Greedy Part-Wise Learning of Sum-Product Networks

Robert Peharz, Bernhard C. Geiger and Franz Pernkopf
{robert.peharz, geiger, pernkopf}@tugraz.at

Signal Processing and Speech Communication Laboratory
Graz, University of Technology

Abstract. Sum-product networks allow to model complex variable interactions while still granting efficient inference. However, most learning algorithms proposed so far are explicitly or implicitly restricted to the image domain, either by assuming variable neighborhood or by assuming that dependent variables are related by their magnitudes over the training set. In this paper, we introduce a novel algorithm, learning the structure and parameters of sum-product networks in a greedy bottom-up manner. Our algorithm iteratively merges probabilistic models of small variable scope to larger and more complex models. These merges are guided by statistical dependence test, and parameters are learned using a maximum mutual information principle. In experiments our method competes well with the existing learning algorithms for sum-product networks on the task of reconstructing covered image regions, and outperforms these when neither neighborhood nor correlations by magnitude can be assumed.

1 Introduction

Recently, a new type of probabilistic graphical models called sum-product network (SPN) was proposed [1]. Motivated by arithmetic circuits [2, 3] and aiming at expressive models, still allowing efficient inference, they represent the network polynomial of a Bayesian network [2] with a deep network architecture containing sum and product nodes. In that way, SPNs combine the domains of deep learning and probabilistic graphical models. On the one hand, SPNs can be interpreted as deep neural networks with sum and product nodes as neurons, where the sum nodes compute a *weighted* sum (with non-negative weights) of its inputs. Besides the network structure, the weights determine the network input-output function, i.e. they represent the parameters of the network. In order to allow efficient inference, the SPN should fulfill certain constraints on the network structure, namely completeness and consistency or decomposability [1]. On the other hand, SPNs represent Bayesian networks (BNs) with rich latent structure – since sum nodes can be interpreted as hidden variables being summed out – with a high degree of context-specific independence among the hidden variables. The observable variables are placed as leaves of the BN, interacting with each

other only via their latent parents. The BN interpretation opens the door for learning techniques from probabilistic graphical models, such as EM.

In [1], a learning algorithm tailored for image processing was proposed. This algorithm recursively divides an image into pairs of smaller rectangles, and learns the weights of the allotted sum nodes using a kind of hard EM, penalizing the evocation of non-zero weights[1]. This algorithm relies on locality of image regions to define the basic SPN structure, and cannot be easily applied to domains without notions of locality. In [5], a hard gradient descent method optimizing the conditional likelihood was proposed, showing convincing results on image classification tasks. The used structure is a 4-layered network on top of a image-feature generation process proposed in [6]. Therefore, also this algorithm is restricted to the image domain. Dennis and Ventura [7] use the same algorithm as in [1] for learning the SPN parameters, but propose an algorithm for finding the basic structure automatically. Their algorithm recursively splits so-called *regions*, i.e. sets of observable random variables, into pairs of smaller regions, using a heuristic application of k-means. This approach clusters variables together which have similar magnitude trends over the dataset. Therefore, also this approach is primarily useful for the image domain, and the prior knowledge about locality is implicitly given by the fact that neighboring pixels typically have similar color values. Furthermore, as the authors note, the application of k-means in this manner is rather unusual and lacks justification. Recently, after we had submitted the first version of this paper, Gens and Domingos [8] proposed a structure learning framework which is applicable to general domains: they recursively apply splits on training instances (clustering) – leading to sum nodes, and splits on variables – leading to product nodes. Further related work, which proposes to learn a tractable Markov network by using a representation as arithmetic circuit, can be found in [9].

In this paper, we propose a novel algorithm for learning SPNs, where our structure learning mechanism is well justified and can be applied to discrete random variables, continuous random variables, and mixtures thereof. Our method does not rely on explicit or implicit locality assumptions, but learns the SPN structure guided by independence tests and a maximum mutual information principle. It constructs SPNs starting from simple models over small variable scopes, and grows models over larger and larger variable scopes, building successively more expressive models (bottom-up approach). This gives an alternative to the top-down approaches proposed in [1, 7, 8], which determine the SPN structure by recursive splits of variable scopes. Therefore, our method is closer in spirit to training of deep belief networks [10–12], which also aim to extract successively more abstract features in a bottom-up manner.

The paper is organized as follows: In section 2, we introduce our notation and formally review SPNs. In section 3, we introduce our approach for learning SPNs in a bottom-up manner. In section 4, we experimentally show that our method competes well with the existing approaches in the task of image completion, and

---

[1] The claimed ℓ^0-norm penalization in [1] is not truly implemented in the provided software [4], since already evoked non-zero weights are *not* penalized any more.

outperforms them when their underlying assumptions are not met. Section 5 concludes the paper, and gives possible directions for future work.

2 Background and Notation

Assume a set of random variables (RVs) $\mathbf{X} = \{X_1, \ldots, X_D\}$, where each X_d can take values out of the set $\mathbf{val}(X_d) \subseteq \mathbb{R}$. When $\mathbf{val}(X_d)$ is finite, we say that X_d is *discrete*. In this case, inputs concerning X_d are represented using $|\mathbf{val}(X_d)|$ binary *indicator* nodes. When $\mathbf{val}(X_d)$ is infinite, inputs can be represented by *distribution* nodes (e.g. Gaussian). For the moment, let us assume that all RVs are discrete. Let $x_d^j \in \mathbf{val}(X_d)$ be the jth state of X_d, and I_d^j be the corresponding indicator node, which can assume values out of $\{0, 1\}$.

