Learning Selective Sum-Product Networks

Robert Peharz ROBERT.PEHARZ@TUGRAZ.AT
Signal Processing and Speech Communication Lab, Graz University of Technology

Robert Gens RCG@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington

Abstract

We consider the selectivity constraint on the structure of sum-product networks (SPNs), which allows each sum node to have at most one child with non-zero output for each possible input. This allows us to find globally optimal maximum likelihood parameters in closed form. Although being a constrained class of SPNs, these models still strictly generalize classical graphical models such as Bayesian networks. Closed form parameter estimation opens the door for structure learning using a principled scoring function, trading off training likelihood and model complexity. In experiments we show that these models are easy to learn and compete well with state of the art.

1. Introduction

Probabilistic graphical models are the method of choice for reasoning under uncertainty. In classic graphical model literature the learning and inference problems are traditionally treated separate. On the learning side, researchers have used efficient and effective approximations for learning. However, this easily leads to the situation that a learned model, although it fits the data generating distributions well, the inference mechanism becomes expensive or intractable. This requires the development and application of approximate inference methods, such as Gibbs sampling and variational methods. In (Darwiche, 2003; Lowd & Domingos, 2008; Poon & Domingos, 2011) and related work, an approach of inference-aware learning emerged, i.e. to control the inference cost already during learning. At the same time, these models also introduce a more fine grained view onto learned models, implementing in a natural way context-specific independence (CSI) and related concepts. Using the *differential approach to inference* (Darwiche, 2003), inference is also conceptually easy in these models.

In this paper, we consider sum-product networks (SPNs) introduced in (Poon & Domingos, 2011). SPNs are a graphical representation of an inference machine for a probability distribution over its input variables using two types of arithmetic operations: weighted sums and products. The sum nodes can be seen as marginalized latent random variables (RVs), such that an SPN can be interpreted as a latent, potentially deep and hierarchically organized graphical model. When the SPN is constrained to be *complete* and *decomposable*, many interesting inference scenarios, such as marginalization, MPE inference and calculating marginal posteriors given evidence, can be performed linear in the network size. In this paper, we are interested into an additional constraint on the networks structure which we call *selectivity*. This notion was actually introduced in the context of arithmetic circuits (Darwiche, 2003; Lowd & Domingos, 2008) under the term *determinism*. However, we find the term "deterministic SPN" misleading, since it suggests that these SPNs partly model deterministic relations among the observable variables. This is in general not the case, so we deliberately use the term selective SPN here. In words, an SPN is selective when for each possible input and each possible parametrization, each sum node has at most one child with positive output. In that way, the generally latent RVs associated with sum nodes are deterministic functions of the observable RVs. The benefit of selectivity is that maximum likelihood parameters can be easily computed in closed form using observed frequency estimates. This opens the door for optimizing the SPN structure using a principled global scoring function, while all SPN structure learning algorithms proposed so far (Dennis & Ventura, 2012; Gens & Domingos, 2013; Peharz et al., 2013; Lee et al., 2013) optimize some *local* criterion. A key observation for structure learning is that the data likelihood decomposes into local functions of

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

data counts associated with the sum nodes. This allows us to quickly evaluate the likelihood for a structure candidate, which greatly accelerates structure learning based on greedy optimization. Both aspects, closed form ML parameters and fast likelihood computation work analogously as with Bayesian networks (BNs) over discrete RVs. However, as we will see, selective SPNs are strictly more general than BNs.

In section 2 we review SPNs. In section 3 we introduce selective SPNs and discuss ML parameter learning and structure learning. Experiments are presented in section 4 and section 5 concludes the paper. Proofs can be found in the appendix. We use letters X, Y and Z to denote RVs. Sets of RVs are denoted by bold face letters, e.g. $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$. The set of values of an RV X is denoted as $\mathbf{val}(X)$. Corresponding lower-case letter denote generic values of RVs, e.g. $x \in \mathbf{val}(X)$, $y \in \mathbf{val}(Y)$, $x_i \in \mathbf{val}(X_i)$ are generic values of X, Y and X_i, respectively. For sets of RVs $\mathbf{X} = \{X_1, \ldots, X_N\}$, we use $\mathbf{val}(\mathbf{X})$ for the set of compound states, i.e. $\mathbf{val}(\mathbf{X}) = \mathbf{val}(X_1) \times \cdots \times \mathbf{val}(X_N)$, and use $\mathbf{x}$ for generic elements of $\mathbf{val}(\mathbf{X})$. For $\mathbf{Y} \subseteq \mathbf{X}$ and $X \in \mathbf{X}$, $\mathbf{x}_\mathbf{Y}$ and $\mathbf{x}_X$ denotes the projection of $\mathbf{x}$ onto $\mathbf{Y}$ and X, respectively.

2. Sum-Product Networks

A sum-product network $\mathcal{S} = (\mathcal{G}, \boldsymbol{w})$ over RVs $\mathbf{X} = \{X_1, \ldots, X_N\}$ consists of a rooted, acyclic and directed graph $\mathcal{G}$ and a set of parameters $\boldsymbol{w}$. $\mathcal{G}$ contains three types of nodes: distributions, sums and products. To ease discussion, we introduce the following conventions: D, S and P denote distributions, sums and products, respectively. We use N, F, C and R for generic nodes, where N is an arbitrary node, F and C are used to denote parents and children of other nodes, respectively, and R is used for the root of the SPN. The set of *parents* and *children* of some node N is denoted as $\mathbf{pa}(\mathsf{N})$ and $\mathbf{ch}(\mathsf{N})$, respectively. The *descendants* of a node N, denoted as $\mathbf{desc}(\mathsf{N})$, is recursively defined as the set containing N and any child of a descendant of N. All *leaves* of $\mathcal{G}$ are distributions and all *internal* nodes are either sums or products. For each X we assume K_X distribution nodes $\mathsf{D}_{X,k}$, $k = 1, \ldots, K_X$, i.e. $\mathsf{D}_{X,k}$ represents some PMF or PDF over X. A simple case is when the leave distributions are indicators for RVs with discrete states (Poon & Domingos, 2011), i.e. $\mathsf{D}_{X,k}(X) := \mathbb{1}(X = x^k)$, where x^k is the k^{th} state of X, assuming some arbitrary but fixed ordering of the discrete states. To evaluate an SPN for input $\mathbf{x}$, the nodes are evaluated in an upwards pass from the leaves to the root: $\mathsf{D}_{X,k}(\mathbf{x}) := \mathsf{D}_{X,k}(\mathbf{x}_X)$, $\mathsf{S}(\mathbf{x}) = \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \mathsf{C}(\mathbf{x})$, and $\mathsf{P}(\mathbf{x}) = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} \mathsf{C}(\mathbf{x})$. A sum node S calculates a *weighted* sum of its children, where $w_{\mathsf{S},\mathsf{C}}$ denotes the weight associated with child C. The weights associated with S are constrained to

$$\sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} = 1, \quad w_{\mathsf{S},\mathsf{C}} \geq 0. \tag{1}$$

