Soft Learning Probabilistic Circuits

Soroush Ghandi, Benjamin Quost, Cassio de Campos

Soft Learning Probabilistic Circuits

Soroush Ghandi[1], Benjamin Quost[2], and Cassio de Campos[1]

[1] Eindhoven Unisversity of Technology
[2] University of Technology of Compiègne

Abstract. Probabilistic Circuits (PCs) are prominent tractable probabilistic models, allowing for a range of exact inferences. This paper focuses on the main algorithm for training PCs, LearnSPN, a gold standard due to its efficiency, performance, and ease of use, in particular for tabular data. We show that LearnSPN is a greedy likelihood maximizer under mild assumptions. While inferences in PCs may use the entire circuit structure for processing queries, LearnSPN applies a *hard* method for learning them, propagating at each sum node a data point through one and only one of the children/edges as in a hard clustering process. We propose a new learning procedure named SoftLearn, that induces a PC using a *soft* clustering process. We investigate the effect of this learning-inference compatibility in PCs. Our experiments show that SoftLearn outperforms LearnSPN in many situations, yielding better likelihoods and arguably better samples. We also analyze comparable tractable models to highlight the differences between soft/hard learning and model querying.

Keywords: Probabilistic circuits · Probabilistic inference · Probabilistic graphical models.

1 Introduction

Generative probabilistic models typically aim to learn the joint probability distribution of data, in order to perform probabilistic inference and answer queries of interest. However, not all the probabilistic models are the same in that regard. Models like variational autoencoders (VAEs) [11] and generative adversarial networks (GANs) [8] possess exceptional modeling prowess; nevertheless, their ability to perform probabilistic inference such as marginalization and conditioning is rather limited/intractable.

In contrast, *tractable* probabilistic models, such as probabilistic circuits (PCs), including the prominent sum-product networks (SPNs) [23,28], allow for a wider range of exact inferences arguably at the expense of some power of fitness. PCs are a type of Probabilistic Graphical Models (PGMs), a class of models using a graph-based representation to encode high-dimensional distributions [12]. Unlike Bayesian networks, which have a notoriously high complexity for general queries [3,4], PCs can produce several types of inferences in polynomial time under arguably mild assumptions [33].

While intractable models such as VAE and GANs rely on deep neural networks as their structure, their PGM counterparts find graph structures that fit well the data. Moreover, PCs need to further reach structures that facilitate exact inferences, which translates into a constrained structure learning problem. This latter problem has become an active line of research for many years. Many algorithms have been devised to learn PCs from data, among which LearnSPN [7] is considered a gold standard for its efficiency, performance, and ease of use. In addition to being the most widely known (and used) procedure for learning PCs—if not the best performing one in general, LearnSPN is also the building block of many subsequent algorithms [14,18,34].

In a nutshell, LearnSPN follows a greedy search approach. The data is recursively partitioned into smaller chunks: the structure of the network is defined recursively, either by grouping variables (giving birth to product nodes) or clustering instances (resulting in sum nodes). We claim that this greedy learning approach may result in inappropriate clusters and lead to partitioning marginals rigidly at sub-optimal locations, which can potentially lead to overfitting and poor generalization.

In this paper, we propose the SoftLearn procedure as a counterpart to LearnSPN, with the aim to mitigate such potential drawbacks. SoftLearn is a soft learning scheme akin to LearnSPN which may provide smoother marginals between data clusters so as to reduce the errors induced by misgrouped instances, and therefore lead to better likelihoods and arguably better samples. We compare SoftLearn with LearnSPN over a range of datasets and configurations, and show that SoftLearn manages to outperform LearnSPN in most cases, which empirically validates our claims regarding the potential improvements made by our soft learning scheme. We also draw comparisons with Cutset Networks [5], another prominent tractable probabilistic model with similar goals.

2 Related Work

Arguably, the most common practical approach to learn the structure of PCs is to recursively partition the data matrix over the instances (forming sum nodes) and variables (forming product nodes) in a top-down fashion [7,18]. [1] proposed to cluster the joint space of instances and variables, instead of alternating between instance and variable clustering. Conversely, another category of approaches consists in learning the structure in a bottom-up fashion by incrementally aggregating correlated variables [9,10].

In another line of work, several attempts have been made to learn the structure of PCs in a more principled way, based on non-parametric formulations [30] and/or Bayesian structure learning [29,35]. The Merged-L-SPN [24] algorithm proposes to merge subtrees in a post-processing approach to reduce computation complexity and improve the generalization of LearnSPN. There have also been a variety of approaches aimed at learning other structures of tractable probabilistic models, such as probabilistic sentential decision diagrams (PSDDs) [15] and Cutset networks [5,25].

3 Probabilistic Circuits

In this paper, we use the terms "SPNs" and "PCs" interchangeably. SPNs use circuits with three types of nodes: sum nodes can be interpreted as latent variables; product nodes encode context-specific independence; and leaves encode (tractable) univariate probability distributions [19]. Structurally, an SPN is a single-rooted directed acyclic graph (SDAG). A directed graph is defined by a finite set $\mathbf{N}$ of nodes and a set $\mathcal{E} \subseteq \mathbf{N} \times \mathbf{N}$ of ordered node pairs, called edges. For example, if $N, C \in \mathbf{N}$ and $(N, C) \in \mathcal{E}$, then we have a directed edge between N and C, written as $N \to C$. A acyclic directed graph does not contain any directed cycle (i.e. a path of directed edges from any node to itself, non-directed cycles being allowed). We consider graphs with only one root (i.e., a node without incoming edge), denoted by R. We assume nodes can only belong to a single graph; thus, $\mathbf{N}$ and $\mathcal{E}$ can be left implicit in the notation.

Any node N can be associated with important subsets:

- *children* $\operatorname{ch} N \coloneqq \{C \in \mathbf{N} : N \to C\}$ are those nodes to which there is a directed edge from N,
- *descendants* $\operatorname{de} N \coloneqq \operatorname{ch} N \cup \bigcup_{C \in \operatorname{ch} N} \operatorname{de} C$ can be accessed by a sequence of directed edges from N,
- *leaves* $\operatorname{lv} N \coloneqq \{D \in \operatorname{de} N : \operatorname{ch} D = \emptyset\}$ are descendants of N that do not have children themselves.

Each node N of an SDAG determines a sub-SDAG $N{\downarrow}$ rooted in N with nodes $\{N\} \cup \operatorname{de} N$—the whole SDAG can be written as $R{\downarrow}$. Given a non-leaf node N of an SDAG, two of its children $C_1, C_2 \in \operatorname{ch} N$ are said to be either *overlapping* if and only if $\operatorname{lv} C_1 \cap \operatorname{lv} C_2 \neq \emptyset$, or *disjoint* if and only if $\operatorname{lv} C_1 \cap \operatorname{lv} C_2 = \emptyset$.

Let $\boldsymbol{X}$ be a finite collection of random variables; $X \in \boldsymbol{X}$ denotes that a random variable X belongs to $\boldsymbol{X}$ (collections of random variables and their realizations are in boldface, as opposed to single random variables and their realizations). Here, $\operatorname{val} X$ stands for the set of possible realizations x of X, and $\operatorname{val} \boldsymbol{X}$ for the set of possible realizations $\boldsymbol{x}$ for the variables in $\boldsymbol{X}$, i.e. $\operatorname{val} \boldsymbol{X} = \bigtimes_{X \in \boldsymbol{X}} \operatorname{val} X$. Let $\boldsymbol{Y} \subseteq \boldsymbol{X}$ be a subcollection of the variables in $\boldsymbol{X}$: thus, if $X \in \boldsymbol{Y}$, then $X \in \boldsymbol{X}$ (the converse being not necessarily true). Joint realizations $\boldsymbol{x} \in \operatorname{val} \boldsymbol{X}$ or $\boldsymbol{y} \in \operatorname{val} \boldsymbol{Y}$ can be projected onto a subspace. For example, if $\boldsymbol{Y} \subseteq \boldsymbol{X}$, then $\boldsymbol{x}|_{\boldsymbol{Y}} \in \operatorname{val} \boldsymbol{Y}$; we use the same notation $\boldsymbol{y}|_X \in \operatorname{val} X$ to project $\boldsymbol{y}$ onto a variable $X \in \boldsymbol{Y}$.

