Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning

Antonio Vergari and Nicola Di Mauro and Floriana Esposito

University of Bari "Aldo Moro", Bari, Italy
{antonio.vergari,nicola.dimauro,floriana.esposito}@uniba.it

Abstract. The need for feasible inference in Probabilistic Graphical Models (PGMs) has lead to tractable models like Sum-Product Networks (SPNs). Their highly expressive power and their ability to provide exact and tractable inference make them very attractive for several real world applications, from computer vision to NLP. Recently, great attention around SPNs has focused on structure learning, leading to different algorithms being able to learn both the network and its parameters from data. Here, we enhance one of the best structure learner, LearnSPN, aiming to improve both the structural quality of the learned networks and their achieved likelihoods. Our algorithmic variations are able to learn simpler, deeper and more robust networks. These results have been obtained by exploiting some insights in the building process done by LearnSPN, by hybridizing the network adopting tree-structured models as leaves, and by blending bagging estimations into mixture creation. We prove our claims by empirically evaluating the learned SPNs on several benchmark datasets against other competitive SPN and PGM structure learners.

1 Introduction

Probabilistic Graphical Models (PGMs) [13] use a graph-based representation eliciting the conditional independence assumptions among a set of random variables, thus providing a compact encoding of complex joint probability distributions. The most common task one want to solve using PGMs is *inference*, a task that becomes intractable for complex networks, a difficulty often circumvented by adopting approximate inference. For instance, computing the exact marginal or conditional probability of a query is a #P-complete problem [27].

However, there are many recently proposed PGMs where inference becomes tractable. They include graphs with low *treewidth*, such as *tree-structured graphical models* where each variable has at most one parent in the network structure [7], and their extensions with mixtures [18] or latent variables [6], or *Thin Junction Trees* [4], allowing controlled treewidths. Being more general than all of these models and yet preserving tractable and exact inference, *Sum-Product Networks* (SPNs) [23] provide an interesting model, successfully employed in image reconstruction and recognition [23,9,1], speech recognition [21] and NLP [5] tasks. Similarly to *Arithmetic Circuits* (ACs) [16], to which they are equivalent

for finite domains [26], they compile a high treewidth network into a deep probabilistic architecture. By layering inner nodes, sum and product nodes, they encode the probability density function over the observed variables, represented as leaf nodes. SPNs guarantee inference in time linear to their network size [23], and they possibly becomes more expressively efficient as their depth increases [17].

Recently the attention around SPNs has focused on structure learning algorithms as ways to automate latent interaction discovery among observed variables and to avoid the cost of parameter learning [8,19,10,26]. While many of these efforts concentrated on optimizing the likelihoods of the models, little attention has been devoted to the structural quality of such models, or to understand how data quality effects the learning process.

In this paper we extend and simplify one of the state-of-the-art SPN structure learning algorithm, LearnSPN [10], providing several improvements and insights. We show how to a) learn simpler SPNs, i.e. ones with less edges, parameters and more layers, b) stop the building process earlier while preserving goodness of fit, and c) be more robust and resilient in estimating the dependencies from data. In order to accomplish this we limit the number of node children when building the network, we introduce tractable multivariate distributions, in the form of Chow-Liu trees [7], as leaves of an hybrid architecture without adding complexity to the network, and we enhance the mixture models of an SPN via bootstrap samples, i.e. by applying bagging for the likelihood function estimation.

We produced different algorithmic variants incorporating one or more of these enhancements, and thoroughly evaluated them on standard benchmark datasets, both under the structure quality perspective and the more usual data likelihood gain. We compared them against the original algorithm, the best SPN structure learner up to now, ID-SPN [26], and MT [18], learning mixture of trees, reported to be the second best algorithm in [26] on the same datasets.

2 Sum-Product Networks

Sum-Product Networks have been introduced in [23] as a general architecture efficiently encoding an unnormalized probability distribution over a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. The graphical representation of an SPN consists of a rooted DAG, S, whose leaves correspond to univariate distributions of observable variables in $\mathbf{X}$, while internal nodes are *sum* or *product* nodes. The *scope* of each internal node i, denoted as $\mathbf{X}_{\psi_i}$, is defined as the set of variables appearing as its descendants. The sub-network S_i rooted at node i encodes the unnormalized distribution over its scope. The *parameters* of the network are the positive weights w_{ij} associated to each edge $i \rightarrow j$ in S, where i is a sum node. As in [10,26] we will refer to the whole network as S, and, for a given state $\mathbf{x}$ of the variables $\mathbf{X}$, we will indicate as $S(\mathbf{x})$ the unnormalized probability of $\mathbf{x}$ according to the SPN S , i.e., the value of S's root when the network is evaluated after $\mathbf{X} = \mathbf{x}$ is observed. Intuitively, sum nodes encode mixtures over probability distributions whose coefficients are the children weights, while product nodes identify factorizations over independent distributions. Examples

of different SPNs are shown in Figure 1. For the sake of simplicity, we are considering $\mathbf{X}$ to be discrete valued random variables (the extension to the continuous case is straightforward [23]).

An SPN is said to be *decomposable* if the scopes of the children of product nodes are disjoint, and *complete* when the scopes of sum nodes children are the same. Decomposability and completeness imply *validity* [23], i.e. the property of correctly and exactly computing each evidence probability by evaluating the network, that is, for a network S and a state $\mathbf{x}$, $P(\mathbf{X} = \mathbf{x}) = S(\mathbf{x})/Z$, where Z is the *partition function*, defined as $Z = \sum_{\mathbf{x}} S(\mathbf{x})$. From now on, we will assume the SPNs we are considering to be valid.