The set of all weights, i.e. the network parameters, is denoted as $\boldsymbol{w}$. The *scope* of a node N is defined as

$$\mathbf{sc}(\mathsf{N}) = \begin{cases} X & \text{if } \mathsf{N} = \mathsf{D}_{X,k} \text{ for some k} \\ \bigcup_{\mathsf{C} \in \mathbf{ch}(\mathsf{N})} \mathbf{sc}(\mathsf{C}) & \text{otherwise} \end{cases} \tag{2}$$

We require that an SPN is *complete* and *decomposable* (Poon & Domingos, 2011). An SPN is complete if for each sum node S

$$\mathbf{sc}(\mathsf{C}') = \mathbf{sc}(\mathsf{C}''), \;\; \forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S}), \tag{3}$$

and decomposable if for each product node P

$$\mathbf{sc}(\mathsf{C}') \cap \mathbf{sc}(\mathsf{C}'') = \emptyset, \;\; \forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P}), \mathsf{C}' \neq \mathsf{C}''. \tag{4}$$

It is easily verified that each node N in a complete and decomposable SPN represents a distribution over $\mathbf{sc}(\mathsf{N})$: a decomposable product node represents a distribution assuming independence among the scopes of its children; a complete sum node represents a mixture of its child distributions. The leaves are distributions by definition. Therefore, by induction, all nodes in an SPN represent a distribution over their scope. The distribution represented by an SPN is the distribution represented by its root R. A *sub-SPN* rooted at N, denoted as $\mathcal{S}_\mathsf{N}$, is the SPN obtained by the graph induced by $\mathbf{desc}(\mathsf{N})$, equipped with all corresponding parameters in $\mathcal{S}$. Clearly, $\mathcal{S}_\mathsf{N}$ is an SPN over $\mathbf{sc}(\mathsf{N})$. Complete and decomposable SPNs allow to perform many inference scenarios in an efficient manner, such as *marginalization*, calculate *marginal conditional distributions* and MPE inference (Poon & Domingos, 2011; Darwiche, 2003). In the remainder of this paper, all SPNs are complete and decomposable, and we simply refer to them as SPNs.

3. Selective Sum-Product Networks

Since all nodes in an SPN are distributions, it is natural to define the support $\mathbf{sup}(\mathsf{N})$, i.e. the largest subset of $\mathbf{val}(\mathbf{sc}(\mathsf{N}))$ such that $\mathcal{S}_\mathsf{N}$ has positive output for each element in this set. The support of a node depends on the structure and the parameters of the SPN. We want to introduce a modified notion of support in SPNs which does not depend on its parametrization. It is easy to see that if $\mathbf{y} \in \mathbf{sup}(\mathsf{N})$ for some parametrization $\boldsymbol{w}$, then also $\mathbf{y} \in \mathbf{sup}(\mathsf{N})$ for some other parametrization $\boldsymbol{w}'$ with *strictly* positive parameters, e.g. uniform parameters for all sum nodes. Therefore, we define the *inherent* support of N, denoted as $\mathbf{isup}(\mathsf{N})$, as the support when uniform parameters are used for all sum nodes.

We define selective sum nodes and SPNs as follows.

Definition 1. *A sum node* S *is* selective *if*

$$\mathbf{isup}(\mathsf{C}') \cap \mathbf{isup}(\mathsf{C}'') = \emptyset, \forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S}), \mathsf{C}' \neq \mathsf{C}''. \tag{5}$$

An SPN is selective *if every sum node in the SPN is selective.*

In (Lowd & Domingos, 2008), a less general notion of selectivity (actually determinism) was used, which we call *regular selectivity*. To define this notion, let $I_X(\mathsf{N}) \subseteq \{1, \ldots, K_X\}$ denote the indices of distribution nodes reachable by N, i.e. $\mathsf{D}_{X,k} \in \mathbf{desc}(\mathsf{N}) \Leftrightarrow k \in I_X(\mathsf{N})$.

Definition 2. *An SPN* $\mathcal{S}$ *is* regular selective *if*

1. *The distribution nodes have non-overlapping support, i.e.* $\forall X \in \mathbf{X} : \forall i \neq j : \mathbf{sup}(\mathsf{D}_{X,i}) \cap \mathbf{sup}(\mathsf{D}_{X,j}) = \emptyset$.
2. *Each sum node* S *is* regular selective *w.r.t. some* $X \in \mathbf{sc}(\mathsf{S})$*, denoted as* $\mathsf{S} \rightsquigarrow X$ *and defined as* $\forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S}), \mathsf{C}' \neq \mathsf{C}''$*:*
   (a) $I_X(\mathsf{C}') \cap I_X(\mathsf{C}'') = \emptyset$
   (b) $\forall Y \in \mathbf{sc}(\mathsf{S}), Y \neq X : I_Y(\mathsf{C}') = I_Y(\mathsf{C}'')$

Regular selectivity implies selectivity but not vice versa. Selectivity is an interesting property of SPNs since it allows us to find maximum likelihood (ML) parameters in closed form (see section 3.2). Furthermore, the likelihood function can be evaluated efficiently, since it decomposes into a sum of terms associated with sum nodes. These properties are shared with BNs and important for efficient structure learning of selective SPNs. However, in the next section we will see that already the restricted class of regular selective SPNs is strictly more expressive than BNs.

3.1. Regular Selective Sum-Product Networks

In Figure 1 we see canonical examples of BNs over three binary variables X, Y and Z, represented as regular selective SPNs. Note that the independency represented by a head-to-head structure can not be captured directly, but has to be enforced by putting equality constraints on the parameters connected with dashed lines in the SPN in Figure 1(d). This is not surprising since an SPN is a general purpose inference machine, comparable to the junction tree algorithm which also 'moralizes' head-to-head structures in the first step. To exploit a-priori independence represented by a head-to-head structure, one would have to use *specialized* inference methods. Arbitrary larger BNs can be constructed by using the canonical templates shown in Figure 1.