An SPN structure [1] is a connected acyclic directed graph, whose leaves are the indicator nodes for RVs $\mathbf{X}$, and all non-leaves are either *sum* or *product* nodes. A product node calculates the product of its child nodes. A sum node calculates a *weighted* sum of its child nodes, where the weights are non-negative parameters. We assume SPNs organized in layers, where sum and product layers alternate when proceeding to higher layers. The first layer is an input layer, the second is a product layer and the last (output) layer is a sum layer. All nodes are allowed to receive input only from a strictly lower layer. We call these SPNs (organized in layers and feed-forward) *layered* SPNs. In a layered SPN, we have L sum and L product layers, such that in total the SPN contains $2L + 1$ layers, where the first layer, the input layer, contains the indicator (or distribution) nodes. Let P^l be the lth product layer (i.e. the $(2l)^{\text{th}}$ layer in the SPN), and S^l the lth sum layer (i.e. the $(2l + 1)^{\text{th}}$ layer in the SPN). Let S_k^l be the kth sum node in the lth sum layer, and like-wise P_k^l for product layers. In graphical representations of SPN structures, we assume that nodes within one layer are numerated from left to right. The *parents* of some node N are denoted as $\mathrm{pa}(N)$, and the children are denoted as $\mathrm{ch}(N)$. Let the *scope* $\mathrm{sc}(N)$ of a node be a subset of the index set $\{1, \ldots, D\}$ of the RVs $\mathbf{X}$. For an indicator node I_d^j, the scope is defined as $\mathrm{sc}(I_d^j) = \{d\}$. For sum and product nodes, the scope is recursively defined as $\mathrm{sc}(N) = \bigcup_{C \in \mathrm{ch}(N)} \mathrm{sc}(C)$. Let $\mathbf{X}_{\mathrm{sc}(N)}$ be the sub-set of $\mathbf{X}$ which is indexed by $\mathrm{sc}(N)$. A *root* is a node R with $\mathrm{pa}(R) = \emptyset$. In [1, 7], only SPNs with a single root R were considered, and where $\mathrm{sc}(R) = \{1, \ldots, D\}$. In this paper, we also strive for SPNs with a single root, representing the full variable scope; however, as intermediate step, we also consider SPNs with multiple roots, and roots whose scope is a strict sub-set of $\{1, \ldots, D\}$ (see section 3). For now, let us assume SPNs with a single root R. A *sub-SPN* induced by some node N is the SPN defined by the sub-graph induced by N and all its descendants, including the corresponding parameters. N is the (single) root of its induced sub-SPN.

Let $\mathbf{e} = \left(e_1^1, \ldots, e_1^{|\mathbf{val}(X_1)|}, \ldots, e_D^1, \ldots e_D^{|\mathbf{val}(X_D)|} \right)$ denote some input to the SPN, i.e. a binary pattern for the indicator nodes. Let $N(\mathbf{e})$ denote the value of node N for input $\mathbf{e}$. For indicator nodes, $I_d^j(\mathbf{e}) = e_d^j$. To input *complete* evidence, i.e. a variable assignment $\mathbf{x} = (x_1, \ldots, x_D)$, the value indicator node

are set $e_d^j = 1$ if $x_d = x_d^j$, and $e_d^j = 0$ otherwise. When $\mathbf{e}$ encodes some complete evidence $\mathbf{x}$, we write $\mathbf{e} \sim \mathbf{x}$, and also use $N(\mathbf{x})$ for $N(\mathbf{e})$. The values for sum and product nodes are determined by an upward pass in the network. The root-value $R(\mathbf{x})$ is the *output* of an SPN for assignment $\mathbf{x}$. An SPN defines the probability distribution

$$P(\mathbf{x}) := \frac{R(\mathbf{x})}{\sum_{\mathbf{x}' \in \mathbf{val}(\mathbf{X})} R(\mathbf{x}')}. \tag{1}$$

While (1) can also be defined for standard neural networks with non-negative outputs, SPNs become truly powerful when they are *valid* [1], which means that for all state collections $\boldsymbol{\xi}_d \subseteq \mathbf{val}(X_d)$, $d \in \{1, \ldots, D\}$, it holds that

$$\sum_{x_1 \in \boldsymbol{\xi}_1} \cdots \sum_{x_D \in \boldsymbol{\xi}_D} P(x_1, \ldots, x_D) = \frac{R(\mathbf{e})}{\sum_{\mathbf{x}' \in \mathbf{val}(\mathbf{X})} R(\mathbf{x}')}, \tag{2}$$

where here $e_d^j = 1$ if $x_d^j \in \boldsymbol{\xi}_d$, and otherwise $e_d^j = 0$. In words, a valid SPN allows to efficiently marginalize over *partial* evidence by a single upward pass. The efficient marginalization in SPNs stems from a compact representation of the network polynomial of an underlying Bayesian network [1, 2]. Poon and Domingos [1] give sufficient conditions for the validity of an SPN, namely *completeness* and *consistency*:

Definition 1. *An SPN is* complete, *if for each sum node S all children of S have the same scope.*

Definition 2. *An SPN is* consistent, *if for each product node P and each two of its children $C, C' \in \mathrm{ch}(P), C \neq C'$, it holds that when an indicator I_d^j is a descendant of C, no indicator $I_d^{j'}$, $j \neq j'$, is a descendant of C'.*

Completeness and consistency are sufficient, but not necessary for validity; however, these conditions are necessary when also every sub-SPN rooted at some node N should be valid [1]. Definition 2 is somewhat cumbersome, and it is also questionable how consistency should be interpreted in the case of continuous RVs. Therefore, Poon and Domingos provide a simpler and more restrictive condition, which implies consistency, namely decomposability:

Definition 3. *An SPN is* decomposable, *if for each product node P and each two of its children $C, C' \in \mathrm{ch}(P), C \neq C'$, it holds that $\mathrm{sc}(C) \cap \mathrm{sc}(C') = \emptyset$.*

