Expectation Maximization for Sum-Product Networks as Exponential Family Mixture Models

Mattia Desana MATTIA.DESANA@IWR.UNI-HEIDELBERG.DE
Christoph Schnörr SCHNOERR@MATH.UNI-HEIDELBERG.DE
Heidelberg University, Institute of Applied Mathematics, Image and Pattern Analysis Group
Im Neuenheimer Feld 205 69120 Heidelberg

Abstract

Sum-Product Networks (SPNs) are a recent class of probabilistic models which encode very large mixtures compactly by exploiting efficient reuse of computation in inference. Crucially, in SPNs the cost of inference scales linearly with the number of edges E but the encoded mixture size C can be *exponentially larger* than E. In this paper we obtain an *efficient* ($O(E)$) implementation of Expectation Maximization (EM) for SPNs which is the first to include EM updates both on mixture *coefficients* (corresponding to SPN weights) and mixture *components* (corresponding to SPN leaves). In particular, the update on *mixture components* translates to a weighted maximum likelihood problem on leaf distributions, and can be solved exactly when leaves are in the *exponential family*. This opens new application areas for SPNs, such as learning large mixtures of tree graphical models. We validate the algorithm on a synthetic but non trivial "soft-parity" distribution with 2^n modes encoded by a SPN with only $O(n)$ edges.

1. Introduction

Sum-Product Networks (SPNs, Poon and Domingos (2011)) are a class of probabilistic models which represent distributions through a Directed Acyclic Graph (DAG) with sum and products as internal nodes and tractable probability distributions as leaves. The crucial property of SPNs is that the cost of inference scales linearly with the number of edges E in the DAG, and therefore inference is always tractable - in contrast to graphical models where inference can be intractable even for relatively small graphs with cycles. Thanks to their flexibility and the ability to control the cost of inference, SPNs found practical use in a broad range of machine learning tasks including computer vision and structure learning (e.g. Poon and Domingos (2011); Gens and Domingos (2012); Rooshenas and Lowd (2014); Cheng et al. (2014); Amer and Todorovic. (2015)).

We now describe a key observation that motivates the work in this paper. Given a set of continuous or discrete variable X, it is well known that a SPN $S(X)$ with E edges represents a mixture model in the form $\sum_{c=1}^{C} \lambda_c P_c(X)$, where the mixture size C can be *exponentially larger* than E. The mixture *coefficients* λ_c are obtained, roughly speaking, by taking products of a subset of SPN weights, and the mixture *components* $P_c(X)$ are factorizations obtained as products of distributions at the SPN leaves (see e.g. fig. 1). It seems naïve to express the SPN as this large mixture, since it is intractably large. However, we note that Maximum-Likelihood problems are *much simpler* in this representation than in the SPN form. For instance, the log likelihood of the mixture ($\sum_n \ln \sum_c \lambda_c P_c(x_n)$) is *convex* if optimized with respect to coefficients λ_c only - the part determined by the SPN weights. Furthermore, one could apply Expectation Maximization - the method of choice for models with hidden variables (Dempster et al. (1977)) - to train *jointly* the weights (determining coefficients λ_c) and parameters at the *leaves* (determining components P_c), which are typically not trained in SPN learning methods.

The reason that prevents to apply mixture learning methods to SPNs is then the *intractable mixture size*. Traditional methods for learning SPNs typically ignore the mixture representation and find maximum likelihood (ML) solutions directly over the SPN weights (see Gens and Domingos (2012)). The resulting optimization is heavily non linear and typically suffers from vanishing updates in deeper nodes, which leads to use in practice variants of ML that are justified mostly by empirical results. Very recently, methods exploiting the mixture representation have been proposed, but a complete derivation of EM for SPNs including updates in the mixture components has been not derived so far (Peharz (2015); Zhao and Poupart (2016)). We face then the following problem: is it possible

to apply EM for mixture models to SPNs without explicitly expanding the encoded mixture?