However, regular selective SPNs are in fact more expressive than classic BNs. In Figure 2(a) we see a regular selective SPN representing a BN with context-specific independence (CSI): When $X = x$ the conditional distribution

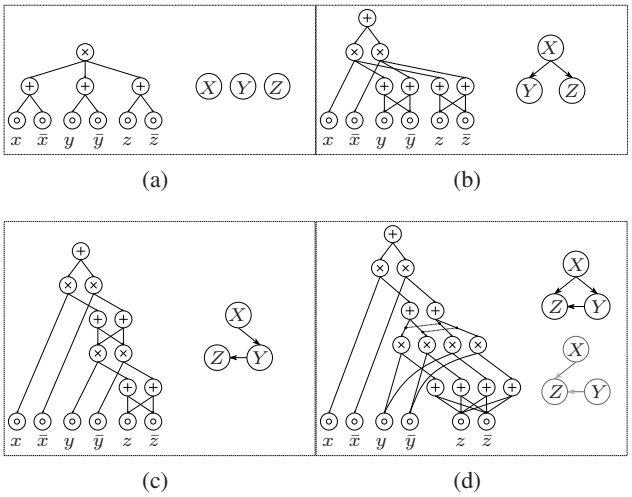

Figure 1. Example BNs represented as regular selective SPNs. Nodes with ∘ denote indicators. (a): empty BN. (b): common parent. (c): chain. (d): full BN; The a-priori independency represented by a head-to-head structure has to be simulated by equality constraints on parameters connected with dashed lines.

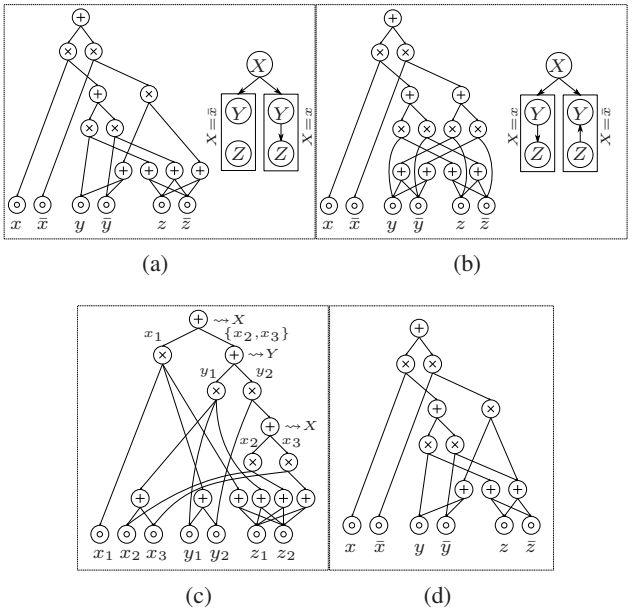

Figure 2. Advanced probabilistic models represented as regular selective SPN. (a): BN with CSI. (b): multi-net. (c): regular selective SPN with RV X having three states. The regular selective sum nodes represent a partition of the state space of their corresponding variable. (d): SPN with shared component.

is a BN $Y \rightarrow Z$; for $X = \bar{x}$, the conditional distribution is a product of marginals. Therefore, Y and Z are independent in the context $X = \bar{x}$. BNs with CSI are not new and discussed in (Boutilier et al., 1996; Chickering et al., 1997) and references therein. What is remarkable, however, is that when represented as regular selective SPNs, inference for BNs with CSI is treated exactly the same way as for classic BNs. Typically, classic inference methods for BNs can not be automatically used for BNs with CSI.

Furthermore, regular selective SPNs can represent what we call nested multi-nets. In Figure 2(b) we see a regular selective SPN which uses a BN $Y \rightarrow Z$ for the context $X = x$ and a BN $Y \leftarrow Z$ for the context $X = \bar{x}$. Since the network structure of Y and Z depend on the state of X, this SPN represents a multi-net. When considering more than 3 variables, the context-specific models are not restricted to be BNs but can them self be multi-nets. In that way one can build more or less arbitrarily nested multi-nets. We are not aware of any work on classic graphical models using this concept. Note that BNs with CSI are actually a special case of nested multi-nets. The concept of nested multi-nets is potentially very powerful, since it can be advantageous to condition on different variables in different contexts.

In (Poon & Domingos, 2011), a latent variable was associated with each sum node S. In the context of selective SPNs, the states of these variables represent events of the form $\mathbf{x}_{\mathbf{sc}(\mathsf{S})} \in \mathbf{isup}(\mathsf{C})$. Therefore, these variables are deterministic functions of the data and actually observed here. In particular for *regular* selective SPNs, they represent a partition of (a subset of) the state space of a single variable. For binary RVs, there is only one partition of the state space and the RVs associated with sum nodes simply 'imitate' the binary RVs. In Figure 2(c) we see a more advanced example of an regular selective SPN containing an RV X with three states. The topmost sum node is regular selective w.r.t. X and its two children represent the events $X \in \{x_1\}$ and $X \in \{x_2, x_3\}$. The next sum node represents a partition of the two states of RV Y. Finally, in the branch $X \in \{x_2, x_3\}, Y = y_2$ we find a sum node which is again regular selective w.r.t. X and further splits the two states x_2 and x_3. This example shows that regular selective can be interpreted like a decision graph. However, in the classical work on CPTs represented as decision graphs (Boutilier et al., 1996; Chickering et al., 1997), each variable is allowed to appear only *once* in the diagram. Furthermore, SPNs naturally allow to share components among different parts of the network. In Figure 2(d) we see a simple example adapted from Figure 2(a) where the two conditional models over $\{Y, Z\}$ share a sum representing a marginal over Z. This model uses one marginal over Z for the context $X = x, Y = y$ and another marginal for all other contexts. This implements a form of *parameter tying*, a technique which is widely used in classical graphical models. In this section we saw that regular selective SPNs, although being a restricted class of SPNs, still generalize classical graphical models. We want to stress again the remarkable fact that the concepts presented here – CSI, nested multi-nets, decision diagrams with multiple RV appearance, and component sharing – do not need any extra treatment in the inference phase, but are naturally treated within the SPN framework.