To end this section, we illustrate how continuous data can be modeled using SPNs. Following [1], one can simply use *distribution nodes* (instead of indicator nodes) for continuous RVs, e.g. with nodes returning the value of a Gaussian PDF (Gaussian nodes) as output. A simple example, showing a 4-component GMM with diagonal covariance matrix, is shown in Fig. 1. The parameters of the Gaussians, mean and variance, are considered as parameters of the Gaussian nodes in the input layer and are not shown in the figure.
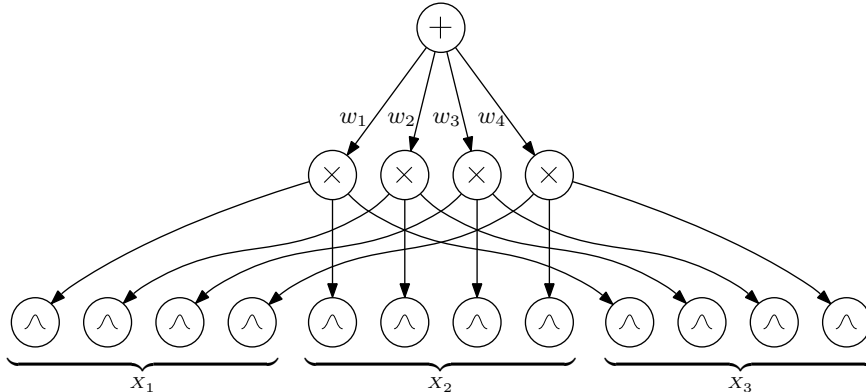
Fig. 1. SPN representing a Gaussian mixture model over three variables X_1, X_2, X_3 with 4 components and diagonal covariance matrix. The component priors (sum weights) are w_1, w_2, w_3, w_4, satisfying $w_1 + w_2 + w_3 + w_4 = 1$, $w_1, w_2, w_3, w_4 \geq 0$.

3 Greedy Part-Wise Learning of SPNs

In this section, we present our approach for part-wise learning of SPNs, where we restrict ourselves to complete and decomposable SPNs. We start with some observations serving as guidelines for our approach. First of all, an SPN with a single root R defines a probability distribution over $\mathbf{X}_{\mathrm{sc}(R)}$ according to (1). Consequently, an SPN with multiple roots $R_1, \ldots, R_r$ defines *multiple* probability distributions over $\mathbf{X}_{\mathrm{sc}(R_1)}, \ldots, \mathbf{X}_{\mathrm{sc}(R_r)}$, respectively, where in general scopes $\mathrm{sc}(R_i)$ and $\mathrm{sc}(R_j)$, $i \neq j$, can differ from each other. The representations of these distributions potentially share computational results and parameters. For example, the SPN in Fig. 2 has 4 roots, where the roots S_1^2 and S_2^2 represent two (in general distinct) distributions over the whole scope $\{X_1, X_2, X_3\}$, and roots S_3^1 and S_4^1 represent distributions over scope $\{X_2, X_3\}$. Furthermore, we see that each sub-SPN is again an SPN over the scope $\mathrm{sc}(N)$, where in the simplest case an SPN consists of a single node in the input layer. We call these single-node SPNs *atomic* SPNs. In Fig. 2, all atomic SPNs are indicator nodes[2]. However, as already noted in section 2, atomic SPNs are not restricted to be indicator nodes, but can also be distribution nodes. Even further, atomic SPNs can be probability models with arbitrarily large scopes, not only modeling single variables – they are merely not represented as SPNs in this framework, but represent some external "input"-probabilistic models. Product nodes represent distributions which assume *independence* between the variable sets indexed by their child nodes. Sum nodes represent *mixtures of distributions* represented by product nodes. We recognize that larger SPNs are simply *composite smaller* SPNs, where the basis of this inductive principle are atomic SPNs. This recursive view of SPNs is also followed in the recent paper of Gens and Domingos [8].

---

[2] An indicator node I_d^j is an SPN, which represents the distribution assigning all probability mass to the event $X_d = x_d^j$.
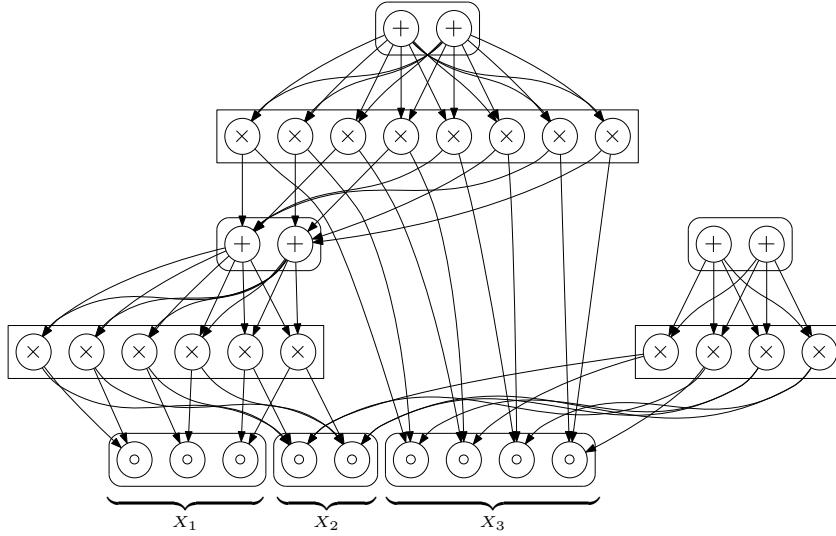
Fig. 2. Example of a multi-root SPN over variables X_1, X_2, X_3 with 3, 2 and 4 states, respectively. Nodes with ○ denote indicator nodes. Weights of sum nodes are omitted.