To compute $S(\mathbf{x})$, the whole network is evaluated bottom-up. For a leaf node i, representing the variable X_k, $S_i(x)$ corresponds to univariate distribution values for x, i.e. $S_i(x) = P(X_k = x)$. While for a generic internal node i, a) if it is a product node, then $S_i(\mathbf{x}_{\psi_i}) = \prod_{i \to j \in S} S_j(\mathbf{x}_{\psi_j})$; b) if it is a sum node, then $S_i(\mathbf{x}_{\psi_i}) = \sum_{i \to j \in S} w_{ij} S_j(\mathbf{x}_{\psi_j})$. If the weights of each sum node i sum to one, $\sum_j w_{ij} = 1$, and the leaf distributions are normalized, then the network will compute the exact, normalized, probability, i.e. $\forall \mathbf{x}, P(\mathbf{X} = \mathbf{x}) = S(\mathbf{X})$. For the rest of the paper we will assume SPNs being normalized in this way. Following these considerations, it can be demonstrated that all the marginal probabilities, the partition function and all MPE queries and states can be computed in time linear in the *size* of the network, i.e. its number of edges [10].

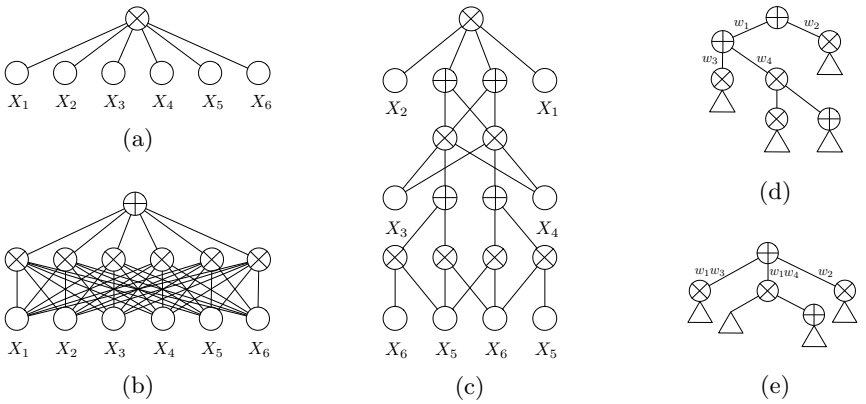


Fig. 1: Examples of SPNs: a naive factorization over 6 random variables (1a), a shallow mixture standing for a pointwise kernel density estimation (1b) and a deeper architecture (1c) over the same scope (weights are omitted for simplicity). An SPN with alternated layers of nodes of the same kind (1e) obtained from pruning the one in (1d) and yet encoding the same distribution.

Note that this does not automatically imply the tractability of inference, for it to be feasible the number of edges should be polynomial in the number of random variables. One way to control the number of edges is to layer the nodes

into a deep architecture, where parameters are reused across the levels. We define the *depth* of a network as the longest path from the root to a leaf node in networks with strictly interleaving layers of nodes of the same kind. Note that it is always possible to convert an SPN in such a layered architecture. A node c having parents $\{p_i\}_{i=1}^{I}$ sharing its same type can be pruned, and c's children, $\{g_j\}_{j=1}^{J}$, can be directly attached to each p_i. If c was a sum node, the new weights for each g_j can be computed as $w_{p_i c} w_{c g_j}$, see Figures 1d and 1e. For a visual comparison of how the depth impacts the network size, see Figure 1c, where local dependencies are exploited compared to the shallow SPN in Figure 1b where they are not and each leaf is fully connected to the upper layer. Moreover, it has been shown that increasing the depth of a network makes it more expressively efficient [17]. Even the number of weights in a network impacts weight learning feasibility [9], when hand-crafted SPNs are employed, usually some sparsity constraint is applied during learning to prune as many edges as possible [23]. The opportunity to directly learn the structure of an SPN offers a way to govern the time future inferences and learning stages will take. However, up to now, the focus of structure learning algorithms has not been the quality of the learned architectures in terms of depth, number of parameters and edges, but the ability to better capture the data probability distribution in terms of the achieved likelihood scores.

2.1 Structure learning

Learning the structure of SPNs has always been tackled as a constraint-based search problem exploiting heuristics and statistical tests to elicit the local latent relationships in the data [8,19]. The first principled structure learning algorithm is LearnSPN [10], it performs a greedy construction of treed SPNs, also referred to as *formula* SPN [17], i.e. networks with inner nodes having at most one parent. Nevertheless it is still the most attractive for its simplicity, parallelizability and ability to learn the networks weights as well. It is the starting point for all the extensions we will introduce.

The core idea behind LearnSPN, which is sketched in Algorithm 1, is to grow a tree top down by recursively partitioning the input data matrix consisting of a set T of rows as i.i.d instances, over V, the set of columns, i.e. the features. For each call of LearnSPN on a submatrix, column splits add child nodes to product nodes, while those on rows extend sum nodes. To split columns, the corresponding features are checked for independency by means of a statistical test in the splitFeatures procedure, while clusterInstances is employed to aggregate rows together by a similarity criterion. The weights of sum nodes children represent the proportions of instances falling into the computed clusters (line 13). Termination is achieved in two cases, when the current submatrix contains only one column (line 3) or when the number of its rows falls under a certain threshold m (line 5). In the former, a leaf node, standing for a univariate distribution, is introduced by estimating it from the submatrix data entries, i.e., for categorical random variables by counting their values occurrences while applying Laplace

Algorithm 1 LearnSPN(T, V, α, m)