3.2. Maximum Likelihood Parameters

In (Lowd & Domingos, 2008; Lowd & Rooshenas, 2013) the learned models represent BNs and MNs, respectively, inheriting the results for parameter learning. For both, ML parameters can be obtained relatively easy in the complete data case. As we saw, selective SPNs can represent strictly more models than BNs with CSI, raising the question how to estimate its parameters.

Assume that we have a given selective SPN structure $\mathcal{G}$ and a set of completely observed i.i.d. samples $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$. We wish to maximize the likelihood:

$$L(\mathcal{D}; \boldsymbol{w}) = \prod_{n=1}^{N} \mathsf{R}(\mathbf{x}^n) \tag{6}$$

We assume that there *exists* a set of SPN parameters $\boldsymbol{w}$ such that $\mathsf{R}(\mathbf{x}^n) > 0, n = 1, \ldots, N$. Otherwise, the likelihood would be zero for all parameter sets, i.e. every parameter set would be optimal. The following definitions will help us to derive the ML parameters.

Definition 3. *A* calculation path $P_{\mathbf{x}}(\mathsf{N})$ *for node* N *and sample* $\mathbf{x}$ *is a sequence of nodes* $P_{\mathbf{x}}(\mathsf{N}) = (\mathsf{N}_1, \ldots, \mathsf{N}_J)$*, where* $\mathsf{N}_1 = \mathsf{R}$ *and* $\mathsf{N}_J = \mathsf{N}$*,* $\mathsf{N}_j(\mathbf{x}) > 0$*,* $j = 1, \ldots, J$ *and* $\mathsf{N}_j \in \mathbf{ch}(\mathsf{N}_{j-1})$*,* $j = 2, \ldots, J$*.*

Definition 4. *Let* $\mathcal{S}$ *be a complete, decomposable and selective SPN. The* calculation tree $\mathcal{T}_{\mathbf{x}}$ *for sample* $\mathbf{x}$ *is the SPN induced by all nodes for which there exists a calculation path.*

It is easy to see that for each sample $\mathbf{x}$ we have $\mathcal{S}(\mathbf{x}) = \mathcal{T}_{\mathbf{x}}(\mathbf{x})$. The following lemma will help us in our discussion.

Lemma 1. *For each complete, decomposable and selective SPN* $\mathcal{S}$ *and each sample* $\mathbf{x}$*,* $\mathcal{T}_{\mathbf{x}}$ *is a tree.*

Calculation trees allow us to write the probability of the n^{th} sample as

$$\mathsf{R}(\mathbf{x}^n) = \left(\prod_{\mathsf{S} \in \mathbf{S}} \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}^{u(n,\mathsf{S},\mathsf{C})} \right) \left(\prod_{X \in \mathbf{X}} \prod_{k=1}^{K_X} \mathsf{D}_{X,k}^{u(n,X,k)} \right), \tag{7}$$

where $\mathbf{S}$ is the set of all sum nodes in the SPN, $u(n, \mathsf{S}, \mathsf{C}) := \mathbb{1}[\mathsf{S} \in \mathcal{T}_{\mathbf{x}^n} \wedge \mathsf{C} \in \mathcal{T}_{\mathbf{x}^n}]$ and $u(n, X, k) := \mathbb{1}[\mathsf{D}_{X,k} \in \mathcal{T}_{\mathbf{x}^n}]$. Since the sample probability factorizes

over sum nodes, sum children and distribution nodes, the likelihood (6) can be written as

$$L(\mathcal{D};\boldsymbol{w}) = \left(\prod_{\mathsf{S}\in\mathbf{S}}\prod_{\mathsf{C}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}^{\#(\mathsf{S},\mathsf{C})}\right)\left(\prod_{X\in\mathbf{X}}\prod_{k=1}^{K_X}\mathsf{D}_{X,k}^{\#(X,k)}\right), \tag{8}$$

where $\#(\mathsf{S},\mathsf{C}) = \sum_{n=1}^{N} u(n,\mathsf{S},\mathsf{C})$ and $\#(X,k) = \sum_{n=1}^{N} u(n,X,k)$. Note that the products over distribution nodes D_X in (7) and (8) equal 1 in the case when all D_X are indicator nodes, i.e. when the data is discrete. Using the well known results for parameter estimation in BNs, we see that the ML parameters are given as

$$w_{\mathsf{S},\mathsf{C}} = \begin{cases} \frac{\#(\mathsf{S},\mathsf{C})}{\#(\mathsf{S})} & \text{if } \#(\mathsf{S}) \neq 0 \\ \frac{1}{|\mathbf{ch}(\mathsf{S})|} & \text{o.w.,} \end{cases} \tag{9}$$

where $\#(\mathsf{S}) = \sum_{\mathsf{C}'\in\mathbf{ch}(\mathsf{S})} \#(\mathsf{S},\mathsf{C}')$. In practice one can apply Laplace smoothing to the ML solution and also easily incorporate a Dirichlet prior on the parameters.

3.3. Structure Learning

Since parameter learning can be accomplished in closed form for selective SPNs, we can further aim at optimizing the SPN structure. To this end, we propose a similar scoring function as in (Lowd & Domingos, 2008):

$$S(\mathcal{D},\mathcal{G}) = LL(\mathcal{D},\mathcal{G}) - \lambda\, O(\mathcal{G}), \tag{10}$$

$$O(\mathcal{G}) = c_s \sum_{\mathsf{S}\in\mathbf{S}_\mathcal{G}} |\mathbf{ch}(\mathsf{S})| + c_p \sum_{\mathsf{P}\in\mathbf{P}_\mathcal{G}} |\mathbf{ch}(\mathsf{P})|, \tag{11}$$

where $LL(\mathcal{D},\mathcal{G})$ is the log-likelihood evaluated for structure $\mathcal{G}$ using ML parameters, $O(\mathcal{G})$ is the (worst case) inference cost of the SPN, $\mathbf{S}_\mathcal{G}$ and $\mathbf{P}_\mathcal{G}$ are the sets of all sum and product nodes in $\mathcal{G}$ and $\lambda, c_s, c_p \geq 0$ are trade-off parameters. The inference cost is measured by a weighted sum of the number of children of sum and product nodes. This allows us to trade-off different costs for additions and multiplications. It also indirectly penalizes the number of parameters, since the number of free parameters is $\sum_{\mathsf{S}\in\mathbf{S}_\mathcal{G}}(|\mathbf{ch}(\mathsf{S})|-1)$.