Using this interpretation of multi-root SPNs, we can define the *trivial* multi-root SPN which merely contains atomic nodes. Fig. 3 shows the trivial SPN for the same RVs as in Fig. 2. This SPN consists merely of the indicator nodes of
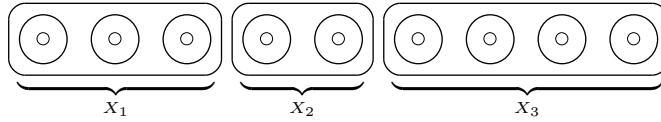

Fig. 3. Trivial multi-root SPN variables X_1, X_2, X_3 with 3, 2 and 4 states, respectively.

X_1, X_2, X_3, which are at the same time roots *and* atomic distributions. The key idea of our approach is to start from the trivial SPN containing only atomic distributions, and generate larger and larger SPNs with successively increasing scope, until we eventually obtain an SPN whose root has a scope over all RVs $\mathbf{X}$ we aim to model. In this paper, the final model will have a single root, where as intermediate step a series of multi-root SPNs is generated.

To make our approach precise, we adopt the notion of *regions*, *partitions*, and *region graphs* [7], which represents SPNs on a larger scale. The notion of a region is inspired by image modeling, i.e. when RVs $\mathbf{X}$ represent pixels of an image. However, the approach developed here is not necessarily restricted to the image domain.

Definition 4. *Given a layered, complete and decomposable SPN, the* region $\mathcal{R}$ *with scope* $\mathrm{sc}(\mathcal{R}) \subseteq \{1, \ldots, D\}$ *is the set of atomic or sum nodes, which have all the same scope* $\mathrm{sc}(\mathcal{R})$. *Regions containing only atomic nodes (e.g. indicator or distribution nodes) are called* atomic *regions. Regions containing only sum nodes are called* composite *regions.*

In this paper, we assume for simplicity that regions are either atomic or composite, i.e. they do not contain atomic nodes and sum nodes simultaneously. This restriction, however, is not essential, since we could model a region both with "external" atomic models and with composite SPNs. For some scope s, we define $\mathcal{R}(s)$ as the region with this scope, i.e. $\mathrm{sc}(\mathcal{R}(s)) = s$. While two SPN nodes can have the same scope, regions per definition have a *unique* scope. Regions can be interpreted as *dictionaries* of distributions over the same scope $\mathrm{sc}(\mathcal{R})$. In Fig. 2 and Fig. 3, regions are depicted as boxes with rounded corners. We now define partitions [7], which describe the decomposition of regions into smaller disjoint regions.

Definition 5. *Given a layered, complete and decomposable SPN, let* $\mathcal{R}_p$ *be a region and* $\boldsymbol{\mathcal{R}}_c$ *be a set of disjoint regions, with* $\mathrm{sc}(\mathcal{R}_p) = \bigcup_{\mathcal{R} \in \boldsymbol{\mathcal{R}}_c} \mathrm{sc}(\mathcal{R})$. *The partition* $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c)$ *is the set of product nodes whose parent nodes are all contained in* $\mathcal{R}_p$, *and which have exactly one child in each* $\mathcal{R} \in \boldsymbol{\mathcal{R}}_c$. *The scope of a partition* $\mathrm{sc}(\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c))$ *is defined as* $\mathrm{sc}(\mathcal{R}_p)$.

Note that since we only consider layered, complete and decomposable SPNs, each product node has to be contained in *exactly one partition*. Partitions do *not* have a unique scope, since each parent region $\mathcal{R}_p$ can be composed by several *different* partitions. We define the set of product nodes $\mathcal{P}(\mathcal{R}_p) := \bigcup \mathcal{P}(\mathcal{R}_p, \cdot)$, which contains all product nodes with same scope. In Fig. 2 and Fig. 3, partitions are depicted as boxes with edged corners. A *region graph* is defined as follows.

Definition 6. *Given a layered, complete and decomposable SPN, the* region graph $\mathcal{G}$ *of this SPN is a bipartite directed acyclic graph, with two distinct set of nodes* $\boldsymbol{\mathcal{R}}$ *and* $\boldsymbol{\mathcal{P}}$, *where* $\boldsymbol{\mathcal{R}}$ *are all* non-empty *regions and* $\boldsymbol{\mathcal{P}}$ *are all* non-empty *partitions of the SPN. Region nodes are connected only with partition nodes, and vice versa.* $\mathcal{R} \in \boldsymbol{\mathcal{R}}$ *is a parent region of* $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c)$ *if and only if* $\mathcal{R} = \mathcal{R}_p$. $\mathcal{R} \in \boldsymbol{\mathcal{R}}$ *is a child region of* $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c)$ *if and only if* $\mathcal{R} \in \boldsymbol{\mathcal{R}}_c$.

Using the notion of a region graph, we can define the *parts* of a region.

Definition 7. *Let* $\mathcal{G}$ *be the region graph of a layered, complete and decomposable SPN. The* parts *of a region* $\mathcal{R} \in \mathcal{G}$ *is the set of regions*

$$\mathrm{parts}(\mathcal{R}) := \{\mathcal{R}' | \exists \mathcal{P}(\mathcal{R}, \boldsymbol{\mathcal{R}}_c) \in \mathcal{G} : \mathcal{R}' \in \boldsymbol{\mathcal{R}}_c\}. \tag{3}$$

We are now ready to sketch our general approach, which is shown in Algorithm 1. We start with the trivial multi-root SPN containing only atomic regions. In each iteration, some disjoint regions $\boldsymbol{\mathcal{R}}_c$ are selected and *merged* into a *parent region* $\mathcal{R}_p$, generating a new partition $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c)$. Note that while in each iteration a partition $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c)$ is newly generated, i.e. it was not in the region graph