A SPN encodes a probabilistic model over a collection of variables $\boldsymbol{X}$ [21,23,28]. It consists of an SDAG with structural constraints, and composed of three distinct types of nodes: sum nodes, associated with (numerical) parameters $\boldsymbol{w}$; product nodes, and distribution nodes: these latter describe simple distributions at the leaves, which can be recursively combined using sums and products, so that the root encodes a complex distribution.

Every node N in a SPN is associated with a collection of random variables, called its *scope*, over which it defines a probability distribution: e.g., $\boldsymbol{X}_N$ stands for the scope of N. The scope of a non-leaf node N is the union of its child scopes: $\boldsymbol{X}_N = \bigcup_{C \in \operatorname{ch} N} \boldsymbol{X}_C$. The root scope is $\boldsymbol{X}_R = \boldsymbol{X}$ (we assume every random

variable to be in the scope of at least one leaf). We will write the projection of a node scope using the node symbol: $\boldsymbol{x}|_N \coloneqq \boldsymbol{x}|_{\boldsymbol{X}_N}$.

A *sum node* S in the SPN is associated with a function $\boldsymbol{w}_S$ on $\operatorname{ch} S$ that returns edge weights $w_{S\to C} \coloneqq \boldsymbol{w}_S(C)$ for any $C \in \operatorname{ch} S$: thus, $\boldsymbol{w}_S$ is the *weight vector* associated with S. The weights should be non-negative, and we assume them normalized: for all $C \in \operatorname{ch} S$, $\sum_{C\in\operatorname{ch} S} w_{S\to C} = 1$ and $w_{S\to C} \geq 0$. The numerical parameters $\boldsymbol{w}$ of the entire SPN can simply be seen as the map $\boldsymbol{w} : S \mapsto \boldsymbol{w}_S$ that identifies this vector for any sum node in the SPN. The value $S(f)$ of any sum node S is recursively obtained from the values of its children: $S(f) \coloneqq \sum_{C\in\operatorname{ch} S} w_{S\to C} C(f)$. A *product node* P in the SPN is associated with a value $P(f)$, also recursively computed using the values of its children: $P(f) \coloneqq \prod_{C\in\operatorname{ch} P} C(f)$. Finally, any leaf in the SDAG is a *distribution node* D in the SPN. Distribution nodes define a probability distribution over their scope $\boldsymbol{X}_D$. We will assume the scope $\boldsymbol{X}_D$ of any leaf node D to consist of a single random variable, and thus leaf nodes to encode univariate distributions.

Assume we wish to calculate the expectation $R(f) \coloneqq \mathbb{E}_{R\downarrow,\boldsymbol{w}}(f)$ of a function f according to the distribution encoded by the SPN with root R and weights $\boldsymbol{w}$. Let f be a product of indicators over variables in $\boldsymbol{X}$ (this allows for a range of probabilistic queries, including conditionals and marginals). Let $\mathbb{E}_D$ denote the expectation operator with respect to the distribution in any leaf node D. The expectation $R(f)$ can be computed by propagating the expected values $D(f) \coloneqq \mathbb{E}_D(f_D)$ in the leaves to the root node, thereby associating each node N with an expected value $N(f)$. Distribution nodes, being always the leaves in the SDAG, are the terminal points of the recursive definition of the node values. We assume in the sequel that we can *efficiently evaluate* the expectations in the leaves, which opens the way to efficiently computing the expectation $R(f)$.

SPNs typically use additional structural assumptions to ensure the probabilistic model encoded is proper [21]. For any sum node S and any product node P in the SPN, it holds that

A1: $\boldsymbol{X}_{C_1} = \boldsymbol{X}_{C_2}$ for all $C_1, C_2 \in \operatorname{ch} S$; *(smoothness)*
A2: $\boldsymbol{X}_{C_1} \cap \boldsymbol{X}_{C_2} = \emptyset$ for all $C_1, C_2 \in \operatorname{ch} P$. *(decomposability)*

A SPN that meets both (A1) and (A2) is said to be *valid*.

4 Learning PCs

We first detail LearnSPN [7] and point out a potential drawback; then, we introduce our method SoftLearn, explaining how it differs from LearnSPN, and illustrating how it might mitigate some issues.

LearnSPN employs a greedy search in the space of SPNs, and augments the network in a top-down fashion accordingly. It initializes the network with a single node R representing the entire dataset (with scope $\boldsymbol{X}$), and then proceeds by recursively partitioning the dataset into smaller chunks based on instance/variable-wise groupings found in the data. For each variable-wise grouping, a product node P is added to the network (representing a partition of the variables into

Algorithm 1 LearnSPN($\mathcal{D}|_N, \boldsymbol{X}_N$)

1: **Input:** set of instances $\mathcal{D}|_N \subseteq \text{val}\,\boldsymbol{X}_N$ for a scope $\boldsymbol{X}_N$
2: **Output:** an SPN $N\downarrow$ representing a distribution over $\boldsymbol{X}_N$ learned from $\mathcal{D}|_N$
3: **if** $|\boldsymbol{X}_N| = 1$ **then**
4: $N \leftarrow$ leaf univariate distribution node D estimated from the variable's values in $\mathcal{D}|_N$
5: **else**
6: partition $\boldsymbol{X}_N$ into approximately independent subsets $\boldsymbol{X}_{C_j}$, that is, $(\boldsymbol{X}_{C_j})_{j=1,\ldots,J}$ is a partition of $\boldsymbol{X}_N$
7: **if** $J > 1$ **then**
8: $N \leftarrow \bigotimes_{j=1}^{J} \text{LearnSPN}(\mathcal{D}|_{C_j}, \boldsymbol{X}_{C_j})$
9: **else**
10: partition $\mathcal{D}|_N$ into subsets of similar instances $\mathcal{D}_i|_{C_i}$, where $C_i = N$, with $i = 1, \ldots, I$
11: **if** $I > 1$ **then**
12: $N \leftarrow \bigoplus_{i=1}^{I} \frac{|\mathcal{D}_i|_{C_i}|}{|\mathcal{D}|_N|} \text{LearnSPN}(\mathcal{D}_i|_{C_i}, \boldsymbol{X}_{C_i})$
13: **else**
14: $N \leftarrow \bigotimes_{j=1}^{|\boldsymbol{X}_N|} \text{LearnSPN}(\mathcal{D}|_{X_j}, \{X_j\})$
15: **return** N

conditionally independent groups); and each time instances are clustered, a sum node S is added to the network (representing a mixture of the corresponding instances). This process is recursively applied until a stopping criterion is met, at which point each group of data corresponds to a univariate distribution that can be modeled reliably in the corresponding leaf of the network. Product nodes P are created by using independence tests (pairwise tests will form a dependency graph, and variables in distinct components of the graph become the scope of the children of P), while sum nodes S are created by performing hard clustering on the instances with the induced children having same scope as S.

The learning scheme of LearnSPN can seemingly be improved, as it appears not to be consistent with how queries in PCs work. During inference, a PC may use the entire structure to process a particular query, while the relevant information to that query is only used to train *some parts* of the network structure, since at each sum node, a datapoint is propagated through one and only one of the children/edges as in a hard clustering process. Inherently, the response to a query will be mostly affected by the network parts trained on the relevant information; thus, this incompatibility does not lead to an erroneous response as long as the clustering in LearnSPN can well classify the queried datapoint. However, for datapoints that lie near the cluster borders, the response can become considerably erroneous in case of misgrouping by the clustering approach used in Line 10 of Algorithm 1.

Our proposal, SoftLearn, induces a PC using a soft clustering process, so as to alleviate the costs of such misgrouping. Thus, after clustering, each datapoint is shared among the children of the sum node proportionally to its cluster memberships; this way, each datapoint will be propagated through the entire

network, but with different weights indicating its importance to a particular part of the network. In order to do so, during clustering, for a set of K clusters $\{C_1, C_2, \ldots, C_K\}$, a set of K 'datapoint' weights are associated to each datapoint, $\{v_1, v_2, \ldots, v_K\}$ such that $\sum_i v_i = 1$. Each instance in the dataset starts with a datapoint weight equal to 1 and this is divided among the children of sum nodes as the algorithm recursively performs clustering.