We aim to find an SPN maximizing (10). Since one can expect that this problem is inherently hard, we use greedy hill-climbing (GHC) similar as in the structure learning literature for classic graphical models (Buntine, 1991; Cooper & Herskovits, 1992; Heckerman et al., 1995), BNs with CSI (Boutilier et al., 1996; Chickering et al., 1997) and ACs (Lowd & Domingos, 2008; Lowd & Rooshenas, 2013). The basic idea is to start with an initial network and use a set of operations for transforming a network into a neighbor network. One then chooses the neighbor network which improves the score most, until no improvement is made. A desirable property of these operators is that it should be possible to transform any network in the considered class into any other network of the same class, using a finite sequence of transformations. Ideally, this set of operations should also be relatively small. The general class of selective SPNs is quite flexible and it is difficult to define a small set of such operators. We therefore restrict the search to a restricted class of models called *sum-product trees*.

Definition 5. *A* sum-product tree *(SPT) is a sum-product network where each sum and product node has at most one parent.*

Note that in SPTs the *distribution* nodes still can have multiple parents. SPTs can model arbitrary distributions, since they can represent completely connected BNs (cf. Figure 1(d)). However, they usually sacrifice model compactness and can not naturally capture Markov chains (cf. Figure 1(c)) and models with shared components (cf. Figure 2(d)). We make following assumptions concerning SPTs, not restricting generality: i) All parents of distribution nodes are sum nodes. If a distribution node has a product node as parent, we interpret the edge between them as a sum node with a single child having weight 1. Such sum nodes with a single distribution node as child are *not* counted in the scoring function (10) and removed after learning is completed. ii) Otherwise, we do not allow sum or product nodes with single children, since such networks can be simplified by short wiring and removing these nodes, trivially improving the score. iii) We do not allow chains of product nodes or chains of sum nodes which are all regular selective w.r.t. to the same RV X. Such chains can be collapsed to a single node in regular selective SPTs.

We define two operations for regular selective SPTs: **Split** (Algorithm 4) and **Merge** (Algorithm 5). Intuitively, the $\mathbf{Split}$ operation is used to resolve independence assumptions among child distributions C_1 and C_2 of a product node P. This is done by conditioning on events $X \in \mathbf{sup}(\mathsf{D}_{X,k})$ and $X \in \bigcup_{k'\in I_X(\mathsf{C}_1)\setminus\{k\}} \mathbf{sup}(\mathsf{D}_{X,k'})$. The conditional distributions of these events are represented by P' and P'' respectively. Note that when C_1 and C_2 are the only two children, the $\mathbf{ReduceNetwork}$ operation will cause that P is replaced by a sum node. $\mathbf{Merge}$ can be thought to reverse a conditioning process of a $\mathbf{Split}$ operation.

Proposition 1. *When applied to a complete, decomposable and regular selective SPT, the operators* $\mathbf{Split}$ *and* $\mathbf{Merge}$ *again yield a complete, decomposable and regular selective SPT.*

The following theorem shows that these operators are complete with respect to our model class.

Theorem 1. *Any regular selective SPT can be transformed into any other regular selective SPT by a sequence of* $\mathbf{Split}$ *and* $\mathbf{Merge}$.

As initial network we use the product of marginals (cf. Fig-

Algorithm 1 $\mathbf{CopySubSPN}(\mathsf{N})$

Require: N is a sum or product node
1: Copy all sum and product nodes in $\mathbf{desc}(\mathsf{N})$
2: Connect copied nodes as in original network
3: Connect distribution nodes to copied nodes as in original network
4: Return copy of N

Algorithm 2 $\mathbf{Dismiss}(\mathsf{N}, X, I)$

Require: $I \subset I_X(\mathsf{N})$
1: **for** $\mathsf{S} \in \mathbf{desc}(\mathsf{N})$ **do**
2: Disconnect all $\mathsf{C} \in \mathbf{ch}(\mathsf{S}) : I_X(\mathsf{C}) \subseteq I$
3: **end for**
4: Delete all unreachable nodes

Algorithm 3 $\mathbf{ReduceNetwork}$

1: $\mathbf{ShortWire}$: For all nodes N which are sums or products and which have a single child C, connect $\mathbf{pa}(\mathsf{N})$ as parents of C and delete N.
2: $\mathbf{CollapseProducts}$: Combine chains of product nodes to a single product node.
3: $\mathbf{CollapseSums}$: Combine chains of sums which are regular selective to the same X to a single sum node.

Algorithm 4 $\mathbf{Split}(\mathsf{P}, \mathsf{C}_1, \mathsf{C}_2, X, k)$

Require:
$\mathsf{C}_1, \mathsf{C}_2 \in \mathbf{ch}(\mathsf{P})$
$|I_X(\mathsf{C}_1)| \geq 2, k \in I_X(\mathsf{C}_1)$
1: Disconnect $\mathsf{C}_1, \mathsf{C}_2$ from P
2: $\mathsf{C}'_1 \leftarrow \mathbf{CopySubSPN}(\mathsf{C}_1)$
3: $\mathsf{C}'_2 \leftarrow \mathbf{CopySubSPN}(\mathsf{C}_2)$
4: $\mathbf{Dismiss}(\mathsf{C}'_1, I_X(\mathsf{C}_1) \setminus \{k\})$
5: $\mathbf{Dismiss}(\mathsf{C}_1, \{k\})$
6: Generate P' and connect C_1 and C_2 as children
7: Generate P'' and connect C'_1 and C'_2 as children
8: Generate S and connect P' and P'' as children
9: Connect S as child of P
10: $\mathbf{ReduceNetwork}$

