stationary point of $\text{LL}_{\text{ent}}(\theta; D, \tau)$. 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 ($\text{LL}_{\text{ent}}(\theta; D, \tau)$), 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. $D$, which takes $O(|p| \cdot |D|)$ time, Alg. 3 takes a negligible $O(|p|)$ 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-regularize $\theta_{11}$ compared to $\theta_{12}$. 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 $\theta_{12}$. Suppose $n=10$, consider maximizing Eq. (2) with an empty dataset (i.e., we maximize $\text{ENT}(p_{m_1})$ 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.

---

3 This can be done by Alg. 4 shown in Appendix A.2 Lem. 1 proves that Alg. 4 takes $O(|p|)$ time.
4 Details for solving Eq. (5) is given in Appendix B.2.
Empirical evaluation

We empirically evaluate both proposed regularization methods on the twenty density estimation datasets (Van Haaren & Davis, 2012). Since we are only concerned with parameter learning, we adopt PC structures (defined by its DAG) learned by Strudel (Dang et al., 2020). 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) 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 $LL_{\text{train}}$ and $LL_{\text{val}}$ 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 $(LL_{\text{val}} - LL_{\text{train}})/LL_{\text{val}}$, 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, 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 (Vergari et al., 2021), it is infeasible to optimize the entropy regularization objective $LL_{\text{ent}}(\theta; D, \tau)$ (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) (Chow & Liu, 1968) 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

\[\alpha \in \{0.1, 0.4, 1.0, 2.0, 4.0, 10.0\}, \beta \in \{0.9996, 0.999, 0.996\}, \tau \in \{0.001, 0.01, 0.1\}.\]
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 (Dang et al., 2020) and LearnPSDD (Liang et al., 2017) as well as the best performance (Best SPN) over 4 SPN learning algorithms: EinSumNet (Peharz et al., 2020a), LearnSPN (Gens & Pedro, 2013), ID-SPN (Rooshenas & Lowd, 2014), and RAT-SPN (Peharz et al., 2020b). With the help of data softening and entropy regularization ($\alpha = 0.1$, $\beta = 0.002$, 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    |
| baidio   | -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   | pubmed* | -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   |

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.
References

Allenby, G. M. and Rossi, P. E. Hierarchical bayes models. *The handbook of marketing research: Uses, misuses, and future advances*, pp. 418–440, 2006.

Chan, H. and Darwiche, A. On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, 163(1):67–90, 2005.

Choi, A. and Darwiche, A. On relaxing determinism in arithmetic circuits. In *International Conference on Machine Learning*, pp. 825–833. PMLR, 2017.

Choi, Y., Dang, M., and Broeck, G. V. d. Group fairness by probabilistic modeling with latent fair decisions. *arXiv preprint arXiv:2009.09031*, 2020a.

Choi, Y., Farnadi, G., Babaki, B., and Van den Broeck, G. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10077–10084, 2020b.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. *preprint*, 2020c.

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

Dang, M., Vergari, A., and Broeck, G. V. d. Strudel: Learning structured-decomposable probabilistic circuits. *arXiv preprint arXiv:2007.09331*, 2020.

Dang, M., Khosravi, P., Liang, Y., Vergari, A., and Van den Broeck, G. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021.

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

Darwiche, A. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Feng, Y., Wang, D., and Liu, Q. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Gens, R. and Pedro, D. Learning the structure of sum-product networks. In *International conference on machine learning*, pp. 873–880. PMLR, 2013.

Goan, E. and Fookes, C. Bayesian neural networks: An introduction and survey. In *Case Studies in Applied Bayesian Data Science*, pp. 45–87. Springer, 2020.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *NIPS*, 2014.

Grandvalet, Y. and Bengio, Y. Entropy regularization., 2006.

Heckerman, D. A tutorial on learning with bayesian networks. *Innovations in Bayesian networks*, pp. 33–82, 2008.
Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pp. 448–456. PMLR, 2015.

Jeffrey, R. C. The logic of decision. University of Chicago press, 1990.

Khosravi, P., Choi, Y., Liang, Y., Vergari, A., and Van den Broeck, G. On tractable computation of expected predictions. In Advances in Neural Information Processing Systems 32 (NeurIPS), dec 2019.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In Proceedings of the 14th international conference on principles of knowledge representation and reasoning (KR), pp. 1–10, 2014.

Liang, Y. and Van den Broeck, G. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI), August 2017. URL http://starai.cs.ucla.edu/papers/LiangXAI17.pdf

Liang, Y., Bekker, J., and Van den Broeck, G. Learning the structure of probabilistic sentential decision diagrams. In Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI), 2017.

Mei, J., Jiang, Y., and Tu, K. Maximum a posteriori inference in sum-product networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Di Mauro, N., Poupart, P., and Kersting, K. Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. arXiv preprint arXiv:1901.03704, 2019.

Pan, R., Peng, Y., and Ding, Z. Belief update in bayesian networks using uncertain evidence. In 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’06), pp. 441–444. IEEE, 2006.

Pearl, J. Probabilistic reasoning in intelligent systems: networks of plausible inference. Elsevier, 2014.

Peharz, R., Gens, R., and Domingos, P. Learning selective sum-product networks. In LTPM workshop, volume 32, 2014.

Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. On the latent variable interpretation in sum-product networks. IEEE transactions on pattern analysis and machine intelligence, 39(10):2030–2044, 2016.

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

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

Perez, L. and Wang, J. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621, 2017.

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

Rooshenas, A. and Lowd, D. Learning sum-product networks with direct and indirect variable interactions. In International Conference on Machine Learning, pp. 710–718. PMLR, 2014.

Russ, W. P., Figliuzzi, M., Stocker, C., Barrat-Charlaix, P., Socolich, M., Kast, P., Hilvert, D., Monasson, R., Cocco, S., Weigt, M., et al. An evolution-based model for designing chorismate mutase enzymes. Science, 369(6502):440–445, 2020.
Shen, Y., Choi, A., and Darwiche, A. Tractable operations for arithmetic circuits of probabilistic models. In Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 3943–3951. Citeseer, 2016.

Shih, A. and Ermon, S. Probabilistic circuits for variational inference in discrete graphical models. In Advances in Neural Information Processing Systems 33 (NeurIPS), december 2020. URL https://cs.stanford.edu/~andyshih/assets/pdf/SEneurips20.pdf.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826, 2016.

Van den Broeck, G., Lykov, A., Schleich, M., and Suciu, D. On the tractability of SHAP explanations. In Proceedings of the 35th AAAI Conference on Artificial Intelligence, Feb 2021. URL http://starai.cs.ucla.edu/papers/VdBAAAI21.pdf.

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

Vergari, A., Di Mauro, N., and Esposito, F. Simplifying, regularizing and strengthening sum-product network structure learning. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 343–358. Springer, 2015.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. A compositional atlas of tractable circuit operations: From simple transformations to complex information-theoretic queries. arXiv preprint arXiv:2102.06137, 2021.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pp. 1096–1103, 2008.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision, pp. 2223–2232, 2017.
Supplementary Material

A Proofs

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

A.1 Proof of Theorem 3

We break down the proof into two parts — correctness of the forward pass [Alg. 1] and correctness of the backward pass [Alg. 2]. As stated in the theorem, assume that we are given a deterministic PC \( p \), a boolean dataset \( D \) containing \( N \) samples \( \{x^{(i)}\}_{i=1}^{N} \), and hyperparameter \( \beta \in (0.5, 1] \). Define \( K \) as the number of variables in \( X \), i.e., \( X = \{X_k\}_{k=1}^{K} \).

Correctness of the forward pass

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

\[
\text{value}_i[n] = \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot \mathbb{1}[x_k^{(i)} = x_k] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k] \right) \cdot \mathbb{1}[x \in \text{supp}(n)],
\]

where \( x_k \) denotes the \( k \)-th feature of \( x \).

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

\[
\text{value}_i[n] = \beta \cdot \mathbb{1}[x_k^{(i)} \in \text{supp}(n)] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} \notin \text{supp}(n)] = \beta \cdot \mathbb{1}[x_k^{(i)} = \text{Lit}(n)] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} = \neg \text{Lit}(n)]
\]

\[
= \sum_{x \in \{x \in \text{val}(X) \mid x_k = \text{Lit}(n)\}} \prod_{l=1,l \neq k}^{K} \left( \beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1 - \beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l] \right) 
\cdot \left( \beta \cdot \mathbb{1}[x_k^{(i)} = \text{Lit}(n)] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} = \neg \text{Lit}(n)] \right)
\]

\[
= \sum_{x \in \{x \in \text{val}(X) \mid x_k = \text{Lit}(n)\}} \prod_{l=1,l \neq k}^{K} \left( \beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1 - \beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l] \right)
\cdot \left( \beta \cdot \mathbb{1}[x_k^{(i)} = x_k] \cdot \mathbb{1}[x_k = \text{Lit}(n)] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k] \cdot \mathbb{1}[x_k = \neg \text{Lit}(n)] \right)
\]

\[
= \sum_{x \in \{x \in \text{val}(X) \mid x_k = \text{Lit}(n)\}} \prod_{l=1,l \neq k}^{K} \left( \beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1 - \beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l] \right)
\cdot \left( \beta \cdot \mathbb{1}[x_k^{(i)} = x_k] \cdot \mathbb{1}[x_k = \text{Lit}(n)] \right)
\]

\[
\overset{(b)}{=} \sum_{x \in \{x \in \text{val}(X)\}} \prod_{l=1}^{K} \left( \beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1 - \beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l] \right) \cdot \mathbb{1}[x \in \text{supp}(n)],
\]

where (a) holds because the added term

\[
\sum_{x \in \{x \in \text{val}(X) \mid x_k = \text{Lit}(n)\}} \prod_{l=1,l \neq k}^{K} \left( \beta \cdot \mathbb{1}[x_l^{(i)} = x_l] + (1 - \beta) \cdot \mathbb{1}[x_l^{(i)} \neq x_l] \right) = 1;
\]
the sum condition \( x_k = \text{Lit}(n) \) after (b) can be lifted thanks to the indicator \( 1[x_k = \text{Lit}(n)] \).