Contribution. We obtain this goal by formalizing the relations between the SPN and its encoded mixture, and deriving a new result relating the mixture model to its representation as SPN in Lemma 5. The new theoretical results are then applied to implement efficiently Expectation Maximization over the mixture encoded by the SPN. This results in an efficient ($O(E)$) method for learning SPNs that is the first, to our knowledge, which:

1. Corresponds exactly to the widely used EM algorithm for mixture models: previous work used approximations thereof, as in Poon and Domingos (2011), or do not update mixture *components* but only the mixture *coefficients* (Peharz (2015); Zhao and Poupart (2016)).

2. Does not suffer from the problem of vanishing updates at deep nodes, and therefore it does not need to employ variants of ML that work well in practice but have mostly an empirical justification (see Gens and Domingos (2012)).

3. Allows to train distributions at the SPN leaves in the form of a *weighted maximum likelihood* problem. Solutions can often be found efficiently and in *close form* when leaves belong to the *exponential family* - for instance, for multivariate Gaussian and tree graphical models leaves with arbitrary structure (fig. 2).

Our algorithm can be easily adapted for iterative and parallel computation and can be extended efficiently to the case of shared weights used in SPNs with convolutional architecture (e.g. Cheng et al. (2014)). We provide an analysis of learning with EM compared to traditional methods in a synthetic but non trivial test case, whose results indicate that EM is comparable to existing SPN learning methods when only weight updates are used, and superior when updates of the leaf distribution parameters are introduced. The ability to learn efficiently complex distributions at the leaves which was not available until now allows for new applications of SPNs that should be explored in future applied work, such as learning very large mixtures of tree graphical models with EM (see fig 2). Finally, our theoretical analysis and Lemma 5 contributes to expand on the relationship between SPNs and encoded mixture models, adding to the toolbox of theoretical results on SPNs.

Structure of the Paper. In Section 2 we introduce SPNs and the related notation. In Section 3 we analyze the relationship between SPNs and the encoded mixture model. In Section 4 we derive Expectation Maximization for SPNs. In Section 5 we discuss results on an example application.

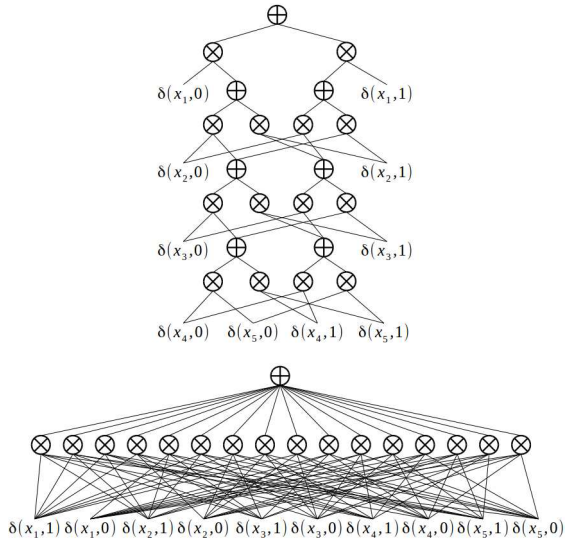

Figure 1. Parity distribution from Poon and Domingos (2011). Top: SPN representing the uniform distribution over states of five variables containing an even number of 1's. Bottom: mixture model for the same distribution.

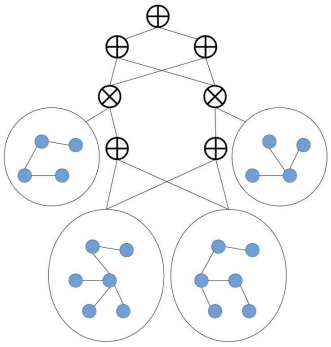

Figure 2. A SPN with tree graphical models as leaves. The SPN *weights* and the *structure* and *potentials* of the trees can be learnt *jointly* and efficiently with our derivation of EM.

2. Sum-Product Networks

We start with the definition of SPN based on Gens and Domingos (2013). Consider a set of variables X (continuous or discrete).

Definition 1. Sum-Product Network (SPN) :

1. A tractable distribution $\varphi(X)$ is a SPN $S(X)$.

2. The product $\prod_k S_k(X_k)$ of SPNs $S_k(X_k)$ is a SPN $S\left(\bigcup_k X_k\right)$ if the sets X_k are disjoint for each k.