Algorithm 5 $\mathbf{Merge}(\mathsf{S}, \mathsf{C}_1, \mathsf{C}_2)$

Require:
$|\mathbf{sc}(\mathsf{S})| \geq 2$
$\mathsf{C}_1, \mathsf{C}_2 \in \mathbf{ch}(\mathsf{S})$
$\mathsf{S} \rightsquigarrow X \Rightarrow \nexists \mathsf{S}' \in \mathbf{desc}(\mathsf{S}) : \mathsf{S}' \rightsquigarrow X$
1: Let $X : \mathsf{S} \rightsquigarrow X$
2: **for** $\mathsf{S}' \in \mathbf{desc}(\mathsf{C}_1), \mathbf{sc}(\mathsf{S}') = \{X\}$ **do**
3: **for** $k \in I_X(\mathsf{C}_2)$ **do**
4: Connect $\mathsf{D}_{X,k}$ as child of S'
5: **end for**
6: **end for**
7: Disconnect C_2 from S
8: Delete all unreachable nodes
9: $\mathbf{ReduceNetwork}$

ure 1(a)). For GHC, we score all neighboring networks which can be reached by any possible $\mathbf{Split}$ or $\mathbf{Merge}$ operation for the current network. Since the number of possible operations is still large, we need to apply some techniques to avoid unnecessary computations. Consider a potential $\mathbf{Split}$ for P, C_1 and C_2. When neither the structure of the sub-SPNs rooted at C_1 and C_2 nor the statistics $\#(\mathsf{S}, \mathsf{C})$ for the sum nodes in these sub-SPNs has changed in the last GHC iteration, the change of score remains the same in the current GHC iteration. The same holds true for $\mathbf{Merge}$ operations. The change of score can therefore be cached, which is essentially the same trick applied for GHC in standard BNs, where local changes do not affect the change of scores in other parts of the network. Furthermore, note that two operations $\mathbf{Split}(\mathsf{P}, \mathsf{C}_1, \mathsf{C}_2, X, k)$ and $\mathbf{Split}(\mathsf{P}, \mathsf{C}_1, \mathsf{C}_3, X, k)$, where $\mathsf{C}_2 \neq \mathsf{C}_3$, employs the same change to the sub-SPNs rooted at C_1 and C'_1 (cf. Algorithm 4). The score change of a $\mathbf{Split}$ operation is composite of score changes of $\mathbf{Dismiss}$ operations, which appear in several $\mathbf{Split}$ operations and can be re-used.

The most similar work to our approach is about learning arithmetic circuits representing BNs (Lowd & Domingos, 2008) and MNs (Lowd & Rooshenas, 2013). The main difference is that we use the additional $\mathbf{Merge}$ operation which allows to undo earlier $\mathbf{Split}$ operations. Selective SPNs are highly related to probabilistic decision graphs (PDGs) (Jaeger et al.). The main difference is, that PDGs, similar as BNs, are restricted to a fixed variable order. Probabilistic Sentential Decision Diagrams (PSDD) (Kisa et al., 2014) are another concept related to selective SPNs. The primary intent of PSDDs is to represent logical constraints in data domains, ruling out potentially large portions of the state space. Regular selective sum nodes can be viewed as simple decision nodes in the PSDD framework, splitting the state space according to a single variable. Combining these approaches is therefore a potential future direction.

4. Experiments

We learned regular selective SPTs using greedy hill-climbing on the 20 data sets used in (Gens & Domingos, 2013). We used fixed costs $c_s = 2$ and $c_p = 1$ for the scoring function (10), i.e. each arithmetic operation costs 1 unit. We cross-validated the trade-off factor $\lambda \in \{100, 50, 25, 10, 5, 2, 1, 0.5, 0.25, 0.125, 0.0625\}$ on the validation set. We validated the smoothing factor for sum node weights in the range $[0.0001, 2]$. For each network visited during GHC, we measure the validation likelihood and store a network whenever it improves the maximal validation likelihood LL_{best} seen so far. Additionally we used early stopping as follows. When the validation likelihood has not improved for more than 100

Table 1. Test likelihoods comparison on 20 data sets. selSPT: selective SPTs (this paper) with λ cross-validated independently. selSPT (at): selective SPTs (this paper) with λ auto-tuned. LearnSPN: (Gens & Domingos, 2013). ACMN: (Lowd & Rooshenas, 2013). WinMine: (Chickering, 2002). ID-SPN: (Rooshenas & Lowd, 2014). $\downarrow$ ($\uparrow$) means that the method has significantly worse (better) test-likelihood then selSPT (at) on a 95% confidence level using a one-sided t-test. No result for Reuters-52 using ACMN was available.

Dataset	selSPT	selSPT (at)	LearnSPN	ACMN	WinMine	ID-SPN
NLTCS	-6.036	-6.025	-6.110	$\uparrow$-5.998	-6.025	-6.020
MSNBC	$\downarrow$-6.039	-6.037	$\downarrow$-6.113	$\downarrow$-6.043	$\downarrow$-6.041	$\downarrow$-6.040
KDD	$\downarrow$-2.170	-2.163	-2.182	-2.161	$\uparrow$-2.155	$\uparrow$-2.134
Plants	$\downarrow$-13.296	-12.972	-12.977	$\uparrow$-12.803	$\uparrow$-12.647	$\uparrow$-12.537
Audio	$\downarrow$-41.486	-41.232	$\uparrow$-40.503	$\uparrow$-40.330	$\uparrow$-40.501	$\uparrow$-39.794
Jester	$\downarrow$-54.945	-54.376	$\uparrow$-53.480	$\uparrow$-53.306	$\uparrow$-53.847	$\uparrow$-52.858
Netflix	$\downarrow$-58.371	-57.978	$\uparrow$-57.328	$\uparrow$-57.216	$\uparrow$-57.025	$\uparrow$-56.355
Accidents	$\downarrow$-27.103	-26.882	$\downarrow$-30.038	$\downarrow$-27.107	$\uparrow$-26.320	$\downarrow$-26.983
Retail	-10.879	-10.882	-11.043	-10.883	-10.874	$\uparrow$-10.847
Pumsb-star	$\downarrow$-23.225	-22.656	$\downarrow$-24.781	$\downarrow$-23.554	$\uparrow$-21.721	$\uparrow$-22.405
DNA	-80.446	-80.437	$\downarrow$-82.523	$\uparrow$-80.026	$\downarrow$-80.646	$\downarrow$-81.211
Kosarek	$\downarrow$-10.893	-10.854	-10.989	-10.840	-10.834	$\uparrow$-10.599
MSWeb	$\uparrow$-9.892	-9.934	$\downarrow$-10.252	$\uparrow$-9.766	$\uparrow$-9.697	$\uparrow$-9.726
Book	-35.918	-36.006	-35.886	$\uparrow$-35.555	$\downarrow$-36.411	$\uparrow$-34.137
EachMovie	$\downarrow$-56.322	-55.724	-52.485	-55.796	$\uparrow$-54.368	$\uparrow$-51.512
WebKB	$\downarrow$-160.246	-158.523	-158.204	$\downarrow$-159.130	$\uparrow$-157.433	$\uparrow$-151.838
Reuters-52	$\downarrow$-89.127	-88.484	-85.067	N/A	$\uparrow$-87.555	$\uparrow$-83.346
20 Newsg.	$\downarrow$-159.867	-158.684	-155.925	$\downarrow$-161.130	-158.948	$\uparrow$-151.468
BBC	-259.264	-259.351	-250.687	$\uparrow$-257.102	$\uparrow$-257.861	$\uparrow$-248.929
Ad	$\uparrow$-16.271	-16.940	$\downarrow$-19.733	-16.530	$\downarrow$-18.349	$\downarrow$-19.053