- Inductive case: product units. Suppose \( n \) is a product unit with children \( \{c_j \}_{j=1}^{\infty} \). 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.,

\[
\text{value}_i[c_j] = \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) \cdot 1[x \in \text{supp}(c_j)].
\]

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

\[
\text{value}_i[n] = \prod_{j=1}^{\infty} \text{value}_i[c_j]
\]

\[
= \prod_{j=1}^{\infty} \left\{ \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) \cdot 1[x \in \text{supp}(c_j)] \right\}
\]

\[
= \prod_{j=1}^{\infty} \left\{ \sum_{x \in \text{val}(\bigcup_{j=1}^{\infty} \phi(c_j))} \prod_{k \in K_{c_j}} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) \cdot 1[x \in \text{supp}(c_j)] \right\}
\]

\[
= \sum_{x \in \text{val}(\bigcup_{j=1}^{\infty} \phi(c_j))} \prod_{k \in K_{c_j}} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) 1[x \in \text{supp}(n)],
\]

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

- Inductive case: sum units. Suppose \( n \) is a sum unit with children \( \{c_j \}_{j=1}^{\infty} \). Suppose the value \( \text{value}_i[c_j] \) of any child unit \( c_j \) is defined according to Eq. (6) we have

\[
\text{value}_i[n] = \sum_{j=1}^{\infty} \text{value}_i[c_j]
\]

\[
= \sum_{j=1}^{\infty} \left\{ \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) \cdot 1[x \in \text{supp}(c_j)] \right\}
\]

\[
= \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) \cdot \left( \sum_{j=1}^{\infty} 1[x \in \text{supp}(c_j)] \right)
\]

\[
= \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot 1[x_k^{(i)} = x_k] + (1 - \beta) \cdot 1[x_k^{(i)} \neq x_k] \right) 1[x \in \text{supp}(n)],
\]

15
According to Def. 5, the context of the root node

\[ \text{context}_n = \bigcup_{c \in \text{pa}(n)} \text{context}_{c} \]

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

We have shown that for any unit \(n\), the value stored in \(\text{value}_{n}\) follows line 8 of Alg. 1; \(\text{value}_{n}\) 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 \(\text{context}_n\) of each sum unit w.r.t. sample \(x^{(i)}\) computed by Alg. 2 is defined as

\[
\text{context}_n = \bigcup_{c \in \text{pa}(n)} \text{context}_{c} \bigcup \text{context}_n \supseteq \text{context}_n
\]

We proceed to show that \(\text{context}_n\) is defined as

\[
\text{context}_n = \bigcup_{c \in \text{pa}(n)} \text{context}_{c} \bigcup \text{context}_n \supseteq \text{context}_n.
\]

We have shown that for any unit \(n\), the value stored in \(\text{value}_{n}\) follows line 8 of Alg. 1; \(\text{value}_{n}\) 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 \(\text{context}_n\) of each sum unit w.r.t. sample \(x^{(i)}\) computed by Alg. 2 is defined as

\[
\text{context}_n = \bigcup_{c \in \text{pa}(n)} \text{context}_{c} \bigcup \text{context}_n \supseteq \text{context}_n.
\]

and the flow \(\text{flow}_{n,c}\) of each edge \((n,c)\) s.t. \(n\) is a sum unit is:

\[
\text{flow}_{n,c} = \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} \left( \beta \cdot \mathbb{1}[x_k^{(i)} = x_k] + (1 - \beta) \cdot \mathbb{1}[x_k^{(i)} \neq x_k] \right) \cdot \mathbb{1}[x \in \gamma_n].
\]

**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., \(\gamma_{n_r} = \text{supp}(n_r)\). Since in line 3 of Alg. 2 the value \(\text{context}_{n_r}\) is set to \(\text{value}_{n_r}\), we know that

\[
\text{context}_{n_r} = \bigcup_{c \in \text{pa}(n_r)} \text{context}_{c} \bigcup \text{context}_{n_r} \supseteq \text{context}_{n_r}.
\]

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

\[
\text{flow}_{g_j, m_j} = \frac{\text{value}_{m_j}}{\text{value}_{g_j}} \cdot \text{context}_{g_j}
\]

\[
= \frac{\sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \mathbb{1}[x \in \gamma_{g_j}]}{\sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \mathbb{1}[x \in \text{supp}(g_j)]}
\]

\[
\cdot \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \mathbb{1}[x \in \text{supp}(m_j)]
\]

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

\[
\mathbb{1}[x \in \gamma_{g_j}] = \mathbb{1}[x \in \gamma_{g_j} \cap \text{supp}(g_j)].
\]

Consider conditioning \(\text{supp}(g_j)^c\) and \(\gamma_{g_j}^c\) on the variables \(\phi(g_j)\) (i.e., the variable scope of \(g_j\)). For any partial variable assignment \(e\) over \(\phi(g_j)\), if \(e \in \text{supp}(g_j)\), then \(e \in \gamma_{g_j}^c\). Denote \(K_{g_j}\) as the set of index for the variables in \(\phi(g_j)\). We have

\[
\sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \mathbb{1}[x \in \gamma_{g_j}^c] \cdot \mathbb{1}[x \in \text{supp}(g_j)]
\]

\(^6\)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.
which is equivalent to Eq. (7).