However, propagating the weights associated with data points throughout the network and using them to learn a PC requires extensive adjustments to other parts of LearnSPN as well, beyond simply re-engineering the procedures, since clustering methods used to induce sum nodes, independence tests to induce product nodes, and distribution nodes at leafs need all to be adapted to deal with weighted datapoints.

We note that LearnSPN is a greedy maximizer of data likelihood, because each new node in the recursive construction can only increase the likelihood with respect to the alternative early stop of the procedure. This argument is detailed in Appendix D. A number of adaptations and implementations of LearnSPN have been proposed. We use the version implemented by [2] (see Algorithm 1) upon which we build our method, since such implementation achieved state of the art results. We nevertheless provide the results of the original implementation [7] in our experiments in Section 5.

4.1 Univariate density estimation

Following [2], we model the distribution over discrete or continuous variables using a multinomial or Gaussian distribution, respectively. These choices are not limiting, since PCs under this formulation can fit any distribution so long as structure learning is able to split the space properly and data are enough. In our case, each datapoint propagated in the learning algorithm has an associated weight. We employ the weights in our calculations as a measure of frequency. Take a univariate dataset $\mathcal{D}|_D = \{d_1, d_2, \ldots, d_m\}$ and a set of corresponding weights $V = \{v_1, v_2, \ldots, v_m\}$ at a particular leaf D, with $v_i > 0$ for all i (datapoints with zero weight are discarded). For discrete variables, we will have:

$$\widehat{P}(\boldsymbol{X}_D = k) = \frac{C_k}{\sum_{j=1}^{m} C_j}, \quad \text{where } C_j = \sum_{i:\ d_i = j} v_i. \tag{1}$$

For continuous variables, we use a Gaussian $\mathcal{N}(\widehat{\mu}, \widehat{\sigma})$ obtained with the proper derivations to achieve the reweighted estimation with Bessel's correction (derivations are omitted for ease of exposure): $\widehat{\mu} = (\sum_i v_i d_i)/(\sum_i v_i)$,

$$\widehat{\sigma} = \sqrt{\frac{\sum_{i=1}^{m} v_i}{(\sum_{i=1}^{m} v_i)^2 - \sum_{i=1}^{m} v_i^2} \sum_{i=1}^{m} v_i (d_i - \widehat{\mu})^2}. \tag{2}$$

4.2 Independence tests

For discrete variables, [2] propose to use the chi-square test of independence, and to add a product node if the variables can be split into independent sub-

sets. In brief, the chi-square test uses the difference between expected (under null hypothesis of independence) and observed "frequencies", using contingency tables. In order to account for weighted instances, we form *weighted contingency tables* and then proceed with the chi-square test based on the weighted tables. The calculations are similar to what is done in Expression (1) (we leave for the reader to fill in the simple gaps to perform this computation).

In the case of two continuous variables or of mixed variables, we first discretize the continuous variables, and then apply the weighted chi-square test, while [2] use the Kruskal and Kendall's tau tests. Adapting Kruskal and Kendall's tau to deal with weighted data and meaningful commensurable interpretation among tests seems to be a challenge by itself. Our choice is motivated by the clear interpretation of partial frequencies that now all tests have in common, regardless of the data type, so all tests are commensurable. Their efficacy is not largely affected so long as enough data points are available (which is the case, as we would indeed want to stop expanding the model otherwise).

4.3 Clustering

In addition to handling soft membership degrees so as to quantify the relevance of an instance to a group, the clustering methods in our approach should also be able to admit weighted datapoints as inputs. We investigate two options. The first one is an adjusted version of the K-means clustering algorithm. In order to work with weighted data, we only change the update rule of centroids to account for datapoint weights. Originally, assuming a centroid C is associated with m datapoints $\{d_1, d_2, \ldots, d_m\}$, it would be updated as: $C_{\text{upd}} = (\sum_i d_i)/m$. In our adjusted version where datapoints are associated with weights $\{v_1, \ldots, v_m\}$, the update rule is simply $C_{\text{new}} = \sum_i v_i d_i / \sum_i v_i$.

Additionally, we utilize a weight function $v(d, \{C_i\}, i)$, that computes the weight v_i of each datapoint d in each group i based on the group centroid C_i. We define an arguably natural reweighted function as:

$$v_i \coloneqq v(d, \{C_i\}, i) = \frac{\exp\{\beta(1 - \frac{||d - C_i||_F}{\sum_i ||d - C_i||_F})\}}{\sum_i \exp\{\beta(1 - \frac{||d - C_i||_F}{\sum_i ||d - C_i||_F})\}}.$$

To put it in words, this reweighted function (i) computes the distances of the datapoint d to group centroids $\{C_i\}_{\forall i}$ and normalizes them; (ii) computes an intermediate relevancy degree $1 - ||d - C_i||_F / \sum_i ||d - C_i||_F$ to each group, which trivially gives more (resp. less) value to groups that are closer to (resp. further away from) the datapoint; and (iii) applies a softmax function to the relevancy degrees.

The second clustering method we investigate is based on estimating mixtures of distributions using the Expectation-Maximization (EM) algorithm, under a conditional independence assumption. Let the dataset be $\mathcal{D}|_S$ over variables $\boldsymbol{X}|_S$. In order to perform EM clustering, we assume the underlying distribution to be

Algorithm 2 SoftLearn($\mathcal{D}|_N, \boldsymbol{X}_N, V|_N$)

Input: set of instances $\mathcal{D}|_N \subseteq \text{val}\,\boldsymbol{X}_N$ for a scope $\boldsymbol{X}_N$, set of weights $V|_N$ corresponding to datapoint instances
Output: an SPN $N\downarrow$ representing a distribution over $\boldsymbol{X}_N$ learned from $\mathcal{D}|_N$
if $|\boldsymbol{X}_N| = 1$ **then**
    $N \leftarrow$ leaf univariate distribution node D estimated from the variable's values in $\mathcal{D}|_N$ with weights in $V|_N$
else
    partition $\boldsymbol{X}_N$ into approximately independent subsets $\boldsymbol{X}_{C_j}$ using weighted independence tests with weights $V|_N$, that is, $(\boldsymbol{X}_{C_j})_{j=1,\ldots,J}$ is a partition of $\boldsymbol{X}_N$, using $\mathcal{D}|_N$ and $V|_N$
    if $J > 1$ **then**
        $N \leftarrow \bigotimes_{j=1}^{J} \text{SoftLearn}(\mathcal{D}|_{C_j}, \boldsymbol{X}_{C_j}, V|_{C_j})$
    else
        partition $\mathcal{D}|_N$ using a weighted soft clustering with datapoint weights $V|_N$, yielding new weights $\{V_i\}_{\forall i}$, with $i = 1, \ldots, I$ (I is the number of groups)
        update $V_i \leftarrow V_i \cdot V|_N$ and let $s_i = \sum V_i$
        if $I > 1$ **then**
            $N \leftarrow \bigoplus_{i=1}^{I} \frac{s_i}{\sum_j s_j} \text{SoftLearn}(\mathcal{D}|_{C_i}, \boldsymbol{X}_{C_i}, V_i)$
        else
            $N \leftarrow \bigotimes_{j=1}^{|\boldsymbol{X}_N|} \text{SoftLearn}(\mathcal{D}|_{X_j}, \{X_j\}, V|_{X_j})$
return N

a mixture of c fully factorized distributions (c is the targeted number of children for S): $P(\boldsymbol{X}) = \sum_{i=1}^{c} P(i) \prod_{X \in \boldsymbol{X}} P(X|i)$. In a nutshell, EM estimates the model parameters by iteratively alternating between an expectation and a maximization step, until a stopping criterion is met. The expectation step amounts to updating the (soft) memberships of the instances to the clusters, based on the current distribution estimates; the maximization step, to update the univariate distributions $P(X|i)$ and the group priors $P(i)$ based on the new memberships. In order to make EM work with weighted data, we only need to make two changes to the algorithm: we modify the univariate distribution updates in the maximization step to include the weights, as for univariate distribution estimation in Section 4.1, and we make the updates for the group priors $P(i)$ proportional to the sum of weights (instead of whole counts). Both steps are repeated iteratively until convergence to a stationary point (as usual in EM). The adjustments result in a soft learning scheme SoftLearn, whose pseudocode is given in Algorithm 2, which is not only more compatible with the soft inference scheme of PCs, but also uses each datapoint to learn every part of the network.