3. The weighted sum $\sum_k w_k S_k(X)$ of SPNs $S_k(X)$ is a SPN $S(X)$ if the weights w_k are nonnegative (notice that X is in common for each SPN S_k).

By associating a node to each product, sum and tractable distribution and adding edges between an operator and its inputs a SPN can be represented as a rooted Directed Acyclic Graph (DAG) with sums and products as internal nodes and tractable distributions as leaves. This definition generalizes the SPN with indicator variables presented in Poon and Domingos (2011), since indicator variables are a special case of distribution in the form of a Dirac delta centered on a particular state of the variable, e.g. $\delta(x, 0)$ (fig. 2). A SPN is *normalized* if weights of outgoing edges of sum nodes sum to 1: $\sum_k w_k = 1$. A SPN can be chosen normalized without loss of generality (see Peharz (2015)).

Notation. We use the following notation throughout the paper. X denotes a set of variables (continuous or discrete) and x an assignment of these variables. We consider a SPN $S(X)$. $S_q(X_q)$ denotes the sub-SPN rooted at node q of S, with $X_q \subseteq X$. $S(x)$ is the value of S evaluated for assignment x (see below). $\varphi_l(X_l)$ denotes the distribution at leaf node l. In the DAG of S, $ch(q)$ and $pa(q)$ denote the children and parents of q respectively. (q, i) indicates an edge between q and its child i. If q is a sum node this edge is associated to a weight w_i^q, and for simplicity of notation we associate outgoing edges of product nodes to a weight 1. Finally, $\mathcal{E}(S)$, $\mathcal{L}(S)$ and $\mathcal{N}(S)$ denote respectively the set of edges, leaves and sum nodes in S.

Parameters. $S(X)$ is governed by two sets of *parameters*: the set of sum node weights W (terms w_i^q for each sum node edge) and the set of leaf distribution parameters θ. We write $S(X|W, \theta)$ to explicitly express this dependency. Each leaf distribution φ_l is associated to a parameter set $\theta_l \subseteq \theta$, which are for instance the mean and covariance for Gaussian leaves, and the tree structure and potentials for tree graphical model leaves (see section 4.3).

Evaluation. Assuming that the DAG of $S(X)$ has E edges, then the following quantities can be evaluated with a cost $O(E)$ Poon and Domingos (2011) :

1. $S(x)$, which is computed by first evaluating the leaf distributions with evidence set x and then evaluating sum and product nodes in inverse topological order.

2. $\frac{\partial S(x)}{\partial S_q}$, evaluated processing nodes in topological order (root to leaves) with the following recursive equation (total derivatives), noting that $\frac{\partial S(x)}{\partial S_{root}} = 1$:

$$\frac{\partial S(x)}{\partial S_q} = \sum_{k \in pa(q)} \frac{\partial S(x)}{\partial S_k} \frac{\partial S_k(x)}{\partial S_q} \tag{1}$$

$$\frac{\partial S_k(x)}{\partial S_q} = \begin{cases} w_q^k & k\ sum\ node \\ \prod_{i \in ch(k) \setminus q} S_i(x) & k\ prod.\ node \end{cases} \tag{2}$$

The evaluation of $S(x)$ also computes the quantities $S_q(x)$ for *each* node q in S (which is a SPN S_q by def. 1) as an intermediate step of the single $O(E)$ pass. The partition function can be also computed with cost $O(E)$.

3. SPNs as Large Mixture Models

It is well known that any SPN can be transformed into an equivalent SPN made of only two layers which corresponds to a mixture model as in fig. 1 (see e.g. Poon and Domingos (2011)). This interpretation was formalized in Dennis and Ventura (2015). We report it here, slightly changing the notation to accommodate for our further needs.

Definition 2. A *complete subnetwork (subnetwork* for brevity*)* σ_c of S is a SPN constructed by first including the root of S in σ_c, then processing each node q included in σ_c as follows:

1. If q is a sum node, include in σ_c one child $i \in ch(q)$ with relative weight w_i^q. Process the included child.
2. If q is a product node, include in σ_c all the children $ch(q)$. Process the included children.
3. If q is a leaf node, do nothing.