Note that \( \text{context}_i[n] \) is computed as

\[
\text{context}_i[n] = \sum_{j=1}^{\lvert \text{pa}(n) \rvert} \text{flow}_i[g_j, m_j]
\]
Alg. 2:

\[
\begin{align*}
&= \sum_{j=1}^{\text{\#pa}(n)} \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot I[x \in \gamma_{gj} \cap \gamma_{mj}]
\end{align*}
\]

\[
= \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \left( \sum_{j=1}^{\text{\#pa}(n)} I[x \in \gamma_{gj} \cap \gamma_{mj}] \right). \tag{14}
\]

Next, we show that \( \forall m_i, m_j \in \text{pa}(n)(m_i \neq m_j), \gamma_{m_i} \cap \gamma_{m_j} = \emptyset \). We prove this claim using its contrapositive form. Suppose there exists \( x \in \text{val}(X) \) such that \( x \in \gamma_{m_i} \) and \( x \in \gamma_{m_j} \). According to the definition of context, if \( x \in \gamma_{m_i} \), 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, \( x \in \gamma_c \). Similarly, there must exist 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., \( \exists c_1, c_2 \in \text{in}(n')(c_1 \neq c_2) \), such that \( x \in \gamma_{c_1} \) and \( x \in \gamma_{c_2} \). This contradicts the assumption that the PC is deterministic. Therefore, the claim at the beginning of this paragraph holds. Thus,

\[
\sum_{j=1}^{\text{\#pa}(n)} I[x \in \gamma_{gj} \cap \gamma_{mj}] = \sum_{j=1}^{\text{\#pa}(n)} I[x \in \gamma_{mj}] = I[x \in \bigcup_{j=1}^{\text{\#pa}(n)} \gamma_{mj}] = I[x \in \bigcup_{j=1}^{\text{\#pa}(n)} \gamma_{mj} \cap \text{supp}(n)] = I[x \in \gamma_n],
\]

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

Plug in Eq. (14) we have

\[
\text{context}_r[n] = \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot \left( \sum_{j=1}^{\text{\#pa}(n)} I[x \in \gamma_{gj} \cap \gamma_{mj}] \right)
\]

\[
= \sum_{x \in \text{val}(X)} \prod_{k=1}^{K} H(x, x^{(i)}, k) \cdot I[x \in \gamma_n].
\]

Computing \( F_{n,c}(D_\beta) \) Finally, we can compute \( F_{n,c}(D_\beta) \) from the flows (i.e., \( \text{flow}_{[n,c]} \)) computed by Alg. 2

\[
F_{n,c}(D_\beta) = \sum_{x \in \text{val}(X)} \text{weight}(D_\beta, x) \cdot I[x \in \gamma_n \land x \in \gamma_c]
\]

\[
= \sum_{x \in \text{val}(X)} \sum_{i=1}^{N} \prod_{k=1}^{K} \left( \beta \cdot I[x_k^{(i)} = x_k] + (1-\beta) \cdot I[x_k^{(i)} \neq x_k] \right) \cdot I[x \in \gamma_n \land x \in \gamma_c]
\]

\[
= \sum_{x \in \text{val}(X)} \sum_{i=1}^{N} \prod_{k=1}^{K} \left( \beta \cdot I[x_k^{(i)} = x_k] + (1-\beta) \cdot I[x_k^{(i)} \neq x_k] \right) \cdot I[x \in \gamma_n \land x \in \gamma_c]
\]

18
Algorithm 4 PC entropy

1: Input: A deterministic PC \( p \)
2: Output: \( \text{entropy}[n] := \text{ENT}(p_n) \) for every unit \( n \)
3: foreach \( n \) traversed in postorder do
4:  if \( n \) is an input unit then \( \text{entropy}[n] = \text{ENT}(p_n) \) /entropy of the input distribution
5:  elseif \( n \) is a product unit then \( \text{entropy}[n] = \sum_{c \in \text{in}(n)} \text{entropy}[c] \)
6:  else //\( n \) is a sum unit then
7:     \[
    \text{entropy}[n] = - \sum_{c \in \text{in}(n)} \theta_{n,c} \log \theta_{n,c} + \sum_{c \in \text{in}(n)} \theta_{n,c} \cdot \text{entropy}[c]
    \]
8: = \( \sum_{i=1}^{N} \text{flow}_i[n,c] \).

Finally, we note that [Alg. 1] and [Alg. 2] both run in time \( O(|p| \cdot |D|) \).

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 \( \text{ENT}(p) \) can be decomposed recursively as follows:

\[
\text{ENT}(p) = \begin{cases}
\sum_{c \in \text{in}(n)} \left( -\theta_{n,c} \log \theta_{n,c} + \theta_{n,c} \cdot \text{ENT}(p_c) \right) & \text{if } n \text{ is a sum unit,} \\
\sum_{c \in \text{in}(n)} \text{ENT}(p_c) & \text{if } n \text{ is a product unit,}
\end{cases}
\]