We close this section with an (extreme-case) illustrative example of the potential benefits of SoftLearn (so please bear with us). Assume that data $(X, Y) \in \mathcal{R}^2$ is generated from a PC as per Equation (3) (we use a flat notation for the PC):

$$(X, Y) \sim \; 0.5 \cdot \mathcal{N}_X(-0.5, 1) \otimes \mathcal{N}_Y(-2, 0.2) \oplus 0.5 \cdot \mathcal{N}_X(0.5, 1) \otimes \mathcal{N}_Y(2, 0.2). \quad (3)$$

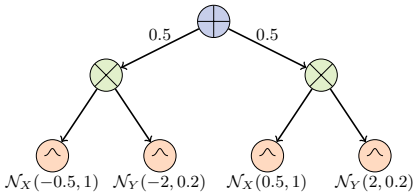


Fig. 1: PC structure equivalent to Expression (3), with a root sum node (in blue) with balanced weights to its children, which are two product nodes (in green), and four leaf distribution nodes (in salmon).

Figure 1 shows the equivalent graphical representation, which respects Assumptions A1–A2 and hence induces a valid joint distribution for X, Y. Figure 2 shows a sample with 1000 datapoints (green points) for each of the two mixture components, which are independent bivariate Gaussians. Sampling is performed top-down, choosing the direction to take in a sum node based on its weights, while following all paths from product nodes, yielding a full sample when the corresponding leaf nodes are reached and sampled. This generating distribution is obviously unknown to the learning algorithms.

We start running both LearnSPN and SoftLearn on this dataset. Initially, the conditions in Line 7 of Algorithm 1 and in Line 7 of Algorithm 2 both fail, X and Y being not independent (X is shifted based on Y), and the algorithms proceed with creating a sum node. Assume now that the clustering fails and partitions the points as per the gray line in Figure 2 (that is, at $X = 0$). Hence, in the continuation, LearnSPN will have to recursively deal with the groups of points in the left- and right-side of the gray line separately and independently, while SoftLearn will weight the pertinence of each data point to each of the two groups. After that first sum node, a product node will not appear again (X and Y being still considerably dependent). Assume now that the next clusterings run to create new sum nodes work perfectly well (otherwise, the difference in favor of SoftLearn could be even stronger, as we will see), and hence split points perfectly (positive Y go to one side and negative Y to the other). Finally, X and Y will be found independent (enough) and the four bivariate Gaussians will appear as in the following expressions. LearnSPN gives the model in Expression (4): $(X, Y) \sim$

$$\begin{aligned}\sim\ & .49(0.7 \cdot \mathcal{N}_X(-1, .69) \otimes \mathcal{N}_Y(-2, .2) \oplus .3 \cdot \mathcal{N}_X(-.69, .54) \otimes \mathcal{N}_Y(2.01, .2)) \oplus \\ & .51(.31 \cdot \mathcal{N}_X(.64, .53) \otimes \mathcal{N}_Y(-1.98, .2) \oplus .69 \cdot \mathcal{N}_X(.98, .68) \otimes \mathcal{N}_Y(2, .19)); \quad (4)\end{aligned}$$

and SoftLearn the model in Expression (5): $(X, Y) \sim$

$$\begin{aligned}\sim\ & .5(.5 \cdot \mathcal{N}_X(-.51, 1.01) \otimes \mathcal{N}_Y(-1.99, .2) \oplus .5 \cdot \mathcal{N}_X(.48, .99) \otimes \mathcal{N}_Y(2, .2)) \oplus \\ & .5(.51 \cdot \mathcal{N}_X(-.47, .99) \otimes \mathcal{N}_Y(-1.99, .2) \oplus .49 \cdot \mathcal{N}_X(.52, 1) \otimes \mathcal{N}_Y(2, .2)). \quad (5)\end{aligned}$$

In this hypothetical example, both approaches learn reasonable parameter estimates for the distribution leafs over Y (the true generating distribution is in

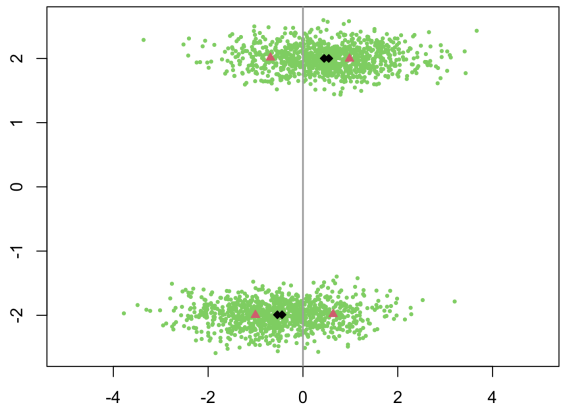


Fig. 2: Green points (X, Y) (resp. horizontal and vertical axes as usual) generated from the PC in Expression (3). The gray line is a hypothetical bad partition obtained for the root node. SoftLearn yields the mean parameters of the Gaussian leafs represented by the black diamonds, which still captures the whole Gaussians on both sides of the gray cut of the first step; standard LearnSPN yields the red triangles as means, as both clusters are necessarily treated separately.

Expression (3)), but LearnSPN struggles to get good estimates for X due to the bad clustering at the first sum node (the true should be Gaussians $\mathcal{N}(0.5, 1)$) and $\mathcal{N}(-0.5, 1)$)). On the other hand, SoftLearn has less difficulty to achieve good estimates for X. The Gaussian means of the distribution leaf nodes obtained by LearnSPN are shown in Figure 2 as red triangles, while the same is shown for SoftLearn in black diamonds. The figure clearly shows the difference between the hard LearnSPN approach and SoftLearn, suggesting that the latter might better cope with bad clustering results. Arguably, such results are inevitable when performing clustering with high-dimensional heterogeneous data—even though they are unlikely to be in practice as bad as in this example. It is not hard to imagine node pruning/merging techniques that would make the outcome of SoftLearn more compact in this situation, as some terms in Expression (5) represent somewhat similar Gaussians (and by that potentially we could recover even the simpler true structure of Expression (3)).

5 Experiments

We conduct a variety of experiments to evaluate our hypothesis about learning-inference compatibility and potential drawbacks of LearnSPN, by comparing its performance against our proposal SoftLearn. We proceed in three steps: (i) we compare the test log-likelihood of SoftLearn against that of LearnSPN on a variety of discrete and mixed datasets; (ii) we visually compare the quality of samples generated by SoftLearn and LearnSPN on an image dataset; and

Data	LearnSPN		CNET	Soft Learn	Data	LearnSPN		CNET	Soft Learn
	Gens	Correia				Gens	Correia		
NLTCS	-6.11	-5.99	-6.10	**-5.97**	DNA	-82.52	-83.67	-109.79	**-82.06**
MSNBC	-6.11	**-6.04**	-6.06	-6.04	Kosarek	-10.98	-11.04	-11.53	**-10.89**
KDD-2k	**-2.18**	-2.35	-2.21	-2.34	MSWeb	-10.25	-9.85	-10.20	**-9.68**
Plants	-12.97	-12.87	-13.37	**-12.57**	Book	-35.88	-34.33	-40.19	**-33.03**
Audio	-40.50	-39.84	-46.84	**-39.65**	E.Movie	**-52.48**	-56.84	-60.22	-55.22
Jester	-75.98	-53.23	-64.50	**-53.00**	WebKB	**-158.20**	-159.53	-171.95	-158.70
Netflix	-57.32	-56.82	-69.74	**-56.49**	Reut.52	**-85.06**	-87.93	-91.35	-88.33
Accid.	-30.03	**-28.89**	-31.59	-29.54	20ng	-155.92	-122.16	-176.56	**-121.09**
Retail	-11.04	-11.09	-11.12	**-10.88**	BBC	-250.68	**-247.81**	-300.33	-249.38
Pumsb.	-24.78	**-24.10**	-25.06	-24.81	Ad	-19.73	-18.53	**-16.31**	-20.30

Table 1: Performance results of SoftLearn vs. LearnSPN [7], LearnSPN [2], and CNET [26] over binary datasets.