Input: a set of row instances T over a set of column features V; m: minimum number of instances to split; α: Laplace smoothing parameter
Output: an SPN S encoding a pdf over V learned from T
if $|V| == 1$ **then**
    $S \leftarrow \mathsf{univariateDistribution}(T, V, \alpha)$
else if $|T| < m$ **then**
    $S \leftarrow \mathsf{naiveFactorization}(T, V, \alpha)$
else
    $\{V_j\}_{j=1}^{C} \leftarrow \mathsf{splitFeatures}(V, T)$
    if $C > 1$ **then**
        $S \leftarrow \prod_{j=1}^{C} \mathsf{LearnSPN}(T, V_j, \alpha, m)$
    else
        $\{T_i\}_{i=1}^{R} \leftarrow \mathsf{clusterInstances}(T, V)$
        $S \leftarrow \sum_{i=1}^{R} \frac{|T_i|}{|T|} \mathsf{LearnSPN}(T_i, V, \alpha, m)$
return S

Algorithm 2 naiveFactorization(T, V, α)

Input: a set of row instances T over a set of column features V, α Laplace smoothing parameter
Output: an SPN S encoding a product of factors V estimated from T
return $S \leftarrow \prod_{j=1}^{|V|} \mathsf{univariateDistribution}(T, V_j, \alpha)$

smoothing with parameter α. In the latter, the random variables for the submatrix columns are modeled with a naive factorization, i.e. they are considered to be independent and a product node is put over a set of univariate leaf nodes, as in Algorithm 2.

It is worth noting that the two splitting procedures depend on each other: the quality of row clusterings is likely to be enhanced by column splits correctly identifying dependent features. At the same time, well done instance splits would allow for finer independence tests in the next call of the algorithm. The likelihood on the data is never computed explicitly, the search for local hidden relationships leads to small submatrices whose likelihood can be easily estimated via naive factorizations. In [10] it is said that if the two splitting procedures are able to exactly separate the columns into independent groups and the rows by similarity, they would lead to locally optimal structures in the terms of data likelihood.

While the presented version of LearnSPN allows for different kinds of splitting procedures, the way they are implemented is crucial. In [10] a G-Test is used to check for the independence of pairs of random variables in splitFeatures: if the test p-value is less than a threshold ρ, then the two features are considered to be independent. Two subsets of approximately dependent features are produced by exploring features starting from one randomly chosen. Rows are aggregated into an adaptive number of clusters l, by employing the hard online EM algorithm. Columns V_j are assumed to be independent given the row clusters C_i, i.e., $P(V) = \sum_i P(C_i) \prod_j P(V_j|C_i)$. To control the cluster numbers, an exponential

prior on clusterings in the form of $e^{-\lambda l|V|}$ is used, with λ as a tuning parameter. In this concrete formulation, the algorithm searches for treed SPNs in the hyperparameter space formed by m, α, ρ and λ. If the algorithm finds all the features in the complete data matrix to be independent, it would build an SPN representing a naive factorization consisting in a single product node over $|V|$ leaves (like the one in Figure 1a over $X_1, \ldots, X_6$). However, this degeneration is prevented by forcing a split on the rows during the first call of the algorithm. On the other hand, in the case of each cluster containing a single instance, the network would be similar to a pointwise kernel density estimation (see Figure 1b for a graphical example, where six instances are considered).

3 Contributions

Being greedy by design, LearnSPN, is highly prone to learn far from optimal structures, both in terms of likelihood scores and network quality. This is particularly true when the training data is noisy and/or scarce. The statistical tests implemented by the splitting procedures can easily be mislead into wrong choices, and, worst of all, overfitting could lead to overcomplex networks for which inference can be an issue. Given these shortcomings, our contributions will affect them in several ways: here we show how limiting the number of node children while splitting leads to deeper and simpler networks, how more complex and yet tractable factorizations as leaves are able to reduce network complexity favoring early termination, and how model averaging by bagging can be blended into the definition of SPNs in order to get more robust models.

3.1 Deepening by limiting node splits

Our first contribution is to limit the number of node children while learning, resulting in networks that will be deeper, and potentially with less edges and parameters. Basically, we fix to two the number of submatrices the current matrix can split into, for each call of LearnSPN, i.e., $C \leq 2$ and $R \leq 2$ on lines 8, 12 of Algorithm 1. This is already achieved in LearnSPN when splitFeatures is implemented to decompose the features into two subsets, thus our variation will effectively limit only row splits.

This simplifying idea is based on a number of observations. The first one is that checking earlier and more often for independency among features enhances the quality of row splits, reduces the average number of sum node children making the network less wide but deeper. Successive splits along the same axis can be seen as a hierarchical divisive clustering process whose termination is achieved when splits along the other axis happen. The aim is to slow down this greedy decision process to make it take the most out of data, while trying to exactly determine all the splits along one axis at once would lead to local optima faster. Secondly, we can observe that other splits on the same axis could always be done in the following iterations, but only if necessary. As noted in the previous section, in this way we are not limiting the expressive power of the learned SPNs

at all, since after pruning nodes whose parent has their same type, the number of children per node can be more than two. This variation can be seen as an application of the simplicity bias criterion. By not committing to complex choices too early, the algorithm remains able to explore structures where the splits on the features could lead to better networks. Moreover, it is also more robust to overclustering in noisy situations since in those cases it can receive the valuable help from the anticipated independence tests. As our experiments suggested, this is particularly true when a row split is forced on the first call of LearnSPN.

Under these observations, to implement clusterInstances in our simplified version one could still use any clustering algorithm, but limiting the number of clusters to two, thus resulting in one hyperparameter less to tune.