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 \( O(|p|) \) 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:

\[
\text{ENT}(p) = - \sum_{x \in \text{val}(\phi(n))} \left( \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) \right) \log \left( \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) \right)
\]

\[
= - \sum_{x \in \text{val}(\phi(n))} \left( \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) \mathbb{I}[x \in \text{supp}(c)] \right) \log \left( \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) \mathbb{I}[x \in \text{supp}(c)] \right)
\]

\[
= \sum_{c \in \text{in}(n)} \theta_{n,c} \log \theta_{n,c} \left( \sum_{x \in \text{val}(\phi(n))} \mathbb{I}[x \in \text{supp}(c)] p_c(x) \right)
\]

\[
= \sum_{c \in \text{in}(n)} \theta_{n,c} \log \theta_{n,c} \left( \sum_{x \in \text{val}(\phi(n))} \mathbb{I}[x \in \text{supp}(c)] p_c(x) \right)
\]

\[
= \sum_{c \in \text{in}(n)} \theta_{n,c} \log \theta_{n,c} + \sum_{c \in \text{in}(n)} \theta_{n,c} \cdot \text{ENT}(p_c)
\]

(15)

- **Product units.** If \( n \) is a product unit:

\[
\text{ENT}(p) = \sum_{c \in \text{in}(n)} \text{ENT}(p_c)
\]
Figure 7: An example PC to show that PC entropy is neither convex nor concave.

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

- Product units. If \( n \) is a product unit:
  \[
  \text{ENT}(p_n) = - \sum_{x \in \text{val}(\phi(n))} \left( \prod_{c \in \text{in}(n)} p_c(x) \right) \log \left( \prod_{c \in \text{in}(n)} p_c(x) \right)
  = - \sum_{c \in \text{in}(n)} \left( \sum_{x \in \text{val}(\phi(c))} p_c(x) \log p_c(x) \right)
  = \sum_{c \in \text{in}(n)} \text{ENT}(p_c) . \tag{16}
  \]

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 \( \theta_a = \{\theta_1 = 0.1, \theta_2 = 0.1\} \) and \( \theta_b = \{\theta_1 = 0.12, \theta_2 = 0.12\} \). Denote \( \theta_c = (\theta_a + \theta_b)/2 \), we have

\[
2 \cdot \text{ENT}(p; \theta_c) - \text{ENT}(p; \theta_a) - \text{ENT}(p; \theta_b) \approx -0.0047898 < 0.
\]

Hence the entropy is not concave.

Define parameters \( \theta_d = \{\theta_1 = 0.4, \theta_2 = 0.8\} \) and \( \theta_e = \{\theta_1 = 0.42, \theta_2 = 0.82\} \). Denote \( \theta_f = (\theta_d + \theta_e)/2 \), we have

\[
2 \cdot \text{ENT}(p; \theta_f) - \text{ENT}(p; \theta_d) - \text{ENT}(p; \theta_e) \approx 0.0056294 > 0.
\]

Hence the entropy is not convex.

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

\[
\sum_{i=1}^N \log p(x^{(i)}; \theta) . \tag{17}
\]
Proof. For any input \( x \), \( \log p(x; \theta) \) can be decomposed over sum and product units:

- **Sum units.** Suppose \( n \) is a sum unit, then
  \[
  \log p_n(x; \theta) = \log \left( \sum_{c \in \text{in}(n)} \theta_n,c \cdot p_c(x) \right) 
  = \log \left( \sum_{c \in \text{in}(n)} \theta_n,c \cdot p_c(x) \mathbb{1}[x \in \text{supp}(c)] \right) 
  = \sum_{c \in \text{in}(n)} \mathbb{1}[x \in \text{supp}(c)] \left( \log \theta_n,c + \log p_c(x) \right),
  \]
  where the last equation holds because unit \( n \) is deterministic: \( \forall c_i, c_j \in \text{in}(n)(c_i \neq c_j), \text{supp}(c_i) \cap \text{supp}(c_j) = \emptyset \).

- **Product units.** Suppose \( n \) is a product unit, then
  \[
  \log p_n(x; \theta) = \log \left( \prod_{c \in \text{in}(n)} p_c(x) \right) = \sum_{c \in \text{in}(n)} \log p_c(x).
  \]

According to Eqs. (18) and (19) for any \( x \in \text{val}(X) \), \( \log p(x; \theta) \) 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 \( \text{ENT}(p) \) can be decomposed as follows:
\[
\text{ENT}(p_{n_r}) = - \sum_{(n,c) \in \text{edges}(p_{n_r})} P_{n_r}(n) \cdot \theta_{n,c} \log \theta_{n,c},
\]
where \( \text{edges}(p) \) 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 descendants are either input units or product unit. By definition, we have \( P_m(m) = 1 \), and \( \text{edges}(p_m) = \{(m,c) \mid c \in \text{in}(m)\} \). Thus,
  \[
  - \sum_{(n,c) \in \text{edges}(p_m)} P_m(n) \cdot \theta_{n,c} \log \theta_{n,c} = - \sum_{c \in \text{in}(m)} \theta_{m,c} \log \theta_{m,c} = \text{ENT}(p_m).
  \]

- **Inductive case: product units.** Suppose \( m \) is a product unit such that for each of its children \( c \in \text{in}(m) \), we have
  \[
  \text{ENT}(p_c) = - \sum_{(n',c') \in \text{edges}(p_c)} P_c(n') \cdot \theta_{n',c'} \log \theta_{n',c'}.
  \]