---

Algorithm 1 General Merge Learning

1: **Initialization**: Make trivial SPN and corresponding region graph $\mathcal{G}$.
2: **while** stopping criterion not met **do**
3:     **Select Merge Candidates**: Select set of $A \geq 2$ disjoint regions $\boldsymbol{\mathcal{R}}_c = \{\mathcal{R}_{c,1}, \ldots, \mathcal{R}_{c,A}\}$, with $\forall a \; \mathcal{R}_{c,a} \in \mathcal{G}$ and $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c) \notin \mathcal{G}$, where $\mathcal{R}_p = \mathcal{R}(\bigcup_{a=1}^{A} \mathrm{sc}(\mathcal{R}_{c,a}))$.
4:     **if** $\mathcal{R}_p = \emptyset$ **then**
5:         Insert K sum nodes in $\mathcal{R}_p$.
6:     **end if**
7:     **Select Set of Features** $\mathcal{F} \subseteq \mathcal{R}_{c,1} \times \cdots \times \mathcal{R}_{c,A}$.
8:     **for** $f = (N_1, \ldots, N_A) \in \mathcal{F}$ **do**
9:         Generate product node P_f.
10:        Connect P_f as parent of N_a, $a = 1, \ldots, A$.
11:        Connect P_f as child of all sum nodes in $\mathcal{R}_p$.
12:     **end for**
13:     **Learn parameters of sum nodes in** $\mathcal{R}_p$ **and its ancestor regions.**
14:     Update region graph $\mathcal{G}$.
15: **end while**

---

before, the region $\mathcal{R}_p$ might already have been generated by an earlier merge. A collection of sum nodes, one from each child region, is combined by product nodes. Here, a particular selection of child region nodes is called a *feature* (cf. step 4), where each feature corresponds to a product node. The number of generated sum nodes K will be typically $K \ll |\mathcal{F}|$, i.e. the sum nodes *represent a compression* of the generated features.

Algorithm 1 describes a general scheme for greedy part-wise learning of SPNs. Depending on the strategy of selecting the merge candidates, of selecting features, and of learning parameters, we obtain different learning algorithms. Further questions are to select K and the stopping criterion. We treat these questions in the following sub-sections, where our approach is guided by the concept of *winner* variables.

3.1 Winner Variables

For each region $\mathcal{R}$ in the region graph of some intermediate multi-root SPN, we define a winner variable

$$W_{\mathcal{R}} := \underset{i:N_i \in \mathcal{R}}{\arg\max} \, N_i(\mathbf{X}), \tag{4}$$

where we assume some arbitrary ordering of the nodes in $\mathcal{R}$. As already noted, a region can be interpreted as dictionary of distributions N_i over the same scope. $W_{\mathcal{R}}(\mathbf{x})$ is the indicator of the distribution in $\mathcal{R}$ which describes $\mathbf{X}_{\mathrm{sc}(\mathcal{R})}$ best for sample $\mathbf{x}$, since the corresponding node represents the model with *highest-likelihood*. With respect to some multi-root SPN, each variable $W_{\mathcal{R}}$ represents some *abstract* information of variables $\mathbf{X}_{\mathrm{sc}(\mathcal{R})}$. The goal in our approach is to preserve and to abstract this information, when proceeding to higher SPN levels.

3.2   Selecting Merge Candidates

Similar as in [1, 7] we set $A = 2$, i.e. we consider decompositions of a parent-region into two sub-regions. The target for selecting merge candidates is twofold: (i) we aim to find merge candidates which are "advantageous", and (ii) we want to pursue a merging strategy which yields quickly an SPN with complete scope $\{1, \ldots, D\}$, i.e. which models all variables in $\mathbf{X}$. When we neglect the latter point, our algorithm will proceed slowly, exhaust memory and overfit the SPN. To decide when a merge is advantageous, we use independence tests, motivated by BN structure learning [13]. In BNs, an edge between two variables should be present when they are statistically dependent. The major criticism about this method is the unreliability of statistical (in)dependence tests, which either causes unreliable models, or models with high inference cost. In SPNs, the variables to be modeled are not directly connected by edges, but their interaction happens over latent parents. Here the unreliability of statistical dependence tests does not harm as much as in BNs, since introducing a new, possibly spurious partition, does increase the inference cost only marginally.

In this paper, we use the Bayesian-Dirichlet independence test proposed in [14], for two winner variables $W_{\mathcal{R}'}$ and $W_{\mathcal{R}''}$:

$$\mathrm{BD}(W_{\mathcal{R}'}, W_{\mathcal{R}''}) = \left[1 + \frac{\frac{\Gamma(\gamma)}{\Gamma(\gamma+M)} \prod_{k=1}^{|\mathcal{R}|} \prod_{l=1}^{|\mathcal{R}'|} \frac{\Gamma(\gamma_{k,l}+c_{k,l})}{\Gamma(\gamma_{k,l})}}{\left(\frac{\Gamma(\alpha)}{\Gamma(\alpha+M)} \prod_{k=1}^{|\mathcal{R}|} \frac{\Gamma(\alpha_k+a_k)}{\Gamma(\alpha_k)} \right) \left(\frac{\Gamma(\beta)}{\Gamma(\beta+M)} \prod_{l=1}^{|\mathcal{R}'|} \frac{\Gamma(\beta_l+b_l)}{\Gamma(\beta_l)} \right)} \right]^{-1} \tag{5}$$

Here a_k, b_l are the number of times, counted over all training samples, where $W_{\mathcal{R}'}$ and $W_{\mathcal{R}''}$ are in their k^{th} and l^{th} states, respectively. $c_{k,l}$ is the number of times where $W_{\mathcal{R}'}$ and $W_{\mathcal{R}''}$ are jointly in their k^{th} and l^{th} states. α_k, β_l, and $\gamma_{k,l}$ are Dirichlet priors, set uniformly to 1, and $\alpha = \sum_k \alpha_k$, $\beta = \sum_l \beta_l$, $\gamma = \sum_{k,l} \gamma_{k,l}$. M is the number of samples in the training set. The lower $\mathrm{BD}(W_{\mathcal{R}'}, W_{\mathcal{R}''})$, the more the winner variables $W_{\mathcal{R}'}$, $W_{\mathcal{R}''}$ are dependent, and the more $\mathcal{R}'$ and $\mathcal{R}''$ "prefer" to merge.