3.2 Regularization by tractable multivariate distribution hybridization

As our second contribution, we tackle the problem of fitting tractable multivariate distributions as leaves of an SPN. By substituting them to the naive factorizations we aim at a twofold objective: improving the network likelihood by capturing finer local dependencies when estimating leaf distributions, and being able to stop the building process earlier.

To balance complexity and expressive power, we chose tree-structured distributions as the simplest tractable distributions that are able to model more dependencies than a naive factorization while not adding complexity to the structure by adding additional parameters or layers. *Directed Tree distributions* [18] are Bayesian Networks whose nodes, standing for the random variables X_j, have at most one parent each, Pa_j, which leads to the following factorization for the joint distribution $P(\mathbf{X}) = \prod_j P(X_j|Pa_j)$. From this formulation, it is easy to see how we preserve the same complexity for computing inference on complete evidences: it reduces again to a product of the same number of factors. Differently from a naive factorization, each factor here provides the valuable information about the conditional dependency between parent and child variables. Moreover, tree distributions guarantee that marginalizations, and MPE inference as well, can be computed in time linear to the number of factors [18]. The validity of the network is also preserved, row and column splits guarantee that the scope of the multivariate leaves will not compromise decomposability nor completeness.

The classic Chow-Liu algorithm [7] can be used to learn the tree distribution that best approximates the underlying joint distribution in terms of the Kullback-Liebler divergence. The algorithm builds a maximum spanning tree over the graph formed by the pairwise Mutual Information (MI) values among each pair of columns in the current submatrix. It then turns the undirected tree into a directed one by randomly selecting a root and traversing it. In practice, we substitute the procedure naiveFactorization from line 6 of Algorithm 1 with the procedure LearnCLT as shown in Algorithm 3. In our hybrid architecture now leaves can be simple univariate distributions like before *or* subnetworks S_t encoding multivariate tree distributions over the set $\mathbf{X}_{\psi_t}$ of random variables. Computing $S_t(\mathbf{x}_{\psi_t})$ equals to evaluate the Chow-Liu tree on $\mathbf{x}_{\psi_t}$.

Algorithm 3 LearnCLT(T, V, α)

1: **Input:** a set of instances T over a set of features V, α Laplace smoothing parameter
2: **Output:** a Chow-Liu tree S encoding a pdf over V learned from T
3: $M \leftarrow \mathbf{0}_{|V|\times|V|}$
4: **for each** $X_u, X_v \in V$ **do**
5: $\quad M_{u,v} \leftarrow$ estimateMutualInformation(X_u, X_v, α)
6: $\mathcal{T} \leftarrow$ maximumSpanningTree(M)
7: **return** $S \leftarrow$ traverseTree$(\mathcal{T})$

The complexity of learning a Chow-Liu tree is quadratic in the number of features taken into account, however efficient implementations can lower it to sub-quadratic times [18]. Note that we limit this stage to the last steps of the algorithm, where submatrices have usually only few features.

The hyperparameter α is still needed to smooth the marginals in Algorithm 3, line 5. If we consider the original formulation of LearnSPN, m and α offer the only simplistic forms of regularization, however, when using naive factorizations, m is not as valuable in terminating the search earlier as it is when tree distributions are employed. In fact, to have the naive independency assumption hold, one has to let the search continue up to small submatrices, where even larger ones can be equally or better approximated by a Chow-Liu tree. As we will show in the Section 5, by doing a grid search over the hyperparameter m, the best likelihood wise structures on a validation set, employing naive factorizations, would prefer smaller values for m, while the best ones introducing tree distributions would likely be the ones learned with larger values for m. In this way one is able to prefer even simpler models, possibly avoiding overfitting.

3.3 Strengthening by model averaging

While on the previous sections we concentrated on improving the structure quality, our next contribution will focus on directly increasing the likelihood estimation capability of LearnSPN. In order to do so, we leverage a very well know statistical tool for robust parameter estimation: *bagging* [12].

Before performing a split on rows, we draw k bootstrapped samples T_{B_i} from the current submatrix (sampling rows with replacements) and on each of those we call clusterInstances, thus leading to k learned SPNs, S_{B_i}. We then build the resulting network as a sum node as the parent of all the other sum nodes representing the roots of the networks S_{B_i}. We introduce $1/k$ as the weight for these nodes, as in usual parameter estimation by bagging[1]. The bagged SPN would result in this more robust estimation: $\hat{S} = \sum_{i=1}^{k} \frac{1}{k} S_{B_i}$. Note that while this approach is theoretically applicable at each stage of the algorithm before learning the mixture components, it will eventually build a network with an exponential number of edges, making inference unfeasible in practice. Hence we

[1] We have experimented with weights proportional to the likelihood score obtained by each bootstrapped component, however the gain over uniform ones is negligible.

Algorithm 4 baggingSPN(T, V, α, m, k)

1: **Input:** a set of row instances T over a set of column features V; m: minimum number of instances to split; α: Laplace smoothing parameter
2: **Output:** an SPN S encoding a pdf over V learned from T
3: $\{T_{B_i}\}_{i=1}^{k} \leftarrow$ bootstrapSamples(T, k)
4: **return** $S \leftarrow \sum_{i=1}^{k} \frac{1}{k}$LearnSPN$(T_{B_i}, V, \alpha, m)$

limit this step to the first recursive call, where the split on rows is mandatory and it is more likely to improve the model estimation globally. The procedure, which we call baggingSPN is shown in Algorithm 4.