(iii) we numerically compare the quality of samples generated by SoftLearn and LearnSPN on discrete datasets.

On discrete (binary) datasets, SoftLearn is compared against both implementations of LearnSPN [2,7]. For the rest of the experiments, since we have no access to the original implementation of LearnSPN [7], the comparison is held between SoftLearn and LearnSPN [2]. Throughout the experiments, both algorithms are optimized over two sets of hyperparameters, namely the chi-square test significance $p \in \{0.01, 0.001, 0.0001\}$ and the Laplace smoothing parameter of multinomial density estimation $\alpha \in \{0.1, 0.01, 10^{-6}\}$ (as usual on discrete data counts).

5.1 Test log-likelihood

Binary datasets. We compare the test log-likelihood of our method against LearnSPN [2,7] on twenty real-world datasets, of which thirteen were introduced in [17], and the other seven in [31]. The number of instances in the datasets varies from 2K to 388K, and the number of variables from 16 to 1556. In addition, we include the results of CNET [26], as this latter method combines a hard learning scheme with a hard inference scheme, representing the other side of the learning-inference spectrum. Note that [26] report 3 sets of results for 3 different versions of CNET, among which MCNET (which consists of an ensemble of CNETs) shows strong results and outperforms our method on several datasets; however, since our method does not include an ensemble of models and/or pruning, we only report the results for the vanilla CNET, as it is the most comparable version to SoftLearn (we considered beyond the scope to compare ensemble methods). The results for LearnSPN [7] and CNET [26] are reported from their corresponding paper. The results for LearnSPN [2] and SoftLearn are average of 9 repetitions (random initializations), and are summarized in Table 1.

Dataset	LearnSPN	SoftLearn	Early SoftLearn
bank	-20.139	**-19.993**	**-19.997**
electricity	-11.229	**-11.217**	-11.226
segment	-17.517	**-17.480**	-17.493
german	-22.720	**-22.395**	-22.470
vowel	-16.957	**-16.584**	-16.634
cmc	-9.850	**-9.811**	-9.838

Table 2: Performance results of LearnSPN vs. SoftLearn on mixed datasets. Last column uses SoftLearn with at most 2 iterations within clustering algorithms, thus forcing their early stop (often before convergence).

Mixed datasets. We compare the test log-likelihood of the methods over a selection of datasets from the OpenML-CC18 benchmark [32]. Table 2 presents the results averaged over 9 repetitions of the algorithm.

As the results in Table 1 suggest, SoftLearn manages to outperform both implementations of LearnSPN on 14 out of 20 discrete datasets, and to outperform CNET on 18 out of 20 datasets. It also performs better than LearnSPN [2] on all mixed datasets. This indicates that a soft learning scheme can have a positive impact on the performance of LearnSPN over discrete and mixed datasets. We attribute this to learning-inference compatibility caused by the soft learning scheme. SoftLearn results in softer margins between groups when clustering: as a result, if a datapoint is misgrouped, the induced error will not be as costly as with the original LearnSPN. We would also like to remind that for mixed datasets in SoftLearn, continuous variables are discretized for each independence test, which adds another layer of estimation to the algorithm; however, SoftLearn still manages to outperform its counterpart.

We also empirically study the possibility of early stopping clustering algorithms to try to demonstrate the greater robustness of SoftLearn to potentially worse clustering results (with the benefit of speeding up the learning, as one can have fewer iterations). We limited clustering to only 2 iterations (often stopping before convergence of the method). The results (last column in Table 2) suggest that the speed up comes with little harm to the accuracy of the model, which still outperforms LearnSPN run without early stopping (even though we acknowledge that broader experiments are needed for a more conclusive claim in this regard).

5.2 Image data

We employ LearnSPN [2] and SoftLearn to learn PCs over the binary-MNIST [13] dataset, and then we qualitatively evaluate the generated samples from both PCs. We decide to learn each PC on a single class of the dataset at a time in order to better visualize the samples. Figure 3 shows the generated samples from PCs learned on classes 9 and 5 of the dataset, respectively. As the results would suggest, the samples generated from the PC learned by SoftLearn appear to be

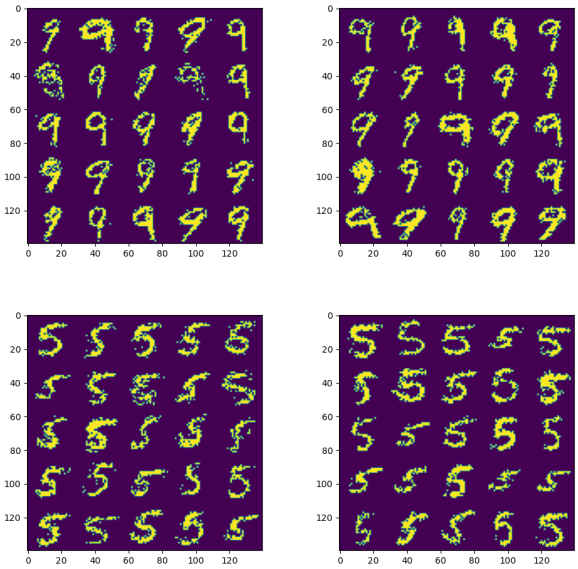


Fig. 3: Samples from PCs trained on Binary MNIST (n.9 and n.5), using (left) LearnSPN and (right) SoftLearn.

less cluttered. We believe the soft clustering to be responsible for some better samples when compared to those generated from the PC trained with LearnSPN. Samples drawn from SoftLearn are mostly clear to interpret, even if they can also be noisy or erroneous.

5.3 Quality of generated samples

Here we aim to evaluate and compare the quality of samples generated from PCs learned via LearnSPN and SoftLearn. To do so, we follow the experiment setup proposed in [6]. For each algorithm and dataset, we (i) learn a PC over the training dataset (using the best-performing set of hyperparameters); (ii) generate a synthetic dataset using the learned PC; (iii) learn another PC over the generated synthetic dataset; and (iv) interpret the test log-likelihood of learned PCs as an indicator for the quality of generated samples. The results of this experiment over 5 datasets are summarized in Table 3. While SoftLearn still manages to outperform LearnSPN on 4 out of 5 synthetic datasets, it shows a greater performance drop in 3 out of 5 experiments. This shows that while SoftLearn objectively generates better samples, whether or not its performance is more affected by the synthetic data remains inconclusive.

Dataset	Test Log Likelihood			
	LearnSPN		SoftLearn	
	Original	Synthetic	Original	Synthetic
NLTCS	-5.997	-6.051	-5.976	-6.022
Audio	-39.823	-40.307	-39.649	-40.143
Retail	-11.074	-11.196	-10.880	-10.974
MSWeb	-9.833	-10.058	-9.696	-9.982
Reuters-52	-87.838	-90.741	-88.609	-97.002

Table 3: Performance drop for PCs trained on synthetically generated samples, averaged over 3 repetitions.

6 Conclusion

In this paper, we shed some light on the importance of learning-inference compatibility of PCs and the potential drawbacks of greedy algorithms such as LearnSPN, which can potentially lead to rigid partitions and poor generalization. We also introduced SoftLearn, a soft structure-learning scheme as an attempt to mitigate the costs of such greedy behaviors. Our experiments showed that this soft method outperforms LearnSPN on a variety of datasets and configurations on test likelihoods, and that it arguably generates better samples likely due to its smoother partition margins.