Example: fig. 3. The term "subnetwork" is taken from Gens and Domingos (2012) that use the same concept. It is easy to show that any subnetwork *is a tree* (Dennis and Ventura (2015)) . Let us call C the *number of different subnetworks* obtainable from S for different choices of included sum node children, and associate an unique index $c \in 1, 2..., C$ to each possible subnetwork $\sigma_1, \sigma_2, ..., \sigma_C$. Poon and Domingos (2011) noted that the number of subnetworks can be *exponentially larger* than the number of edges in S, e.g for the parity distribution (fig. 1).

Definition 3. For a subnetwork σ_c of $S(X|W, \theta)$ we define a *mixture coefficient* λ_c and a *mixture component* $P_c(X|\theta)$:

$$\lambda_c(W) = \prod_{(q,j) \in \mathcal{E}(\sigma_c)} w_j^q \tag{3}$$

$$P_c(X|\theta) = \prod_{l \in \mathcal{L}(\sigma_c)} \varphi_l(X_l|\theta_l) \tag{4}$$

Note that the mixture coefficients are products of all the sum weights in a subnetwork σ_c, and mixture components are factorizations obtained as products of leaves in σ_c (see fig. 3). Note also that mixture coefficients are only determined by W and mixture components only by θ.

Proposition 4. *A SPN S encodes the following mixture model:*

$$S(X|W, \theta) = \sum_{c=1}^{C} \lambda_c(W) P_c(X|\theta) \tag{5}$$

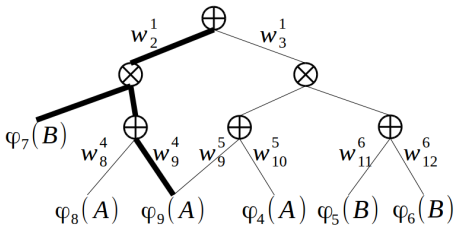

Figure 3. A SPN $S(A, B)$ in which a subnetwork σ_c of S is highlighted. This subnetwork corresponds to a mixture coefficient $\lambda_c = w_2^1 w_9^4$ and component $P_c(A, B) = \varphi_7(B)\,\varphi_9(A)$.

where C denotes the number of different subnetworks in S, and λ_c, P_c are mixture coefficients and components as in eq. 3 and 4.

Proof: see Dennis and Ventura (2015). As mentioned, the same result has been derived independently in Zhao and Poupart (2016). Notice that the size (number of components) of the encoded mixture is C and therefore a direct evaluation of the mixture has cost $O(C)$, while the evaluation of S has cost $O(E)$. Since $C \gg E$, it follows that a SPN encodes a mixture model which can be intractably large if explicitly represented. This gap is exactly what makes SPNs more expressive than "shallow" representations as mixture models. One must note, though, that not every mixture in the form $\sum_{c=1}^{C} \lambda_c P_c(X|\theta)$ can be represented by a SPN with C subnetworks, since terms λ_c and P_c must respect the factorizations eq. 3 and 4 imposed by the SPN DAG.

We now introduce a new Lemma that is crucial to derive our results, reporting it here rather than in the proofs section since it contributes to the set of analytical tools for SPNs.

Lemma 5. *Consider a SPN $S(X)$, a sum node $q \in S$ and a node $i \in ch(q)$. The following relation holds:*

$$\sum_{k:(q,i)\in\mathcal{E}(\sigma_k)} \lambda_k P_k(X) = w_i^q \frac{\partial S(X)}{\partial S_q} S_i(X) \qquad (6)$$

where $\sum_{k:(q,i)\in\mathcal{E}(\sigma_k)}$ denotes the sum over all the subnetworks σ_k of S that include the edge (q, i).

Proof: in Appendix A.1. This Lemma relates the subset of the mixture model corresponding to subnetworks σ_k that cross (q, i) and the value and derivative of the SPN nodes q and i. Note that evaluating the left-hand sum has a cost $O(C)$ (intractable) but the right-hand term has a cost $O(E)$ (tractable). Since in the derivation of EM one needs to evaluate such subsets of solution, this Lemma allows to compute the quantity of interest in a single SPN evaluation. Note also that $\sum_{k:(q,i)\in\mathcal{E}(\sigma_k)} \lambda_k P_k(X)$ corresponds to the evaluation of a non-normalized SPN which is a subset of S - e.g. the colored part in fig. 4, right.