Our approach is similar to the one used in the discriminative framework for tasks like regression, when model averaging is applied over a set of bootstrapped weak learners, e.g. forests of regression trees [12]. In our case it is worth noting that the resulting architecture is still an SPN: by pruning the roots of the SPNs S_{B_i} as shown in the previous section, we end up with a single sum node averaging local and possibly perturbed distributions over the same scopes; as a matter of fact the validity of the network is preserved as well as the ability to use it as a generative model. Inference is tractable as long as the number of edges remains polynomial. Indeed, the newly introduced complexity grows linearly in the number of the bootstrapped components, k. To limit the growth of the number of edges and parameters in the network, it would be possible to merge identical subtrees to compact the model; or, by separating the bootstrapped SPNs aggregation from their learning phases, one could use a more informative procedure, i.e. a L1-regularized gradient descent, to select only the components consistently contributing to the likelihood increase.

4 Related works

The first SPN structure learner has been proposed in [8]. It splits the data matrix top-down, however splits and their meaning are different from LearnSPN: instances are clustered only once, at the start, and feature clustering is achieved by K-Means, which is not able to locate independencies correctly. Arbitrary sum nodes are then introduced as product nodes parents. The EM algorithm is needed to learn the network weights. On the other hand, [19] proceeds bottom-up by selecting the features to merge iteratively with a Bayesian-Dirichlet test, then sum nodes and their parameters are learned by maximizing the MI through the Information Bottleneck method, however considering only the best likelihood scoring features to reduce the high complexity of the approach. Like in [8] the learned SPNs are not tree structured, while the overall approach is still greedy.

The recent ID-SPN algorithm [26], by exploiting both indirect and direct interactions among variables, unifies works on learning SPNs through top-down clustering with works on learning tractable Markov networks [16]. ID-SPN learns Sum-Products of Arithmetic Circuits (SPACs) models which consist of sum and

product nodes as inner nodes, and ACs as leaf nodes. ID-SPN consistently outperforms the previous SPN and several other PGM structure learners [26]. AC leaves can potentially better approximate more complex distributions than our Chow-Liu tree leaves, however, at the cost of increasing structural complexity. Note that our approach differs from ID-SPN not only on the choice of tractable distributions to model leaves, but also on governing the greedy process. Starting from a single AC, ID-SPN splits each leaf into two new ACs only if this improves directly the likelihood on data, while we let the search be guided indirectly by the splitting procedures, estimating leaf distributions only at the end. This, combined with the high complexity of the base algorithm to learn ACs [16], makes ID-SPN very slow in practice.

Another search approach based on directly maximizing the likelihood is found in [20] where less expressive SPNs, named *Selective SPNs*, allow the efficient optimization of a closed form of the likelihood function by stochastic local search.

On the side of mixtures of generative models, a very competitive structure learner algorithm is MT [18]. MT learns a distribution of the form: $Q(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i T_i(\mathbf{x})$, where the distributions T_i, learned with the Chow-Liu algorithm [7], are the mixture components and $\lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1$ are their coefficients. [18] finds the best components and weights as (local) likelihood maxima by using EM, with k fixed in advance. MT is reported as the second most accurate model after ID-SPN in [26]. The hybrid SPNs we propose can express more latent interactions than a single mixture of Chow-Liu trees, moreover, they allow leaf scopes to consist of single random variables or subsets of the whole scope. Hybrid architectures like ours are referred to as *Generalized SPNs* in [22].

While leading to potentially long learning times, the EM algorithm is still the preferred choice to learn mixtures of generative models, like in the recent case of mixtures of *Cutset Networks* (CNets) [24]. CNets are weighted probabilistic model trees with Chow-Liu trees as leaves. Their inner nodes are OR nodes conditioning on a variable, thus they do not represent latent features and despite the depth of the tree they are shallow architectures. Similar works, applying bagging to a generative scenario like ours, are those from [25] and [2]. In the former, bagging is used to regularize the variant of EM proposed to determine the number of components in boosting a mixture of density estimators. In the latter, again applied to density estimation, a perturbing strategy derived from bootstrapped or totally random samples lead to more robust mixtures of tree distributions. A further work by the same authors relaxes the Chow-Liu algorithm on random subspaces to further differentiate mixture components [3].

5 Experiments

To empirically evaluate our enhancements, namely Binary row clustering, Tree distributions as leaf nodes, and model averaging through Bagging, we consider these algorithmic variations of LearnSPN: SPN-B, implementing the first one, SPN-BT adding to that the second one, SPN-BB including the clustering fix and bagging, and SPN-BTB, which incorporates all three of them.

For the original LearnSPN we used the publicly available Java implementation[2] from [10]. For the aforementioned ID-SPN, we used the implementation from the Libra toolkit [14]. However, since we were not able to reproduce the results shown in [26], we used the best learned models scores, kindly provided by the authors (which we thank). As a third competitor, we used MT, whose implementation can also be found in the Libra package. We implemented all our variations in Python[3], taking advantage of the scikit-learn version of EM used for Gaussian Mixture Models[4] for our variant of the clusterInstances procedure.

	$\|V\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$		$\|V\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$
NLTCS	16	16181	2157	3236	**DNA**	180	1600	400	1186
MSNBC	17	291326	38843	58265	**Kosarek**	190	33375	4450	6675
KDDCup2k	65	180092	19907	34955	**MSWeb**	294	29441	3270	5000
Plants	69	17412	2321	3482	**Book**	500	8700	1159	1739
Audio	100	15000	2000	3000	**EachMovie**	500	4525	1002	591
Jester	100	9000	1000	4116	**WebKB**	839	2803	558	838
Netflix	100	15000	2000	3000	**Reuters-52**	889	6532	1028	1540
Accidents	111	12758	1700	2551	**BBC**	1058	1670	225	330
Retail	135	22041	2938	4408	**Ad**	1556	2461	327	491
Pumsb-star	163	12262	1635	2452					

Table 1: Datasets used and their statistics.

