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].
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, \textit{data softening} and \textit{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., \textit{deterministic} 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. For PCs that are not deterministic, every parameter update step can be done in $O(|p| \cdot |D|)$ 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.

\textbf{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., $\mathbf{x}$) for sets of variables and their joint assignments, respectively.

\section{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.

\textbf{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 $\beta \in (0, 1]$ 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 $\mathbf{x}$ with $K$ variables $\mathbf{X} := \{X_i\}_{i=1}^K$, where the $k$th variable takes value $x_k$, we can similarly 'soften' $\mathbf{x}$.

\footnote{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** $P_{x,\beta}$:

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

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 $\text{weight}(D_\beta, x)$ denotes the weight of sample $x$ in $D_\beta$:

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

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 [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 $D$ 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 $P_\theta$ 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 [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) [3], 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 $X$ 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 large fraction of sum units learned by a PC structure learning algorithm [17] are imbalanced. This paper focuses on two classes of PCs that support different types of queries: (i) PCs that allow variable assignments \( (\text{Determinism}) \) and effective regularizers for PCs? This question can be answered affirmatively — while 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 (HBM) [38], 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 —
We highlight the significance of the above issue by examining the fraction of sum units with imbal-

cance. The backward path illustrated in Alg. 2 begins at the root unit \( n \) and backward path that both take \( O(|p|) \) time. The forward path, as shown in Alg. 1, starts from the leaf units and traverses the PC in postorder to compute \( \forall n, \text{value}[n] := \mathbb{1}[x \in \text{supp}(n)] \); afterwards, the backward path illustrated in Alg. 2 begins at the root unit \( n \) and traverses the PC in preorder to
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...
We start with the negative result.

Let \( p \) be a deterministic PC, \( \beta \) be a hyperparameter, \( D \) be a dataset, \( \mathbb{1} \) be a boolean indicator, \( \sum \) be summation over all children, \( \mathcal{F} \) be the set of flows, \( \mathcal{T} \) be a set of all edges, \( \{\mathrm{supp}(n)\} \) be the set of supports, \( \phi \) be a probability, \( x \) be a variable, \( y \) be a variable, \( \theta \) be a vector of parameters, \( \mathcal{D}_\beta \) be the softened dataset. Theorem 3. Let \( f_\beta(x) = \beta \cdot \mathbb{1}[x \in \mathrm{supp}(n)] + (1 - \beta) \cdot \mathbb{1}[x \notin \mathrm{supp}(n)] \) in Alg. 1. Given a deterministic PC \( p \), a boolean dataset \( D \), and hyperparameter \( \beta \in (0.5, 1] \), the set of all flows \( \{F_{n,c}(D) \mid \forall \text{ edge } (n, c)\} \) w.r.t. the softened dataset \( \mathcal{D}_\beta \) can be computed by Alg. 1 within \( O(|p| \cdot |D|) \) 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 \( O(|p|) \) time using the flows, the overall time complexity to compute the MLE parameters is again \( O(|p| \cdot |D|) \).

**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 \( O(|p|) \) time [46]. However, it is still unclear whether the entropy regularization objective \( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \) (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 \( D \) such that \( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \) (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 \( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \). Specifically, Alg. 3 takes as input a deterministic PC \( p \) and all its edge flows w.r.t. \( D \), and returns a set of learned log-parameters that correspond to a stationary point of the objective [46]. 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) 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 (\( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \)) with all other parameters fixed. This is done by solving the set of equations in Eq. (5) using Newton’s method (lines 7-8) 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., node_prob[\( n \)]), which is progressively updated in lines 9-10.

**Theorem 4.** Alg. 3 converges monotonically to a stationary point of \( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \) (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 \( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \), which is determined by the entropies computed in its main loop (lines 4-10), the current parameters (line 5) and 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 (\( \mathbb{L}_{\text{ent}}(\theta; D, \tau) \)) with all other parameters fixed. This is done by solving the set of equations in Eq. (5) using Newton’s method (lines 7-8) 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., node_prob[\( n \)]), which is progressively updated in lines 9-10.