To encourage a quick growing of the SPN regions, we use the scheme shown in Algorithm 2. This method maintains a set of merging candidates $\mathcal{M}$, which

---

Algorithm 2 Select Regions

1: **if** Select Regions is called the first time or $|\mathcal{M}| = 1$ **then**
2:      $\mathcal{M} \leftarrow$ set of all atomic regions.
3: **end if**
4: Select $\boldsymbol{\mathcal{R}}_c = \{\mathcal{R}', \mathcal{R}''\} \in \mathcal{M}$ which minimize $\mathrm{BD}(W_{\mathcal{R}'}, W_{\mathcal{R}''})$, s.t. $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c) \notin \mathcal{G}$.
5: $\mathcal{M} \leftarrow \mathcal{M} \setminus \boldsymbol{\mathcal{R}}_c$
6: $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{R}(\mathrm{sc}(\mathcal{R}') \cup \mathrm{sc}(\mathcal{R}''))$
7: Return $\boldsymbol{\mathcal{R}}_c = \{\mathcal{R}', \mathcal{R}''\}$.

---

is initialized with the disjoint atomic regions. In each iteration of the overall

Algorithm 1, the two most dependent regions are selected from $\mathcal{M}$ and merged to a parent region. The two selected regions are excluded from the merging candidates $\mathcal{M}$ and the parent region is inserted. In this way, the generated region graph is grown to a binary tree. When $\mathcal{M} = 1$, i.e. when the root region has been reached, we start the process again, i.e. $\mathcal{M}$ is reset to the set of all atomic regions. Then a parallel, interleaved binary tree is grown, where the constraint $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c) \notin \mathcal{G}$ guarantees that this tree is different from the first tree. This process, growing interleaved binary trees, is repeated for several iterations (*tree-growing* iterations), where the maximal number of iterations is used as a stopping criterion for Algorithm 1. Note that due to the constraint $\mathcal{P}(\mathcal{R}_p, \boldsymbol{\mathcal{R}}_c) \notin \mathcal{G}$ it can happen that in later iterations no more merging candidates can be found in step 4 of Algorithm 2. In this case, we also stop Algorithm 1.

3.3 Selecting Features and Learning Parameters

We now turn to the problem of selecting features and learning parameters. The most general approach for selecting features $\mathcal{F}$ is to take the Cartesian product of the node sets $\{\mathcal{R}_{c,a}\}_{a=1,\dots,A}$, which for $A = 2$ grows quadratically in the number of nodes in the child regions. We reduce this number and use $\mathcal{F} = \{f = (N_k, N_l) | c_{k,l} > 0\}$, where $c_{k,l}$ is defined in (5). In words, we select those features, whose corresponding product node *wins* at least once against all other potential product nodes. The sum nodes in $\mathcal{R}_p$ can be regarded as a compression of the product nodes $\mathcal{P}(\mathcal{R}_p)$ corresponding to the features $\mathcal{F}$. A natural way to perform this compression is the *information bottleneck method* [15]. Recalling Definition 7, the aim is to maximize the *mutual information* between the winner variable of $\mathcal{R}_p$ and the winner variables of $\text{parts}(\mathcal{R}_p) := \{\mathcal{R}'_1, \dots, \mathcal{R}'_{|\text{parts}(\mathcal{R}_p)|}\}$,

$$\underset{\{w_{kf}\}}{\text{maximize}} \ I(W_{\mathcal{R}'_1}, \dots, W_{\mathcal{R}'_{|\text{parts}(\mathcal{R}_p)|}}; W_{\mathcal{R}_p}) \tag{6}$$

where $\{w_{kf}\}$ are the weights of all sum nodes $S_k \in \mathcal{R}_p$, i.e.

$$S_k(\mathbf{e}) = \sum_{f : P_f \in \mathcal{P}(\mathcal{R}_p)} w_{kf} P_f(\mathbf{e}), \tag{7}$$

where $\sum_f w_{kf} = 1$, $w_{kf} \geq 0$. Since this problem can be expected to be NP-hard[3], we restrict ourselves to a greedy solution, outlined in Algorithm 3 and illustrated in Fig. 4. Our method starts with a number of sum nodes identical to the number of product nodes $\mathcal{P}(\mathcal{R}_p)$, where each product node is the child of exactly one sum node. Then we iteratively combine a pair of sum nodes to a single sum node, such that the mutual information is reduced as little as possible in each iteration. The weights of the new sum node are updated according to the maximum likelihood estimate

$$w_{kf} = \frac{\sum_{m=1}^{M} P_f(\mathbf{x}^m)}{\sum_{f' : P_{f'} \in \text{ch}(S_k)} \sum_{m=1}^{M} P_{f'}(\mathbf{x}^m)}, \tag{8}$$

---

[3] In general, the information bottleneck method is NP-hard

where $\mathbf{x}^m$ denotes the mth sample. Note that by this approach each product node becomes the child of exactly one sum node, i.e. the sum nodes have non-overlapping child sets.