We evaluated the inferred networks comparing both the learned structures quality and their likelihood scores on an array of 19 datasets, firstly introduced in [15] and [11], now a standard to compare graphical model structure learning algorithms [15,10,16,26]. They are binarized versions of datasets from tasks like classification, frequent itemset mining, recommendation. The training, validation and test splits statistics are reported in Table 1. Their features range from 16 to 1556, while training instances from 1670 to 291326, making them very suitable to evaluate how we improve LearnSPN under our dimensions on different scenarios.

Our first experimental objective is to verify whether SPN-B and SPN-BT do learn deeper and more compact structures compared to LearnSPN, and to do this we measure the number of edges, layers and parameters of each learned model. Secondly, we compare all algorithms in terms of average test data likelihoods to verify if structural improvements damage likelihood scores and how much bagging, on the other hand, improves them. ID-SPN does not appear in the first confrontation since we were provided only the model scores.

[2] `http://spn.cs.washington.edu/learnspn/`

[3] Code is available at `http://www.di.uniba.it/~vergari/code/spyn.html`

[4] `http://goo.gl/HNYjfZ`

5.1 Experimental design

For each algorithm, we selected the best parameter configurations based on the average validation log-likelihood scores, then evaluated such models on the test sets. For LearnSPN we performed an exhaustive grid search for $\rho \in \{5, 10, 15, 20\}$, $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$, $m \in \{10, 50, 100, 500\}$ and $\alpha \in \{0.1, 0.2, 0.5, 1, 2\}$, leaving EM restarts to the default 4. For SPN-B and SPN-BT we use the same parameter space for ρ, m and α, to make the comparison as fair as possible. We leave all the default parameters for scikit-learn's EM unchanged, with the exception of the number of restarts which we set to 3. Note that in [10], α and m were not considered hyperparameters, we are introducing them to show effective regularization in the form of early stopping achieved when not naive factorizations are employed. For ID-SPN, please refer to the original article for its complete experimental settings; here we point out that its overparametrization, which is likely a key factor in its performance, required a uniform random search in the parameter space, since an exhaustive one would have been unfeasible. Concerning MT, we learned a number of components k from 2 up to 30, with increments of 2, rerunning each experiment five times to mitigate EM random initializations.

To reduce the complexity of the experiments for both SPN-BB and SPN-BTB, we did not employed a grid search, but we used the best validation values for ρ, m and α as previously found by SPN-B and SPN-BT respectively. We learned $k = 50$ bootstrapped components for each of the two, then, we composed the models by adding one component at a time, selecting as the resulting composite model the one with the best validation score.

5.2 Results and discussion

In Table 2 are reported the edge, the layers and the parameters statistics for the best models learned by LearnSPN, SPN-B and SPN-BT. As it can be seen, the introduction of the limited Binary row clustering always makes the networks deeper and significantly reduces the number of edges for both variants, except for SPN-B on Netflix. It is worth noting that on datasets like BBC, Reuters-52 and MSWeb, while the number of parameter increases, the networks grow deeper and not wider, preventing edge explosions. When SPN-BT yields smaller networks than SPN-B, e.g. on Plants, Audio, Netflix, Kosarek and Book, the gain is huge in terms of edges and parameters saved, while considerable depths are preserved. On the other hand, on cases like NLTCS, MSNBC, KDDCup2K, Jester and BBC no structural improvement is observed if we add to the count the number of edges in the Chow-Liu trees. Table 3 reports the average test log likelihoods; the scores in bold are significantly better than all others under a Wilcoxon signed rank test with p-value of 0.05. Figure 2a shows the total number of times one algorithm wins under this same test. SPN-B proves no worst than the original algorithm on all but two datasets, scoring even six victories, the same value achieved by ID-SPN. The addition of the Chow-Liu trees variant in SPN-BT improves SPN-B scores on 13 datasets, confirming the ability of trees to better capture local dependencies at low levels.

	# edges				# layers			# params		
	LearnSPN	SPN-B	SPN-BT		LearnSPN	SPN-B	SPN-BT	LearnSPN	SPN-B	SPN-BT
NLTCS	7509	1133	1133	(1125)	4	15	15	476	275	275
MSNBC	22350	4258	4258	(3996)	4	21	21	1680	1071	1071
KDDCup2k	44544	4272	4272	(4166)	4	25	25	753	760	760
Plants	55668	13720	5948	(1840)	6	23	20	3819	2397	490
Audio	70036	16421	4059	(478)	8	23	15	3389	2631	105
Jester	36528	10793	10793	(8587)	4	19	19	563	1932	1932
Netflix	17742	25009	4132	(203)	4	25	14	1499	4070	82
Accidents	48654	12367	10547	(6687)	6	25	26	5390	2708	1977
Retail	7487	1188	1188	(1153)	4	23	23	171	224	224
Pumsb-star	15247	12800	9984	(6175)	8	25	23	1911	2662	1680
DNA	17602	3178	4225	(2746)	6	13	12	947	884	1113
Kosarek	7993	8174	2216	(1311)	6	27	21	781	1462	242
MSWeb	17339	9116	7568	(6797)	6	27	34	620	1672	1446
Book	42491	9917	3503	(3485)	4	15	13	1176	1351	430
EachMovie	52693	20756	20756	(17861)	8	23	23	1010	2637	2637
WebKB	52498	45620	8796	(6874)	8	23	16	1712	6087	1128
Reuters-52	307113	77336	77336	(59197)	12	31	31	3641	8968	8968
BBC	318313	63723	63723	(41247)	16	27	27	1134	6147	6147
Ad	70056	23606	23606	(20079)	16	59	59	1060	1222	1222