GHC iterations, we start a counter c which is increased after each iteration. We stop GHC if $\frac{LL_{best}-LL_{val}}{|LL_{best}|} < 0.05 \exp\left(\frac{c \log(0.5)}{100}\right)$, where LL_{val} is the validation likelihood of the current network. For each λ we used the product of marginals (empty BN) as initial network.

We additionally used the following auto-tune scheme for lambda: We set the initial $\lambda = 100$ and start GHC. Whenever GHC converged, we set $\lambda \leftarrow 0.5\lambda$ and continue GHC. We apply the same early stopping rule as for independent cross-validation. This is essentially cross-validation of $\lambda = 100 \,.\, 2^{-r}$, $r \geq 0$, where for the r^{th} initial network the converged $(r-1)^{\text{th}}$ network is used. This approach is faster when no infrastructure for parallel cross-validation is available. Furthermore, it is more conservative with respect to model complexity: by starting with a large λ, GHC first applies **Split** operations which yield a large improvement in likelihood which come at little additional inference cost. The longest running time for GHC with auto-tuned λ was 18 hours for dataset 'Reuters-52'.

In Table 1 we see the test likelihood for selective SPTs, with independent and auto-tuned λ, LearnSPN (Gens & Domingos, 2013), ACMN (Lowd & Rooshenas, 2013), the WinMine Toolkit (Chickering, 2002) and ID-SPN (Rooshenas & Lowd, 2014). We see that the auto-tuned version of selective SPT perform 15 times (13 times significantly) better then the independently tuned version. It compares well to LearnSPN, although being a restricted class of SPNs. The results are also in range with ACMN, WinMine and ID-SPN. There are, however, advantages over these methods: ACMN trains a Markov Model and requires convex optimization of the parameters. WinMine usually trains models with expensive or even intractable inference. ID-SPN has many hyper-parameters and requires extensive cross-tuning.

5. Conclusion

In this paper we investigated an interesting constraint on SPN structure called *selectivity*. Selectivity allows us to find globally optimal ML parameters in closed form and to evaluate the likelihood function in a decomposed and efficient manner. Consequently we can strive to optimize the SPN structure using greedy optimization. In this paper, for the first time the structure of SPNs was optimized in a "principled" manner, i.e. using a global scoring function trading off training likelihood and model complexity in terms of inference cost. In the experiments we showed that this restricted class of SPNs competes well with state of the art.

In future work we want to train selective SPNs allowing sum and product nodes with more than one parent. For this purpose we need to define further graph transformations when using GHC, or use an alternative optimization technique to search the space of selective SPNs. We also want to investigate more general forms of selectivity, extending the considered model class and potentially improving results. In this paper we strived for the model with best generalization, maximizing the validation likelihood and using the complexity penalization in the scoring function a regularizer. However, the approach presented here allows to perform inference-aware learning and also to put hard constraints on the inference costs. This can be important for domains with hard computational constraints.

ACKNOWLEDGMENTS

This research was partly funded by ARO grant W911NF-08-1-0242, ONR grants N00014-13-1-0720 and N00014-12-1-0312, and AFRL contract FA8750-13-2-0019. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, ONR, AFRL, or the United States Government.

The work of Robert Peharz was supported by the Austrian Science Fund (project number P25244-N15), by the office for international relations, TU Graz, and by the Rudolf-Chaudoire donation (granted by the department of electrical engineering, TU Graz)

Proofs

Lemma 1. *For each complete, decomposable and selective SPN $\mathcal{S}$ and each sample $\mathbf{x}$, $\mathcal{T}_\mathbf{x}$ is a tree.*

Proof. For each node N in $\mathcal{T}_\mathbf{x}$ there exists a calculation path $P_\mathbf{x}(\mathsf{N})$ in $\mathcal{S}$ by definition. We have to show that this path is unique. Suppose there were two distinct calculation paths $P_\mathbf{x}(\mathsf{N}) = (\mathsf{N}_1, \ldots, \mathsf{N}_J)$ and $P'_\mathbf{x}(\mathsf{N}) = (\mathsf{N}'_1, \ldots, \mathsf{N}'_{J'})$. Since all calculation paths start at R, there must be a smallest j such that $\mathsf{N}_j = \mathsf{N}'_j$, $\mathsf{N}_{j+1} \neq \mathsf{N}'_{j+1}$ and $\mathsf{N} \in \mathbf{desc}(\mathsf{N}_{j+1}) \wedge \mathsf{N} \in \mathbf{desc}(\mathsf{N}'_{j+1})$. Such N_j does not exist: if N_j is a product node, $\mathsf{N} \in \mathbf{desc}(\mathsf{N}_{j+1}) \wedge \mathsf{N} \in \mathbf{desc}(\mathsf{N}'_{j+1})$ contradicts decomposability. If N_j is a sum node, it would contradict selectivity, since $\mathsf{N}_{j+1}(\mathbf{x}) > 0$ and $\mathsf{N}'_{j+1}(\mathbf{x}) > 0$. Therefore each calculation path is unique, i.e. $\mathcal{T}_\mathbf{x}$ is a tree. □

Theorem 1. *Any complete, decomposable and regular selective sum-product tree can be transformed into any other complete, decomposable and regular selective sum-product tree by a sequence of* $\mathbf{Split}$ *and* $\mathbf{Merge}$.