Algorithm 3 Information Bottleneck Parameter Learning

1: Delete any sum nodes in $\mathcal{R}_p$.
2: For each product node P_i in $\mathcal{P}(\mathcal{R}_p)$ generate a sum node S_i.
3: Connect P_i as a child of S_i, set $w_{ii} = 1$.
4: **while** number of sum nodes $> K$ **do**
5: $\quad$ $I_{\text{best}} \leftarrow -\infty$
6: $\quad$ **for** all pairs S_i, S_j **do**
7: $\quad\quad$ Generate tentative sum node S_{tmp}.
8: $\quad\quad$ Connect $\text{ch}(S_i)$, $\text{ch}(S_j)$ as children of S_{tmp}.
9: $\quad\quad$ Set weights of S_{tmp} according to (8).
10: $\quad\quad$ $\mathcal{R}_{tmp} \leftarrow S_{\text{tmp}} \cup \mathcal{R}_p \setminus \{S_i, S_j\}$.
11: $\quad\quad$ **if** $I(W_{\mathcal{R}'_1}, \dots, W_{\mathcal{R}'_{|\text{parts}(\mathcal{R}_p)|}}; W_{\mathcal{R}\text{tmp}}) > I_{\text{best}}$ **then**
12: $\quad\quad\quad$ $I_{\text{best}} = I(W_{\mathcal{R}'_1}, \dots, W_{\mathcal{R}'_{|\text{parts}(\mathcal{R}_p)|}}; W_{\mathcal{R}\text{tmp}})$
13: $\quad\quad\quad$ $\mathcal{R}_{\text{best}} = \mathcal{R}_{\text{tmp}}$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ $\mathcal{R}_p \leftarrow \mathcal{R}_{\text{best}}$
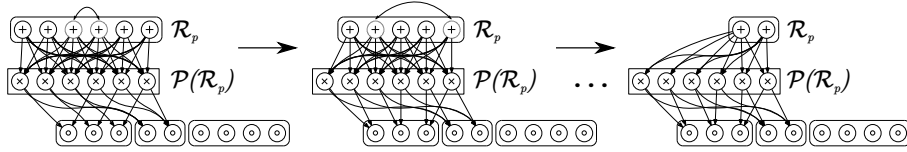17: **end while**

Fig. 4. Illustration of Information Bottleneck parameter learning ($K = 2$) for the first region merge in the trivial multi-root SPN (Fig. 3).

When we want to obtain a single-root SPN, we insert only a single sum node in the root region. We do not need to perform Algorithm 3 in this case, but merely apply (8). For non-root regions, we set K to a fixed value. However, [16] suggests a method to set K in data-driven way: Each merge will inevitably lead to a loss of information[4] w.r.t. the relevant variables $W_{\mathcal{R}'_1}, \dots, W_{\mathcal{R}'_{|\text{parts}(\mathcal{R}_p)|}}$ – if for a specific number of sum nodes a merge causes a loss which is high compared to the previous merges, this suggests that a meaningful representation has been achieved.

---

[4] This rather informal formulation about information loss can be made rigorous: see [17] for a recent effort.

4 Experiments

Poon and Domingos [1] demonstrated that SPNs achieve convincing results on the ill-posed problem of image completion, i.e. reconstructing occluded parts of face images. To this end, they trained SPNs on the ORL face image data set [18] and used MPE-inference (*most probable explanation*) to recover missing pixel values, i.e. they inferred the most likely states of the occluded pixels and the states of the latent variables associated with the sum nodes. MPE-inference is efficient in SPNs and serves as an approximation for MAP-inference, which actually is appropriate for this task. We conjecture that, although marginalization and MPE-inference is easy in SPNs, the exact MAP-problem is still intractable, since MAP is inherently harder than MPE [19]. However, proving or disproving this conjecture is future work and out of the scope of this paper.

We trained SPNs with the method of Poon and Domingos (PD) [1], the method of Dennis and Ventura (DV) [7], and our method (Merge). As in PD and DV, we model single pixels with several Gaussian nodes, where the means are set by the averages of histogram quantiles, and the standard deviation is uniformly set to 1. Using the notions introduced in this paper, this means that single pixels are used as atomic regions, containing a set of Gaussian nodes.

The ORL faces contain 64×64-pixels, which yields more than 8 million evaluations of the BD score (5) in the first iteration of Merge. Although we cache evaluations of the BD score, the computational effort is still large. Therefore, although we emphasize in this paper that our algorithm does not need prior knowledge of the problem domain, we use a similar approach as the PD algorithm, and introduce a "coarser" resolution level. We show the advantage of our algorithm, when no prior knowledge can be assumed, in the experiment following below.
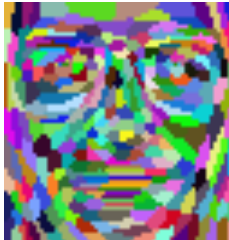

Fig. 5. Result of unsupervised segmentation of ORL faces using affinity propagation on the absolute value of correlation coefficients between pixels.

To find the coarse resolution level, we apply affinity propagation [20] on the absolute value of the correlation coefficient matrix of the pixels, calculated on the training set. This process performs an unsupervised segmentation into image patches, which is shown in Fig. 5. For each image segment, we train a multi-root SPN with 20 roots using our Merge algorithm. These 20 roots serve in turn as

atomic distributions for learning the SPN over the whole image. Within each image segment, pixels are treated as atomic regions. For each application of Merge learning, we used a single tree-growing iteration, i.e. the overall regions graph is constructed as a binary tree. In this experiment, we arbitrarily set the number of Gaussians $G = 10$ for all three algorithms. As in [1,7] we use $K = 20$ sum nodes per composite region for all algorithms. Fig. 6 shows results on the face image completion task for PD, DV, and Merge. All three algorithms show convincing results but differ in the artifacts they produce. In Table 1 we summarize objective evaluation measures for this learning task; the signal-to-noise ratios show that Merge competes well with PD and DV, although no clear preference can be shown. However, we see that while Merge achieves the lowest training likelihood, it achieves the *highest* likelihood on the test set, stating that Merge generalized best in this task.

Fig. 6. Examples of face image reconstructions using MPE-inference. Rows from top to bottom: original image, covered image, PD [1], DV [7], Merge learning (this paper).