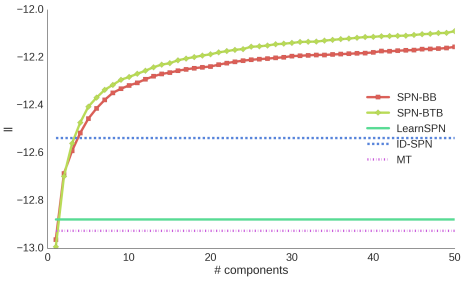
Table 2: Structural quality results for the best validation models for LearnSPN, SPN-B and SPN-BT as the number of edges, layers and parameters. For SPN-BT are reported the number of edges considering those in the Chow-Liu leaves and without considering them (in parenthesis).

	LearnSPN	SPN-B	SPN-BT	ID-SPN	SPN-BB	SPN-BTB	MT
NLTCS	-6.110	-6.048	-6.048	**-5.998**	-6.014	-6.014	-6.008
MSNBC	-6.099	-6.040	-6.039	-6.040	**-6.032**	-6.033	-6.076
KDDCup2k	-2.185	-2.141	-2.141	-2.134	-2.122	**-2.121**	-2.135
Plants	-12.878	-12.813	-12.683	-12.537	-12.167	**-12.089**	-12.926
Audio	-40.360	-40.571	-40.484	-39.794	-39.685	**-39.616**	-40.142
Jester	-53.300	-53.537	-53.546	**-52.858**	**-52.873**	-53.600	-53.057
Netflix	-57.191	-57.730	-57.450	**-56.355**	-56.610	**-56.371**	-56.706
Accidents	-30.490	-29.342	-29.265	**-26.982**	-28.510	-28.351	-29.692
Retail	-11.029	-10.944	10.942	**-10.846**	-10.858	-10.858	**-10.836**
Pumsb-star	-24.743	-23.315	-23.077	**-22.405**	-22.866	-22.664	-23.702
DNA	-80.982	-81.913	-81.840	-81.211	-80.730	**-80.068**	-85.568
Kosarek	-10.894	-10.719	-10.685	-10.599	-10.690	**-10.578**	-10.615
MSWeb	-10.108	-9.833	-9.838	-9.726	-9.630	**-9.614**	-9.819
Book	-34.969	-34.306	-34.280	-34.136	-34.366	**-33.818**	-34.694
EachMovie	-52.615	-51.368	-51.388	-51.512	**-50.263**	**-50.414**	-54.513
WebKB	-158.164	-154.283	-153.911	-151.838	-151.341	**-149.851**	-157.001
Reuters-52	-85.414	-83.349	-83.361	-83.346	**-81.544**	-81.587	-86.531
BBC	-249.466	-247.301	-247.254	-248.929	**-226.359**	**-226.560**	-259.962
Ad	-19.760	-16.234	-15.885	-19.053	-13.785	**-13.595**	-16.012

Table 3: Average test log likelihoods for all algorithms.

	LearnSPN	SPN-B	SPN-BT	ID-SPN	SPN-BB	SPN-BTB	MT
LearnSPN	-	2	1	1	0	0	2
SPN-B	6	-	1	3	1	1	10
SPN-BT	6	13	-	3	2	1	11
ID-SPN	6	12	12	-	7	5	17
SPN-BB	11	18	15	11	-	3	16
SPN-BTB	11	18	18	13	14	-	16
MT	6	7	8	1	3	3	-

(a)



(b)

Fig. 2: Numbers of statistically significant victories (Wilcoxon signed rank test, p-value= 0.05) for the algorithms on the rows compared to those on columns in 2a. Average test likelihood values (y-axis) on Plants for SPN-BB and SPN-BTB at the increase of k (x-axis), and the values from the best models from LearnSPN, ID-SPN and MT are reported in 2b.

To consistently beat ID-SPN one has to give up simpler structures and make more robust ones with SPN-BB and SPN-BTB, which score 11 and 13 wins respectively. The likelihoods obtained on the datasets with fewer instances and many more features, are much higher than the ones from ID-SPN and MT. This proves how bagging can be effectively embedded as a very cheap way to strengthen sum node mixtures. In our experiments for SPN-BB and SPN-BTB, we found a monotonic behavior, resulting in $k = 50$ as the best validation parameter, while for MT this value highly varies, implying that tuning is unavoidable. Trying to balance inference complexity and likelihood accuracy one can limit k for SPN-BB and SPN-BTB to smaller values. In Figure 2b we show an example of the test likelihood gain by increasing k for both algorithms on Plants, plotting the best value achieved by LearnSPN, ID-SPN and MT as a comparison. SPN-BB and SPN-BTB, with $k = 50$, score faster learning times than ID-SPN (run with default parameters) on all but four datasets, and sometimes even than MT which has less components, e.g. for Accidents, the times in seconds are 15472, 9198, 8280, 14073 for ID-SPN, SPN-BB, SPN-BTB and MT, respectively. Additional results including p-values, times, best parameter configurations on validation and the plots for the remaining datasets are availabe in the supplementary material[5].