Then by Lem. 1 we know that
\[
\text{ENT}(p_m) = \sum_{c \in \text{in}(m)} \text{ENT}(p_m) 
= - \sum_{c \in \text{in}(m)} \sum_{(n',c') \in \text{edges}(p_c)} P_c(n') \cdot \theta_{n',c'} \log \theta_{n',c'} 
\overset{(a)}{=} - \sum_{c \in \text{in}(m)} \sum_{(n',c') \in \text{edges}(p_c)} P_m(n') \cdot \theta_{n',c'} \log \theta_{n',c'} 
\overset{(b)}{=} - \sum_{(n',c') \in \text{edges}(p_m)} P_m(n') \cdot \theta_{n',c'} \log \theta_{n',c'},
\]
where \((a)\) holds since for any sum unit \( n' \), \( P_c(n') = P_m(n') \), and \((b)\) follows from the fact that \( \text{edges}(p_m) = \bigcup_{c \in \text{in}(m)} \text{edges}(p_c) \).
The objective in Eq. (2) can be re-written as follows

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

Then by Lemma 1 we have

$$\text{ENT}(p_m) = \sum_{c \in \text{in}(m)} \left( -\theta_{n, c} \log \theta_{n, c} + \theta_{n, c} \cdot \text{ENT}(p_c) \right)$$

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

$$= \sum_{c \in \text{in}(m)} -P_m(m) \theta_{n, c} \log \theta_{n, c} - \sum_{c \in \text{in}(m)} \sum_{(n', c') \in \text{edges}(p_c)} P_m(n') \cdot \theta_{n', c'} \log \theta_{n', c'}$$

$$(a) = - \sum_{(n', c') \in \text{edges}(p_m)} P_m(n') \cdot \theta_{n', c'} \log \theta_{n', c'},$$

where $(a)$ holds because $\text{edges}(p_m) = (\bigcup_{c \in \text{in}(m)} \text{edges}(p_c)) \cup \{(m, c) \mid c \in \text{in}(m)\}$. □

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

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

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

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

$$\mathcal{L}_{\text{ent}}(\theta; n_1, \tau) = \log \theta_1 + 67 \cdot \log \theta_2 + \tau \cdot \text{ENT}(p_{n_1})$$

$$= \log \theta_1 + 67 \cdot \log \theta_2 + \tau \cdot (\theta_1 \log \theta_1 - (1-\theta_1) \log(1-\theta_1))$$

$$+ \tau \cdot \theta_1 \cdot \text{ENT}(p_{n_2}) + \tau \cdot (1-\theta_1) \cdot \text{ENT}(p_{n_3}).$$

First, we observe that to maximize $\mathcal{L}_{\text{ent}}(\theta; n_1, \tau)$, $\theta_3$ should always be 0.5 since the only term that depends on $\theta_3$ is $(1-\theta_1) \cdot \text{ENT}(p_{n_3})$ and $1-\theta_1 > 0$. Therefore, we have

$$\text{ENT}(p_{n_3}) = 78 \cdot \log 2 \approx 54.065.$$
Next, for any fixed $\theta_1 \in (0, 1]$, the objective $L_{\text{ent}}(\theta; n_1, \tau)$ is concave w.r.t. $\theta_2$:

$$L_{\text{ent}}(\theta; n_1, \tau) = 67 \cdot \left( \log \theta_2 - \tau \cdot \theta_1 \cdot (\theta_2 \log \theta_2 + (1 - \theta_2) \log(1 - \theta_2)) \right) + \text{const},$$

where the constant term does not depend on $\theta_2$. Therefore, for any $\theta_1$, we can uniquely compute the optimal value of $\theta_2$. We are left with determining the optimal value of $\theta_1$. Choose $\tau = 1.5$, the derivative of $L_{\text{ent}}(\theta; n_1, \tau)$ w.r.t. $\theta_1$ is (denote $\text{ent}_0 := -(\theta_2 \log \theta_2 + (1 - \theta_2) \log(1 - \theta_2))$)

$$g(\theta_1) := \frac{\partial L_{\text{ent}}(\theta; n_1, \tau)}{\partial \theta_1} = \frac{1}{\theta_1} + 1.5 \cdot \left( \log(1 - \theta_1) - \theta_1 + \text{ENT}(p_{n_2}) - \text{ENT}(p_{n_3}) \right)$$

$$= \frac{1}{\theta_1} + 1.5 \cdot \left( \log(1 - \theta_1) - \theta_1 + 67 \cdot \text{ent}_0 - \text{ENT}(p_{n_3}) \right)$$

where $\text{ENT}(p_{n_3})$ can be viewed as a constant and $\text{ent}_0$ depends on $\theta_1$. Specifically, for any $\theta_1$, we compute $\theta_2$ and hence $\text{ent}_0$ by maximizing Eq. (20). Putting everything together, we have

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

Since $g$ is continuous in range $(0, 1]$, there exists a local maxima of $\theta_1$ between 0.02 and 0.7 as well as between 0.9 and 0.99. Therefore, the entropy regularization objective could have multiple local maxima.

A.3 Proof of Theorem 1

This theorem is a direct corollary of Theorem 5 in [Van den Broeck et al. 2021], 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 $D_\beta$ is essentially uniform, Thm. 1 follows directly from [Van den Broeck et al. 2021].

A.4 Proof of Theorem 2

This proof largely follows the proof of Theorem 5 in [Van den Broeck et al. 2021]. The proof is by reduction from $\#\text{NUMPAR}$, 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 $\sum_{i \in S} k_i = \sum_{i \in \bar{S}} k_i$. $\#\text{NUMPAR}$ is known to be $\#$-P-hard.