This paper attempts to push a reasonably simple idea of soft clustering, yet with intricate changes required in the clustering and independence test methods. We truly believe structure learning to be a major point of improvement for PCs to reach even greater accuracy in real-world applications, in particular for structured/tabular data. Multiple avenues remain to be explored in learning the structure of PCs, which constitute excellent future work. We intend to continue the study with pruning/merging techniques and to move away from excessively structure-learning greedy approaches. In some sense, SoftLearn is a partial step in that direction by trying to mitigate greediness.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Adel, T., Balduzzi, D., Ghodsi, A.: Learning the structure of sum-product networks via an svd-based algorithm. In: Conference on Uncertainty in Artificial Intelligence (2015), `https://api.semanticscholar.org/CorpusID:15429402`
2. Correia, A., Peharz, R., de Campos, C.P.: Joints in random forests. Advances in neural information processing systems **33**, 11404–11415 (2020)
3. De Campos, C.P.: New complexity results for map in bayesian networks. In: IJCAI. vol. 11, pp. 2100–2106. Citeseer (2011)
4. De Campos, C.P., Cozman, F.G.: The inferential complexity of bayesian and credal networks. In: IJCAI. vol. 5, pp. 1313–1318. Citeseer (2005)

5. Di Mauro, N., Vergari, A., Basile, T.M., Esposito, F.: Fast and accurate density estimation with extremely randomized cutset networks. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10. pp. 203–219. Springer (2017)
6. Fakoor, R., Chaudhari, P., Mueller, J., Smola, A.J.: Trade: Transformers for density estimation. arXiv preprint arXiv:2004.02441 (2020)
7. Gens, R., Domingos, P.: Learning the structure of sum-product networks. In: International conference on machine learning. pp. 873–880. PMLR (2013)
8. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020)
9. Hsu, W., Kalra, A., Poupart, P.: Online structure learning for sum-product networks with gaussian leaves. arXiv preprint arXiv:1701.05265 (2017)
10. Kalra, A., Rashwan, A., Hsu, W.S., Poupart, P., Doshi, P., Trimponias, G.: Online structure learning for feed-forward and recurrent sum-product networks. Advances in Neural Information Processing Systems **31** (2018)
11. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
12. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
13. Larochelle, H., Murray, I.: The neural autoregressive distribution estimator. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 29–37. JMLR Workshop and Conference Proceedings (2011)
14. Lee, S.W., Heo, M.O., Zhang, B.T.: Online incremental structure learning of sum–product networks. In: Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part II 20. pp. 220–227. Springer (2013)
15. Liang, Y., Bekker, J., 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)
16. Liu, A., Van den Broeck, G.: Tractable regularization of probabilistic circuits. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 3558–3570. Curran Associates, Inc. (2021)
17. Lowd, D., Davis, J.: Learning markov network structure with decision trees. In: 2010 IEEE International Conference on Data Mining. pp. 334–343. IEEE (2010)
18. Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., Kersting, K.: Mixed sum-product networks: A deep architecture for hybrid domains. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32(1) (2018)
19. Peharz, R., Gens, R., Pernkopf, F., Domingos, P.: On the latent variable interpretation in sum-product networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(10), 2030–2044 (2017). https://doi.org/10.1109/TPAMI.2016.2618381
20. Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van Den Broeck, G., Kersting, K., Ghahramani, Z.: Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 7563–7574. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/peharz20a.html

21. Peharz, R., Tschiatschek, S., Pernkopf, F., Domingos, P.: On theoretical properties of sum-product networks. In: Artificial Intelligence and Statistics. pp. 744–752. PMLR (2015)
22. Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., Ghahramani, Z.: Random sum-product networks: A simple and effective approach to probabilistic deep learning. In: Uncertainty in Artificial Intelligence. pp. 334–344. PMLR (2020)
23. Poon, H., 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)
24. Rahman, T., Gogate, V.: Merging strategies for sum-product networks: From trees to graphs. In: UAI (2016)
25. Rahman, T., Jin, S., Gogate, V.: Look ma, no latent variables: Accurate cutset networks via compilation. In: International Conference on Machine Learning. pp. 5311–5320. PMLR (2019)
26. Rahman, T., Kothalkar, P., Gogate, V.: Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14. pp. 630–645. Springer (2014)
27. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect variable interactions. In: International Conference on Machine Learning. pp. 710–718. PMLR (2014)
28. Sánchez-Cauce, R., París, I., Díez, F.J.: Sum-product networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(7), 3821–3839 (2021)
29. Trapp, M., Peharz, R., Ge, H., Pernkopf, F., Ghahramani, Z.: Bayesian learning of sum-product networks. Advances in neural information processing systems **32** (2019)
30. Trapp, M., Peharz, R., Skowron, M., Madl, T., Pernkopf, F., Trappl, R.: Structure inference in sum-product networks using infinite sum-product trees. In: NIPS Workshop on Practical Bayesian Nonparametrics (2016)
31. Van Haaren, J., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 26, pp. 1148–1154 (2012)
32. Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter **15**(2), 49–60 (2014)
33. Vergari, A., Choi, Y., Liu, A., Teso, S., Van den Broeck, G.: A compositional atlas of tractable circuit operations for probabilistic inference. Advances in Neural Information Processing Systems **34**, 13189–13201 (2021)
34. Vergari, A., Di Mauro, N., Esposito, F.: Simplifying, regularizing and strengthening sum-product network structure learning. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II 15. pp. 343–358. Springer (2015)
35. Vergari, A., Molina, A., Peharz, R., Ghahramani, Z., Kersting, K., Valera, I.: Automatic bayesian density analysis. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33(1), pp. 5207–5215 (2019)

A Dataset Details

Dataset	Vars.	Train	Valid.	Test	Density
NLTCS	16	16181	2157	3236	0.332
MSNBC	17	291326	38843	58265	0.166
KDDCup2k	65	180092	19907	34955	0.008
Plants	69	17412	2321	3482	0.180
Audio	100	15000	2000	3000	0.199
Jester	100	9000	1000	4116	0.608
Netflix	100	15000	2000	3000	0.541
Accidents	111	12758	1700	2551	0.291
Retail	135	22041	2938	4408	0.024
Pumsb-star	163	12262	1635	2452	0.270
DNA	180	1600	400	1186	0.253
Kosarak	190	33375	4450	6675	0.020
MSWeb	294	29441	3270	5000	0.010
Book	500	8700	1159	1739	0.016
EachMovie	500	4524	1002	591	0.059
WebKB	839	2803	558	838	0.064
Reuters-52	889	6532	1028	1540	0.036
20 Newsgrp.	910	11293	3764	3764	0.049
BBC	1058	1670	225	330	0.078
Ad	1556	2461	327	491	0.008

Table 4: Discrete datasets statistics.

Dataset	Vars.		Size
	Categorical	Numeric	
bank	10	7	45211
electricity	2	7	45312
segment	1	19	2310
german	14	7	1000
vowel	3	10	990
cmc	8	2	1473

Table 5: Mixed datasets statistics.

As mentioned in Section 5.1, we use a set of twenty real-world binary datasets [17,31] and 6 real-world mixed datasets [32] for the log-likelihood experiments. Over the binary datasets, the number of instances varies from 2K to 388K, and the number of variables from 16 to 1556; over the mixed datasets, the number of variables varies from 9 to 21, and the number of instances from 990 to 45312.

The details regarding these datasets are outlined in Tables 4 and 5. Note that the binary datasets are already divided into train, validation, and test sets, and hence, the details regarding the size of each set are also reported in the table.

B Experiment Details

In this section, we present supplementary results and details of our experiments, in addition to the results reported in Section 5. For each method (LearnSPN and SoftLearn), and for each dataset category (binary and mixed), we present a table outlining more details regarding the experiments. Each table provides a more detailed version of the results, which decomposes the results over the clustering method and provides the standard deviation of the results over the 9 repetitions of the experiments. In addition to that, each table also provides the hyperparameter configurations to reproduce the reported results. Tables 6 and 8 are dedicated to the experiments of LearnSPN and SoftLearn on binary datasets. Similarly, tables 7 and 9 are dedicated to respective experiments of LearnSPN and SoftLearn on mixed datasets.