6 Conclusions

We focused on enhancing LearnSPN, a state-of-the-art SPN structure learner, by proposing three algorithmic variations to improve the network quality in terms of the numbers of edges, layers and parameters on the one hand and the likelihood score on the other. We showed how limiting the number of node children while

[5] `http://www.di.uniba.it/~vergari/code/spyn.html`

splitting yields simpler and deeper networks; how the introduction of Chow-Liu trees as multivariate leaf nodes leads to even more compact SPNs allowing an early interruption of the building process; and how embedding bagging into sum node mixtures can result in more robust models. An extensive empirical evaluation on standard datasets proved our enhancements to be effective, suggesting a number of future investigations: studying other ways of hybridizing SPNs with tractable multivariate distributions as building blocks; how to apply other ensembling methods like arcing and boosting to sum node splits and the possible application of input corruption techniques to make this generative model even more robust. To balance structure compactness with likelihood gains we could try to prune or graft subtrees from different bootstrapped SPNs encoding exactly or approximately equal distributions. Furthermore, applying these ideas to other tractable models structure learning algorithms could also be an opportunity.

Acknowledgements

Work supported by the project PUGLIA@SERVICE (PON02 00563 3489339) financed by the Italian Ministry of University and Research (MIUR) and by the European Commission through the project MAESTRA, grant no. ICT-2013-612944. Experiments executed on the resources made available by two projects financed by the MIUR: ReCaS (PONa3_00052) and PRISMA (PON04a2_A).

References

1. Amer, M.R., Todorovic, S.: Sum-product networks for modeling activities with stochastic structure. In: (CVPR), 2012 IEEE Conference on. pp. 1314–1321. IEEE (2012)
2. Ammar, S., Leray, P., Defourny, B., Wehenkel, L.: Probability density estimation by perturbing and combining tree structured markov networks. In: Proceedings of the 10th ECSQARU. pp. 156–167. Springer (2009)
3. Ammar, S., Leray, P., Schnitzler, F., Wehenkel, L.: Sub-quadratic markov tree mixture learning based on randomizations of the Chow-Liu algorithm. In: Proceedings of the 5th European Workshop on Probabilistic Graphical Models. pp. 17–24 (2010)
4. Bach, F.R., Jordan, M.I.: Thin junction trees. In: Advances in Neural Information Processing Systems 14. pp. 569–576. MIT Press (2001)
5. Cheng, W., Kok, S., Pham, H.V., Chieu, H.L., Chai, K.M.A.: Language modeling with sum-product networks. In: 15th Annual Conference of the International Speech Communication Association. pp. 2098–2102 (2014)
6. Choi, M.J., Tan, V.Y.F., Anandkumar, A., Willsky, A.S.: Learning latent tree graphical models. Journal of Machine Learning Research 12, 1771–1812 (2011)
7. Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory 14(3), 462–467 (1968)
8. Dennis, A., Ventura, D.: Learning the architecture of sum-product networks using clustering on varibles. In: Advances in Neural Information Processing Systems 25. pp. 2033–2041. Curran Associates, Inc. (2012)

9. Gens, R., Domingos, P.: Discriminative learning of sum-product networks. In: Advances in Neural Information Processing Systems 25. pp. 3239–3247. Curran Associates, Inc. (2012)
10. Gens, R., Domingos, P.: Learning the structure of sum-product networks. In: Proceedings of the 30th International Conference on Machine Learning. pp. 873–880. JMLR Workshop and Conference Proceedings (2013)
11. Haaren, J.V., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the 26th Conference on Artificial Intelligence. AAAI Press (2012)
12. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2009)
13. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
14. Lowd, D., Rooshenas, A.: The Libra Toolkit for Probabilistic Models. CoRR abs/1504.00110 (2015)
15. Lowd, D., Davis, J.: Learning markov network structure with decision trees. In: Proceedings of the 10th IEEE International Conference on Data Mining. pp. 334–343. IEEE Computer Society Press (2010)
16. Lowd, D., Rooshenas, A.: Learning markov networks with arithmetic circuits. In: Proceedings of the 16th International Conference on Artificial Intelligence and Statistics. JMLR Workshop Proceedings, vol. 31, pp. 406–414 (2013)
17. Martens, J., Medabalimi, V.: On the expressive efficiency of sum product networks. CoRR abs/1411.7717 (2014)
18. Meilă, M., Jordan, M.I.: Learning with mixtures of trees. Journal of Machine Learning Research 1, 1–48 (2000)
19. Peharz, R., Geiger, B., Pernkopf, F.: Greedy part-wise learning of sum-product networks. In: Machine Learning and Knowledge Discovery in Databases, LNCS, vol. 8189, pp. 612–627. Springer (2013)
20. Peharz, R., Gens, R., Domingos, P.: Learning selective sum-product networks. In: Workshop on Learning Tractable Probabilistic Models. LTPM (2014)
21. Peharz, R., Kapeller, G., Mowlaee, P., Pernkopf, F.: Modeling speech with sum-product networks: Application to bandwidth extension. In: International Conference on Acoustics, Speech and Signal Processing. pp. 3699–3703. IEEE (2014)
22. Peharz, R., Tschiatschek, S., Pernkopf, F., Domingos, P.: On theoretical properties of sum-product networks. The Journal of Machine Learning Research (2015)
23. Poon, H., Domingos, P.: Sum-product network: a new deep architecture. NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning (2011)
24. 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, LNCS, vol. 8725, pp. 630–645. Springer (2014)
25. Ridgeway, G.: Looking for lumps: Boosting and bagging for density estimation. Computational Statistics & Data Analysis 38(4), 379–392 (2002)
26. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect variable interactions. In: Proceedings of the 31st International Conference on Machine Learning. pp. 710–718. JMLR Workshop and Conference Proceedings (2014)
27. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1–2), 273–302 (1996)