Fix an instance of $\#\text{NUMPAR}$, $k_1, \ldots, k_n$, 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 $\#\text{NUMPAR}$ 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 $\sigma$ 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 $Z := \sum_{x \in \text{val}(X)} F(x_1, \ldots, x_n)$. Denote the entropy of a normalized logistic regressor $G$ as $\text{ENT}(G) := -\sum_{x \in \text{val}(X)} G(x) \log G(x)$.

We now describe an algorithm that computes $|P|$ using an oracle for $\text{ENT}(G)$, 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 $\text{ENT}(G)$ as follows:

$$\text{ENT}(G) = - \sum_{x \in \text{val}(X)} \frac{F(x)}{Z} \log \frac{F(x)}{Z} = - \sum_{x \in \text{val}(X)} \left( \frac{F(x) \log F(x)}{Z} - \frac{F(x)}{Z} \log Z \right)$$

$$= - \sum_{x \in \text{val}(X)} \frac{F(x) \log F(x)}{Z} + \log Z.$$

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

$$\text{ENT}(G) \approx \log Z = \log \left( \sum_{x \in \text{val}(X)} \sigma(w_0 + \sum_{i=1}^{n} w_i \cdot x_i) \right) = \log \left( \sum_{x \in \text{val}(X)} \sigma(w_0 + \sum_{i=1}^{n} w_i \cdot x_i) \right).$$

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

$$\exp(\text{ENT}(G)) \approx \sum_{x \in \text{val}(X)} \sigma\left(-\frac{m}{2} - mc + m(\sum_{i \in [n]} k_i x_i)\right)$$

$$= \sum_{x \in \text{val}(X)} \sigma\left(-\frac{m}{2} - mc + m(\sum_{i : x_i = 1} k_i)\right)$$

$$= \sum_{S \subseteq [n]} \sigma(\text{weight}(S))$$

$$= \frac{1}{2} \sum_{S \subseteq [n]} (\sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S}))).$$

If $S$ is a solution to $\#\text{NUMPAR}$, then

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

Otherwise, 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

$$S \in P : \ 0 \leq \sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \leq \epsilon,$$

$$S \notin P : \ 1 - \epsilon \leq \sigma(\text{weight}(S)) + \sigma(\text{weight}(\bar{S})) \leq 1 + \epsilon.$$

Therefore, we have

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

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

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

This gives a lower and upper bound for $|P|$. For small enough $\epsilon$ (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.

24
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.

**Proof.**

Before verifying the statements, we first formally define the surrogate. Denote the set of reference parameters $\theta^{\text{ref}}$ as the parameters in Alg. 3. Given a set of reference parameters $\theta^{\text{ref}}$, we define the surrogate objective w.r.t. parameter $\theta_{n,c}$ as

$$L_{\text{surr}}(\theta_{n,c}; \theta^{\text{ref}}) := \frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)}; \theta^{\text{ref}} \setminus \{\theta^{\text{ref}}_{n,c}\}, \theta_{n,c})$$

$$+ \tau \cdot P_{n}(n; \theta^{\text{ref}}) \cdot \left( -\theta_{n,c} \log \theta_{n,c} + \theta_{n,c} \cdot \text{ENT}(p_{c}; \theta^{\text{ref}}) \right).$$

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

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

$$\sum_{c \in \text{in}(n)} L_{\text{surr}}(\theta_{n,c}; \theta^{\text{update}}).$$

After solving the above equation, the updated parameters $\{\theta_{n,c} \mid c \in \text{in}(n)\}$ replace the corresponding original parameters in $\theta^{\text{update}}$. 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 $\{\theta_{n,c} \mid c \in \text{in}(n)\}$.

**Proof.** This statement can be proved by showing that $\forall (n,c), \forall \theta$, $L_{\text{surr}}(\theta_{n,c}; \theta)$ 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) $P_{n}(n; \theta^{\text{ref}})$ and ENT$(p_{c}; \theta^{\text{ref}})$ are independent of $\{\theta_{n,c} \mid c \in \text{in}(n)\}$. □

- **Statement #2**: For any sum unit $n$ and any parameters $\theta$, if we update $n$’s parameters (i.e., $\{\theta_{n,c} \mid c \in \text{in}(n)\}$) by maximizing Eq. (23), the true objective Eq. (2) will also improve.
Proof. Consider updating the parameters correspond to sum unit \( n \) (i.e., \( \{ \theta_{n,c} \mid c \in \text{in}(n) \} \)) by maximizing Eq. (23). We can re-arrange the entropy \( \text{ENT}(p_n) \) as follows:

\[
\text{ENT}(p_n) = \frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)}) + \tau \cdot \text{ENT}(p) \quad \text{(a)}
\]

\[
= \frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)}) + \tau \cdot P_n(n) \cdot \sum_{c \in \text{in}(n)} \left( -\theta_{n,c} \log \theta_{n,c} + \theta_{n,c} \cdot \text{ENT}(p_c) \right) + \text{const} \quad \text{(b)}
\]

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

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

\[
\frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)}) + \tau \cdot \text{ENT}(p) \quad \text{(24)}
\]