We compute parameters in the logarithm space for numerical stability.

This can be done by Alg. 4 shown in Appendix A.2. Lem. 1 proves that Alg. 4 takes \( O(|p|) \) time.

Details for solving Eq. (5) is given in Appendix B.2.
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 an 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 $\theta_{11}$ compared to $\theta_{12}$. Suppose $n = 10$, consider maximizing Eq. (2) with an empty dataset (i.e., we maximize $\text{ENT}(p_{n_1})$ 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) 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) for each of the 20 datasets. The detailed proof is in Appendix A.5.

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 $\text{LL}_{\text{train}}$ and $\text{LL}_{\text{val}}$ 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 $(\text{LL}_{\text{val}} - \text{LL}_{\text{train}})/\text{LL}_{\text{val}}$, 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,

\[ \alpha \in \{0.1, 0.4, 1.0, 2.0, 4.0, 10.0\}, \beta \in \{0.9996, 0.999, 0.996\}, \tau \in \{0.001, 0.01, 0.1\}. \]
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 \(Z\), 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 mini-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) 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., \(\forall i, \text{add an edge } Z_i \rightarrow X_i\)). 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.

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HCLT</th>
<th>Best PSDD</th>
<th>Best SPN</th>
<th>Dataset</th>
<th>HCLT</th>
<th>Best PSDD</th>
<th>Best SPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>accidents</td>
<td>-26.74±0.03</td>
<td>-28.29</td>
<td>-26.98</td>
<td>jester</td>
<td>-52.46±0.01</td>
<td>-54.63</td>
<td>-52.56</td>
</tr>
<tr>
<td>ad</td>
<td>-16.07±0.06</td>
<td>-16.52</td>
<td>-19.00</td>
<td>kdd</td>
<td>-2.18±0.00</td>
<td>-2.17</td>
<td>-2.12</td>
</tr>
<tr>
<td>baudio</td>
<td>-39.77±0.01</td>
<td>-41.51</td>
<td>-39.79</td>
<td>kosarek</td>
<td>-10.66±0.01</td>
<td>-10.98</td>
<td>-10.60</td>
</tr>
<tr>
<td>bbc</td>
<td>-251.04±1.19</td>
<td>-258.96</td>
<td>-248.33</td>
<td>msnbc</td>
<td>-6.05±0.01</td>
<td>-6.04</td>
<td>-6.03</td>
</tr>
<tr>
<td>bnetflix</td>
<td>-56.27±0.01</td>
<td>-58.53</td>
<td>-56.36</td>
<td>msweb</td>
<td>-9.98±0.05</td>
<td>-9.93</td>
<td>-9.73</td>
</tr>
<tr>
<td>book</td>
<td>-33.83±0.01</td>
<td>-35.77</td>
<td>-34.14</td>
<td>mlcsl</td>
<td>-5.99±0.01</td>
<td>-6.03</td>
<td>-6.01</td>
</tr>
<tr>
<td>crf52</td>
<td>-86.26±3.67</td>
<td>-92.38</td>
<td>-83.35</td>
<td>pubmed*</td>
<td>-23.64±0.25</td>
<td>-25.28</td>
<td>-22.40</td>
</tr>
<tr>
<td>cwebkb</td>
<td>-152.77±1.07</td>
<td>-160.5</td>
<td>-151.84</td>
<td>tmoviel</td>
<td>-50.81±0.12</td>
<td>-55.41</td>
<td>-51.51</td>
</tr>
<tr>
<td>dna</td>
<td>-79.05±0.17</td>
<td>-82.03</td>
<td>-81.21</td>
<td>tretail</td>
<td>-10.84±0.01</td>
<td>-10.90</td>
<td>-10.85</td>
</tr>
</tbody>
</table>
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


**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.

12
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]