Table 1. Evaluation measures on ORL data. Left: reconstruction SNRs for the top, bottom, left, and right halves of face images covered. Right: Log-likelihoods on training and test set, normalized by number of samples.

	top	bottom	left	right		Train	Test
PD	12.34	10.18	11.58	11.72	PD	-4287.82	-5068.97
DV	11.69	9.29	10.43	10.83	DV	-4356.41	-4673.73
Merge	12.43	9.83	10.96	11.78	Merge	-4493.46	-4667.04

In this first experiment we used prior knowledge about the problem domain, by using AP for segmenting the image into atomic regions. To demonstrate that our method does not rely on the incorporation of prior knowledge, we generated an artificial modification of the ORL data set; First, we rescale the ORL images to size 16×16, yielding 256 variables. As in [7], we permute all pixels, i.e. we destroy the neighborhood information. Next, we discretized all pixels into 10 histogram bins and permute these randomly, i.e. we destroy the relation of the RVs by their magnitude. Then we again performed the same experiment as with the large ORL data, where for each method we again used 10 Gaussian nodes per pixel, setting their means on the discretized values of bins $1 - 10$, and setting the standard deviation uniformly to 1. Table 2 shows the result for the modified ORL data. We now see a clear trend: PD, relying on locality shows consistently the worst SNRs in the image reconstruction task. DV, not relying on locality, but on similar value trends, is consistently better than PD. Merge shows consistently the best SNRs. Similarly, looking at the log-likelihoods, we see that Merge shows the best test likelihood, i.e. it generalizes best in this task.

Table 2. Evaluation measures on down-scaled and permuted ORL data. Left: reconstruction SNRs for the top, bottom, left, and right halves of face images covered. Right: Log-likelihoods on training and test set, normalized by number of samples.

	top	bottom	left	right		Train	Test
PD	15.16	8.49	12.11	10.37	PD	-442.76	-893.18
DV	15.32	9.24	12.62	10.55	DV	-506.81	-623.05
Merge	17.95	10.53	13.22	12.48	Merge	-551.97	-595.66

5 Conclusion

In this paper, we introduced a method to learn SPNs in a greedy bottom-up manner, giving an alternative to the top-down approaches proposed so far. The main principle we follow is that SPNs simply build composite and complex models out of simple and small models in a *recursive* manner. The basis of this recursive principle is given by what we call *atomic* or input distributions. We adopted the notion of *regions* and interpret them as dictionaries of distributions over the same scope. Product nodes or partitions serve as *cross-overs* of dictionaries with non-overlapping scope, corresponding to the notion of decomposability. These cross-overs yield a quickly growing number of new features or product nodes. Sum nodes of the newly created region serve as *compression* of these newly created features. This process can be seen as *abstracting* information, when proceeding to higher levels, which motivates the use of the information bottleneck method for learning sum nodes.

We showed that our method competes well with existing generative approaches to train SPNs. Furthermore, we demonstrated that our method does not rely on assumptions of the image domain, and shows the best overall performance when these are not fulfilled. In future work, we want to explore potential engineering applications for our approach, such as signal, speech and audio

processing. Furthermore, we consider the discriminative paradigm, e.g. applying maximum margin methods for classification. Finally, we want to investigate different structure and parameter learning techniques within our learning framework.

References

1. Poon, H., Domingos, P.: Sum-product networks: A new deep architecture. In: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence. (2011) 337–346
2. Darwiche, A.: A differential approach to inference in bayesian networks. ACM **50**(3) (2003) 280–305
3. Lowd, D., Domingos, P.: Learning arithmetic circuits. In: Twenty Fourth Conference on Uncertainty in Artificial Intelligence. (2008) 383–392
4. Poon, H., Domingos, P.: http://alchemy.cs.washington.edu/spn/ (2011) (online).
5. Gens, R., Domingos, P.: Discriminative learning of sum-product networks. In: Advances in Neural Information Processing Systems 25. (2012) 3248–3256
6. Coates, A., Lee, H., Ng, A.: An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. (2011)
7. Dennis, A., Ventura, D.: Learning the architecture of sum-product networks using clustering on variables. In: Advances in Neural Information Processing Systems 25. (2012) 2042–2050
8. Gens, R., Domingos, P.: Learning the structure of sum-product networks. Proceedings of ICML (2013) 873–880
9. Lowd, D., Rooshenas, A.: Learning markov networks with arithmetic circuits. Proceedings of AISTATS (2013) 406–414
10. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507
11. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems 19. (2007) 153–160
12. Bengio, Y.: Learning Deep Architectures for AI. Volume 2 of Foundations and Trends in Machine Learning. (2009)
13. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
14. Margaritis, D., Thrun, S.: A bayesian multiresolution independence test for continuous variables. In: 17th Conference on Uncertainty in Artificial Intelligence. (2001) 346–353
15. Tishby, N., Pereira, F.C., Bialek, W.: The information bottleneck method. In: Proc. Allerton Conf. on Communication, Control, and Computing. (September 1999) 368–377
16. Slonim, N., Tishby, N.: Agglomerative information bottleneck. In: Advances in Neural Information Processing Systems (NIPS), MIT Press (1999) 617–623

17. Geiger, B.C., Kubin, G.: Signal enhancement as minimization of relevant information loss. In: Proc. ITG Conf. on Systems, Communication and Coding, Munich (2013) 1–6 extended version available: `arXiv:1205.6935 [cs.IT]`.
18. Samaria, F., Harter, A.: Parameterisation of a stochastic model for human face identification. In: Proceedings of the 2nd IEEE Workshop on Applications of Computer Vision. (1994) 138–142
19. Park, J.: Map complexity results and approximation methods. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence. (2002) 338–396
20. Frey, B., Dueck, D.: Clustering by passing messages between data points. Science **315** (2007) 972–976