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

• **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., \( \{ \theta_{n,c} \mid c' \in \text{in}(n) \} \) ). 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. \( \square \)

• **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:

\[
\max_{\theta_{n,c}} L_{\text{surr}}(\theta_{n,c}; \theta_{\text{ref}}), \text{ such that } \sum_{c \in \text{in}(n)} \theta_{n,c} = 1. \quad \text{(25)}
\]

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

\[
\max_{\theta_{n,c}} \min_{\lambda} L_{\text{surr}}(\theta_{n,c}; \theta_{\text{ref}}) - \lambda (1 - \sum_{c \in \text{in}(n)} \theta_{n,c})
\]

26
Algorithm 5: Forward pass (expected flows)

1: Input: A non-deterministic PC $p$; sample $x$
2: Output: $\text{value} [n] := (x \in \text{supp}(n))$ for each unit $n$
3: foreach $n$ traversed in postorder do
4:   if $n$ is an input unit then $\text{value} [n] \leftarrow f_n (x)$
5:   elif $n$ is a product unit then
6:     $\text{value} [n] \leftarrow \prod_{c \in \text{in}(n)} \text{value} [c]$
7:   else // $n$ is a sum unit
8:     $\text{value} [n] \leftarrow \sum_{c \in \text{in}(n)} \theta_{n,c} \cdot \text{value} [c]$

Algorithm 6: Backward pass (expected flows)

1: Input: A non-deterministic PC $p$; $\forall n$, $\text{value} [n]$
2: Output: $\text{eflow} [n, c] := \mathbb{E}_{z \in \text{ref}(\{x \in \text{supp}(n)\}) (\{x, z\} \in \gamma_n \cap \gamma_c)}$
3: foreach pair $(n, c)$, where $n$ is a sum unit and $c \in \text{in}(n)$ do
4:   $\forall n, \text{context} [n] \leftarrow 0$; $\text{context} [n_c] \leftarrow \text{value} [n_c]$
5: foreach sum unit $n$ traversed in preorder do
6:   foreach $m \in \text{pa}(n)$ do (denote $g \leftarrow \text{pa}(m)$)
7:     $\text{flow} [g, m] = f$

Its KKT conditions can be written as:

$$\begin{cases}
\frac{F_{n,c_i}(D)}{|D|} \theta_{n,c_i} - \tau \cdot P_{n,c_i}(n; \theta^{\text{ref}}_c) \log \theta_{n,c_i} + 1 + \text{ENT}(P_{n,c_i}; \theta^{\text{ref}}_c) + \lambda = 0 \quad (\forall 1 \leq i \leq |\text{in}(n)|), \\
\sum_{c \in \text{in}(n)} \theta_{n,c} = 1.
\end{cases}$$

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 $D_\beta$ is equivalent to $D$, we can use the proof in Appendix A.1 and set $\beta = 1$ (the proof holds for any $\beta \in (0, 1]$).

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 $F_{n,c}(D_\beta)$ w.r.t. a softened categorical dataset, we can again adopt Alg. 1 and 2 by choosing

$$f_n (x) = \beta \cdot 1 [x \in \text{supp}(n)] + \frac{1 - \beta}{k} \cdot 1 [x \notin \text{supp}(n)].$$
B.2 Solving Equation 5

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

\[
\begin{align*}
    d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \gamma_{c_i} &= y \quad (\forall i \in \{1, \ldots, |\text{in}(n)|\}), \\
    \sum_{i=1}^{|\text{in}(n)|} e^{\varphi_{n,c_i}} &= 1.
\end{align*}
\]

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

- Solve for $y$. Given variables $\{\varphi_{n,c_i}\}_{i=1}^{|\text{in}(n)|}$, we update $y$ as
  \[
  y = \frac{1}{|\text{in}(n)|} \sum_{i=1}^{|\text{in}(n)|} d_i e^{-\varphi_{n,c_i}} - b \cdot \varphi_{n,c_i} + b \cdot \gamma_{c_i}.
  \]

- Solve for $\{\varphi_{n,c_i}\}_{i=1}^{|\text{in}(n)|}$. Given $y$, we first update each $\varphi_{n,c_i}$ 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:

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

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

\[
\varphi_{n,c_i} = \log\left(\sum_{i=1}^{|\text{in}(n)|} e^{\varphi_{n,c_i}}\right).
\]

B.3 Details of the Experiments on Deterministic PCs

**PC structures** For each dataset, we adopt 16 PCs by running Strudel [Dang et al., 2020] for $\{1000, 1200, 1400, \ldots, 4000\}$ iterations except for the dataset “dna”, which we ran Strudel for $\{50, 100, 150, \ldots, 800\}$ 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.

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 [Dang et al., 2020; Liang et al., 2017], 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.

**Detailed results** As an extension of [Table 1] [Table 2], provides the average test set log-likelihood for all adopted baselines.
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 (Dang et al., 2020), LearnPSDD (Liang et al., 2017), EinSumNet (Peharz et al., 2020a), LearnSPN (Gens & Pedro, 2013), ID-SPN (Rooshenas & Lowd, 2014), and RAT-SPN (Peharz et al., 2020b).

| 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    |
| baudivo  | -39.77| -39.87 | -40.55   | -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     |
| nlcs     | -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    |
| pnumbs*  | -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    |