Dataset	**LearnSPN**					
	EM			**K-means**		
	Test LL	p	α	**Test LL**	p	α
NLTCS	-5.997 ± 0.008	0.01	0.01	-5.995 ± 0.007	0.01	0.1
MSNBC	-6.042 ± 0.003	0.01	0.1	**-6.041** ± 0.001	0.01	0.1
KDDCup2k	-2.350 ± 0.001	0.001	0.1	-2.360 ± 0.003	0.0001	0.1
Plants	-12.878 ± 0.026	0.01	0.1	-12.908 ± 0.020	0.01	0.1
Audio	-39.841 ± 0.018	0.001	10^{-6}	-39.938 ± 0.038	0.001	0.1
Jester	-53.235 ± 0.031	0.001	0.1	-53.234 ± 0.026	0.001	10^{-6}
Netflix	-56.818 ± 0.027	0.001	10^{-6}	-56.844 ± 0.028	0.001	0.1
Accidents	**-28.895** ± 0.122	0.0001	0.1	-29.114 ± 0.070	0.0001	0.01
Retail	-11.092 ± 0.047	0.0001	0.1	-11.142 ± 0.017	0.0001	0.1
Pumsb-star	**-24.101** ± 0.088	0.0001	0.1	-24.206 ± 0.085	0.0001	0.1
DNA	-83.674 ± 0.233	0.0001	0.1	-83.798 ± 0.182	0.0001	0.1
Kosarek	-11.043 ± 0.035	0.0001	0.1	-11.188 ± 0.037	0.0001	0.1
MSWeb	-9.847 ± 0.025	0.0001	0.1	-10.015 ± 0.018	0.0001	0.1
Book	-34.334 ± 0.093	0.0001	0.1	-34.428 ± 0.080	0.0001	0.1
EachMovie	-56.842 ± 0.078	0.0001	0.1	-57.129 ± 0.069	0.0001	0.1
WebKB	-159.533 ± 0.227	0.0001	0.1	-160.601 ± 0.331	0.0001	0.1
Reuters-52	-87.932 ± 0.144	0.0001	0.1	-88.400 ± 0.161	0.0001	0.1
20 Newsgrp.	-122.162 ± 0.138	0.0001	0.1	-122.827 ± 0.121	0.0001	0.1
BBC	-248.293 ± 0.464	0.0001	0.1	**-247.815** ± 0.248	0.0001	0.1
Ad	-18.539 ± 0.152	0.001	0.1	-18.738 ± 0.403	0.01	0.01

Table 6: Performance of LearnSPN [2] on binary datasets.

Dataset	LearnSPN					
	EM			K-means		
	Test LL	p	α	Test LL	p	α
bank	-20.139 ± 0.030	0.01	0.1	-20.277 ± 0.045	0.01	0.01
electricity	-11.229 ± 0.011	0.001	0.01	-11.488 ± 0.010	0.0001	0.1
segment	-17.517 ± 0.081	0.01	0.1	-17.663 ± 0.090	0.01	0.01
german	-22.720 ± 0.158	0.001	0.1	-22.857 ± 0.174	0.001	0.1
vowel	-16.957 ± 0.079	0.01	10^{-6}	-17.086 ± 0.076	0.01	10^{-6}
cmc	-9.850 ± 0.087	0.01	0.1	-9.902 ± 0.128	0.01	0.1

Table 7: Performance of LearnSPN [2] on mixed datasets

Dataset	SoftLearn					
	EM			K-means		
	Test LL	p	α	Test LL	p	α
NLTCS	-5.979 ± 0.006	0.01	10^{-6}	**-5.974** ± 0.002	0.01	0.01
MSNBC	-6.056 ± 0.003	0.01	0.01	-6.048 ± 0.001	0.01	0.01
KDDCup2k	-2.372 ± 0.006	0.0001	10^{-6}	-2.345 ± 0.005	0.01	10^{-6}
Plants	-12.643 ± 0.015	0.01	0.1	**-12.572** ± 0.013	0.01	10^{-6}
Audio	**-39.659** ± 0.023	0.01	0.1	-40.346 ± 0.014	0.01	0.01
Jester	**-53.005** ± 0.041	0.01	0.1	-53.545 ± 0.019	0.01	0.1
Netflix	**-56.491** ± 0.016	0.01	0.1	-57.698 ± 0.015	0.01	10^{-6}
Accidents	-30.092 ± 0.133	0.0001	0.1	-29.544 ± 0.049	0.0001	0.01
Retail	-11.040 ± 0.021	0.0001	0.1	**-10.887** ± 0.010	0.01	10^{-6}
Pumsb-star	-24.818 ± 0.244	0.0001	0.1	-28.397 ± 0.212	0.0001	0.01
DNA	-83.026 ± 0.240	0.0001	0.01	**-82.062** ± 0.078	0.01	10^{-6}
Kosarek	-11.108 ± 0.009	0.0001	0.1	**-10.890** ± 0.020	0.01	10^{-6}
MSWeb	-9.881 ± 0.049	0.001	0.1	**-9.688** ± 0.007	0.001	10^{-6}
Book	-34.173 ± 0.080	0.0001	0.1	**-33.031** ± 0.022	0.01	10^{-6}
EachMovie	-57.401 ± 0.166	0.0001	0.1	-55.225 ± 0.058	0.0001	10^{-6}
WebKB	-162.361 ± 1.400	0.0001	0.1	-158.703 ± 1.015	0.01	0.01
Reuters-52	-90.240 ± 0.525	0.001	0.1	-88.338 ± 0.239	0.01	0.01
20 Newsgrp.	-121.218 ± 0.138	0.0001	0.1	**-121.091** ± 0.127	0.01	10^{-6}
BBC	-249.381 ± 0.937	0.0001	0.1	-250.080 ± 0.533	0.01	0.01
Ad	-20.309 ± 0.794	0.01	0.01	-40.129 ± 1.791	0.01	0.01

Table 8: Performance of SoftLearn on binary datasets.

Dataset	SoftLearn					
	EM			K-means		
	Test LL	p	α	Test LL	p	α
bank	**-19.993** ± 0.044	0.0001	0.1	-20.132 ± 0.033	0.01	0.01
electricity	**-11.217** ± 0.021	0.01	10^{-6}	-11.422 ± 0.012	0.0001	0.1
segment	**-17.480** ± 0.071	0.001	10^{-6}	-17.625 ± 0.100	0.001	0.01
german	-22.423 ± 0.186	0.01	0.1	**-22.395** ± 0.187	0.01	0.1
vowel	**-16.584** ± 0.156	0.01	10^{-6}	-17.347 ± 0.169	0.01	0.01
cmc	-9.820 ± 0.098	0.01	0.01	**-9.811** ± 0.060	0.001	0.01

Table 9: Performance of SoftLearn on mixed datasets

C Comparison with State of the Art

Dataset	LearnSPN		CNET	Learn PSDD	HCLT	EiNet	RAT SPN	Soft Learn
	Gens	Correia						
NLTCS	-6.110	-5.995	-6.10	-6.03	-5.99	-6.015	-6.01	**-5.974**
MSNBC	-6.113	**-6.041**	-6.06	-6.04	-6.05	-6.119	-6.04	-6.048
KDDCup2k	**-2.182**	-2.350	-2.21	-2.12	-2.18	-2.183	-2.13	-2.345
Plants	-12.977	-12.878	-13.37	-13.79	-14.26	-13.676	-13.44	**-12.572**
Audio	-40.503	-39.841	-46.84	-41.98	-39.77	-39.879	-39.96	**-39.659**
Jester	-75.989	-53.234	-64.50	-53.47	**-52.46**	-52.563	-52.97	-53.005
Netflix	-57.328	-56.818	-69.74	-58.41	**-56.27**	-56.544	-56.85	-56.491
Accidents	-30.038	-28.895	-31.59	-33.64	**-26.74**	-35.594	-35.49	-29.544
Retail	-11.043	-11.092	-11.12	**-10.81**	-10.84	-10.916	-10.91	-10.887
Pumsb-star	-24.781	**-24.101**	-25.06	-33.67	-23.64	-31.954	-32.53	-24.818
DNA	-82.523	-83.674	-109.79	-92.67	**-79.05**	-96.086	-97.23	-82.062
Kosarek	-10.989	-11.043	-11.53	-10.81	**-10.66**	-11.029	-10.89	-10.890
MSWeb	-10.252	-9.847	-10.20	-9.97	-9.98	-10.026	-10.12	**-9.688**
Book	-35.886	-34.334	-40.19	-34.97	-33.83	-34.739	-34.68	**-33.031**
EachMovie	-52.485	-56.842	-60.22	-58.01	**-50.81**	-51.705	-53.63	-55.225
WebKB	-158.204	-159.533	-171.95	-161.09	**-152.77**	-157.282	-157.53	-158.703
Reuters-52	**-85.067**	-87.932	-91.35	-89.61	-86.26	-87.368	-87.37	-88.338
20 Newsgrp.	-155.925	-122.162	-176.56	-161.09	-153.40	-153.938	-152.06	**-121.091**
BBC	-250.687	**-247.815**	-300.33	-253.19	-251.04	-248.332	-252.14	-249.381
Ad	-19.733	-18.539	-16.31	-31.78	**-16.07**	-26.273	-48.47	-20.309