Proof. We show by induction that every complete, decomposable and regular selective sum-product tree can be transformed into the product of marginals and vice versa. For the induction basis, consider $\mathbf{X}^1 = \{X_1\}$. There is only a single model, a sum node as parent of distributions D_{X_1}, and therefore the induction basis holds. For the induction step, we assume that the theorem holds for $\mathbf{X}^{n'} = \{X_1, \ldots, X_{n'}\}, n' = 1, \ldots, N-1$, and show that it also holds for $\mathbf{X}^N = \{X_1, \ldots, X_{N-1}, X_N\}$. Consider any complete, decomposable and regular selective sum-product tree. This network can either have a sum node or a product node as root.

First consider the case that the root is a product. Since there are no chains of products allowed, the children $\mathsf{C}_1, \ldots, \mathsf{C}_K$ of the root are sum nodes. The sub-SPNs rooted at the children are over strict sub-scopes of $\mathbf{X}^n$. Therefore, by induction hypothesis, we can transform each sub-SPN into the product of marginals over these sub-scopes. Since $\mathbf{ReduceNetwork}$ is applied as part of $\mathbf{Split}$ and $\mathbf{Merge}$, this will yield the overall product of marginals of $\mathbf{X}^n$, since chains of products are collapsed into a single product node. In this way we reach the product of marginals. We now show how to reach the original network. By applying several times $\mathbf{Split}$, we can yield a network which has a product root with children $\mathsf{C}'_1, \ldots, \mathsf{C}'_K$ which have the same scopes as the original $\mathsf{C}_1, \ldots, \mathsf{C}_K$, and which are also regular selective to the same RVs. Consider C'_k and let X be such that $\mathsf{C}'_k \rightsquigarrow X$. By induction hypothesis, we can transform each child of C'_k into a sum node which is also regular selective to X, and whose children have each exactly one D_X in its descendants, i.e. it completely splits its part of the state space of X. Since these children are collapsed into C'_k, as a result C'_k itself splits the state space of X. By applying $\mathbf{Merge}$, we can combine the children of C'_k such that they represent the same state partition as C_k. By induction hypothesis, the networks rooted at the resulting children can be transformed into the original sub-networks rooted at the children of C_k. This yields the original network.

Now consider the case that the root is a sum node. Consider the set of sum nodes $\mathbf{S}_\mathsf{R}$ which are reachable from the root via a path of sum nodes. $\mathbf{S}_\mathsf{R}$ always contains a sum node S_R which only has product nodes as children $\mathsf{C}_1, \ldots, \mathsf{C}_K$. Otherwise $\mathbf{S}_\mathsf{R}$ would contain infinitely many nodes. Let X be such that $\mathsf{S}_\mathsf{R} \rightsquigarrow X$. The networks rooted at $\mathsf{C}_1, \ldots, \mathsf{C}_K$ can be transformed into products of marginals by repeatedly applying $\mathbf{Merge}$. These transformations can be undone by he same arguments as for networks with product roots. By repeatedly applying $\mathbf{Merge}$, S_R can further be turned into a product node P_R. By repeating this process for every node in $\mathbf{S}_\mathsf{R}$, we reach the product of marginals over $\mathbf{X}^n$. It remains to show that each transformation from S_R to P_R can be undone. Let k be an arbitrary element from $I_X(\mathsf{P}_\mathsf{R})$ and apply repeatedly $\mathbf{Split}$ with X and k to P_R until it turns into a sum node with two children $\mathsf{C}_1, \mathsf{C}_2$, where $I_X(\mathsf{C}_1) = \{k\}$ and $I_X(\mathsf{C}_2) = I_X(\mathsf{P}_\mathsf{R}) \setminus \{k\}$. This process can be recursively repeated for C_2 and all remaining elements in $I_X(\mathsf{S}) \setminus \{k\}$, yielding a sum node which completely splits the state space of X. Using $\mathbf{Merge}$, its children can be combined to represent the same state space partition as the original S_R. This reverts the transformation from S_R to P_R.

Using in the worst case the path over the product of marginals, any regular selective SPT can be transformed into any other regular selective SPT. □

References

Boutilier, Craig, Friedman, Nir, Goldszmidt, Moises, and Koller, Daphne. Context-Specific Independence in Bayesian Networks. In *UAI*, 1996.

Buntine, Wray. Theory Refinement on Bayesian Networks. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 52–60, 1991.

Chickering, D. M. The WinMine Toolkit. Technical Report WA MSR-TR-2002-103, 2002.

Chickering, David M., Heckerman, David, and Meek, Christopher. A Bayesian Approach to Learning Bayesian networks with local structure. In *UAI*, 1997.

Cooper, Gregory F. and Herskovits, Edward. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9:309–347, 1992.

Darwiche, A. A Differential Approach to Inference in Bayesian Networks. *ACM*, 50(3):280–305, 2003.

Dennis, A. and Ventura, D. Learning the Architecture of Sum-Product Networks Using Clustering on Variables. In *Advances in Neural Information Processing Systems 25*, pp. 2042–2050, 2012.

Gens, Robert and Domingos, Pedro. Learning the Structure of Sum-Product Networks. *Proceedings of ICML*, pp. 873–880, 2013.

Heckerman, D., Geiger, D., and Chickering, D. M. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20:197–243, 1995.

Jaeger, Manfred, Nielsen, Jens D., and Silander, Tomi. Learning probabilistic decision graphs . *International Journal of Approximate Reasoning*, 42:84–100.

Kisa, D., den Broeck, G. Van, Choi, A., and Darwiche, A. Probabilistic Sentential Decision Diagrams. In *KR*, 2014.

Lee, S.W., Heo, M.O., and Zhang, B.T. Online Incremental Structure Learning of Sum–Product Networks . In *ICONIP*, pp. 220–227, 2013.

Lowd, D. and Domingos, P. Learning Arithmetic Circuits. In *Twenty Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 383–392, 2008.

Lowd, D. and Rooshenas, A. Learning Markov Networks with Arithmetic Circuits. *Proceedings of AISTATS*, pp. 406–414, 2013.

Peharz, R., Geiger, B., and Pernkopf, F. Greedy Part-Wise Learning of Sum-Product Networks. In *ECML/PKDD*, volume 8189, pp. 612–627. Springer Berlin, 2013.

Poon, H. and Domingos, P. Sum-Product Networks: A New Deep Architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 337–346, 2011.

Rooshenas, A. and Lowd, D. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. *ICML – JMLR W&CP*, 32:710–718, 2014.