Table 10: Performance results of SoftLearn vs. LearnSPN [7], LearnSPN [2], CNET [26], LearnPSDD [15], Hidden Chow Liu Trees [16], Einsum Networks [20], and RAT-SPN [22] over binary datasets.

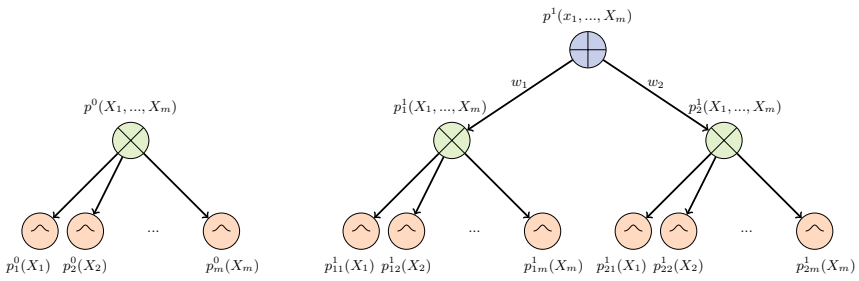
Once again, we would like to note that our motivation for proposing SoftLearn is not to compete with the state of the art methods, but to introduce a better *base model* compared to LearnSPN. LearnSPN is utilized as a base model

(or a building block) for many subsequent algorithms that managed to achieve impressive competitive results, outperforming LearnSPN itself. Such algorithms can utilize SoftLearn interchangeably, and based on the results reported in this paper, we believe that utilizing SoftLearn instead of LearnSPN will lead to noticeable performance gains, helping other algorithms achieve or surpass state of the art. Hence, our experiments were mainly designed to compare SoftLearn with its main competitor, LearnSPN, on a variety of test configurations. Nevertheless, we include a comparison between SoftLearn and some of the state of the art methods (namely LearnPSDD [15], Hidden Chow Liu Trees [16], Einsum Networks [20], and RAT-SPN [22]) on binary datasets in table 10.

SoftLearn manages to outperform LearnPSDD on 16 out of 20 datasets, EiNet on 14 out of 20 datasets, and RAT-SPN on 15 out of 20 datasets. The only method that performs better on average than SoftLearn is HCLT (SoftLearn still outperforms HCLT on 8 out of 20 datasets), which has a larger set of hyperparameters and is fine-tuned over a larger (and more precise) set of hyperparameter configurations. In addition, SoftLearn manages to achieve the best results (among all the aforementioned methods) over 6 out of 20 datasets (On a separate not, we would like to mention that ID-SPN [27] also manages to statistically outperform SoftLearn on average, however, we do not consider ID-SPN a reasonable competitor since its resulting structures are orders of magnitude larger than the structures learned by SoftLearn and LearnSPN. This means that ID-SPN potentially achieves better performance at the cost of over-parametrization and larger amount of computations, which does not lead to an unbiased comparison). These results show that despite its simple design as a base model (like LearnSPN), SoftLearn is competitive to state of the art models, while having a very large room to grow when used as a building block of more elaborate methods.

D Theoretical Intuition

In this section, we will provide theoretical insight into how methods like LearnSPN and SoftLearn optimize the global likelihood. Before going into the details, we first reiterate the learning process of LearnSPN/SoftLearn and establish necessary assumptions. In LearnSPN/SoftLearn, each learning iteration does one of the following tasks: i) adding a (factorized) leaf node at the end of a path; ii) adding an internal sum node; iii) adding an internal product node. We assume that we have the option to terminate LearnSPN/SoftLearn at any arbitrary iteration of learning. Doing so means that we stop clustering over instances/variables and finalize the PC by adding factorized distributions to each path that does not already end in a leaf node. We call the resulting PC produced by this process the *alternative PC* at iteration t. If we terminate the algorithm at the root of the PC, then the alternative PC would be a fully factorized distribution as in Figure 4a. Similarly, if we first add a sum node (with two children) to the root of the PC, and then proceed to terminate the learning, the alternative PC would have a structure similar to Figure 4b



(a) Caption (b) Caption

Fig. 4: Caption

Depending on the task performed by the learning algorithm, the difference between any two consecutive alternative PCs (alternative PC at iteration t and alternative PC at iteration $t+1$) can be: i) nothing (adding leaf nodes is inconsequential in alternative PCs since the process of termination includes adding a leaf node to any path without leaves); ii) a sum node; iii) a product node. If there is no difference between consecutive alternative PCs, then the likelihood of the data stays the same between the two PCs. In the case where the difference between two alternative PCs is in a product node, it can be easily deducted that the likelihood still remains the same, since a product node by itself (with factorized leaves) cannot induce any changes to the likelihood (the product of two factorized distributions over disjoint variables is equal to the product of univariate distributions over each variable). This leaves us only with the case where the difference between two consecutive alternative PCs is in a sum node. In this case (we can take the PCs represented in Figures 4a and 4b as an example; the same logic can be generalized to any arbitrary structure at any arbitrary iteration, with the exception that the difference in the likelihood will be weighted by a positive multiplier), the difference in the likelihood stems from the difference between $p^0(X_1, ..., X_m)$ and $p^1(X_1, ..., X_m)$ (or in the general case, the difference between $p^t(X_1, ..., X_m)$ and $p^{t+1}(X_1, ..., X_m)$), where $p^0(X_1, ..., X_m)$ is a factorized distribution learned on some data $\mathcal{D}$, and $p^1(X_1, ..., X_m)$ is a mixture of C (number of output clusters) factorized distributions based on the resulting clusters, learned from the same data $\mathcal{D}$. If we can guarantee that p^1 has a better likelihood compared to p^0, then we can conclude that in every iteration of learnSPN/SoftLearn, the likelihood either increases or stays the same, a process that gradually maximizes the likelihood of the data as the learning algorithm proceeds.

Whether or not p^1 is an increase over p^0 in terms of likelihood depends on the clustering algorithm. Yet, we can prove that there is always a solution that obtain the same likelihood as (or better than) the alternative option. Without losing generality, assume that we are adding a sum node with two children. In the case of soft clustering, one can equally divide each sample between the two clusters (weighing half to each side), which will lead to the same likelihood as

the alternative fully factorized model. Hence, any soft clustering algorithm that performs a job at least as good as this (which is surely expected) obtains the same or better likelihood after the sum node is added. When using hard clustering, the argument is slightly different (as we cannot "split" the data points in half to use in each side). In this case, one can think of the following split: choose an arbitrary data point and take all its $t \geq 1$ copies to be represented in one of the child of the sum node (with weight w equal to t over the number of data points in consideration) and all other points to the other child (with weight $1 - w$). It is not hard to see that both children will have better likelihood than the one of the alternative model for the part of the data points they represent, and thus the appropriate weighting yields a better model than the alternative model.

In addition to the intuitive perspective on clustering methods, we would also like to mention that some algorithms such as hard/soft EM, if properly initialized, can theoretically guarantee that the resulting mixture will have a better likelihood compared to the alternative factorized model.