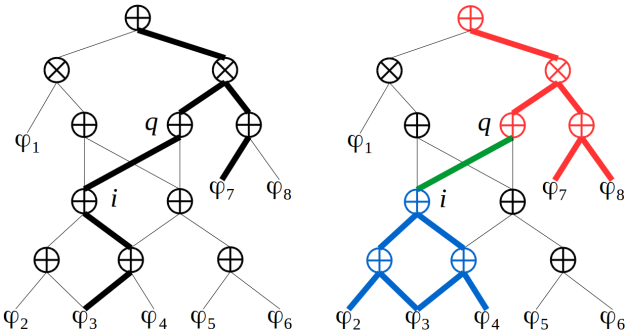

Figure 4. Visualization of Lemma 5. Left: a subnetwork σ_c crossing (q, i). Right: The colored part is the set of edges traversed by all subnetworks crossing (q, i). The blue part represents S_i and the red part covers terms appearing in $\frac{\partial S(X)}{\partial S_q}$.

4. Expectation Maximization

Expectation Maximization is an elegant and widely used method for finding maximum likelihood solutions for models with latent variables (see e.g. Murphy (2012, 11.4)). Given a distribution $P(X) = \sum_{c=1}^{C} P(X, c|\pi)$ where c are latent variables and π are the distribution parameters our objective is to maximize the log likelihood $\sum_{n=1}^{N} \ln \sum_{c=1}^{C} P(x_n, c|\pi)$ over a dataset of observations $\{x_1, x_2, ..., x_N\}$. EM proceeds by updating the parameters iteratively starting from some initial configuration π_{old}. An update step consists in finding $\pi^* = \arg\max_\pi Q(\pi|\pi_{old})$, where $Q(\pi|\pi_{old}) = \sum_{n=1}^{N} \sum_{c=1}^{C} P(c|x_n, \pi_{old}) \ln P(c, x_n|\pi)$. We want to apply EM to the mixture encoded by a SPN which is in principle intractably large. First, using the relation between SPN and encoded mixture model in Proposition 4 we identify $P(c, x_n|\pi) = \lambda_c(W) P_c(x_n|\theta)$, $P(x_n|\pi_{old}) = S(x_n|W_{old}, \theta_{old})$, and therefore $P(c|x_n, \pi_{old}) = P(c, x_n|\pi_{old})/P(x_n|\pi_{old}) = \lambda_c(W_{old}) P_c(x_n|\theta_{old}) / S(x_n|W_{old}, \theta_{old})$. Applying these substitutions and dropping the dependency on W_{old}, θ_{old} for compactness, $Q(W, \theta|W_{old}, \theta_{old})$ becomes:

$$Q(W, \theta) = \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \ln \lambda_c(W) P_c(x_n|\theta) \qquad (7)$$

In the following sections we will efficiently solve the maximization of $Q(W, \theta)$ for W and θ.

4.1. Weights Update

Simplifying $Q(W, \theta)$ through the use of Lemma 5 (Appendix A.2), one needs to maximize the following objective function:

$$W^* = \arg\max_W Q_W(W) \tag{8}$$

$$Q_W(W) = \sum_{q \in \mathcal{N}(S)} \sum_{i \in ch(q)} \beta_i^q \ln w_i^q \tag{9}$$

$$\beta_i^q = w_{i,old}^q \sum_{n=1}^{N} S^{-1}(x_n) \frac{\partial S(x_n)}{\partial S_q} S_i(x_n)$$

The evaluation of terms β_i^q, which depend only on W_{old}, θ_{old} and are therefore constants in the optimization, is the *E step* of the EM algorithm. We now maximize $Q_W(W)$ subject to $\sum_i w_i^q = 1 \forall q \in \mathcal{N}(S)$ (*M step*).

Non shared weights. Supposing that weights at each node q are disjoint parameters, then we can move the $\max$ inside the sum, obtaining separated maximizations each in the form $\arg\max_{w^q} \sum_{i \in ch(q)} \beta_i^q \log w_i^q$, where w^q is the set of weights of edges outgoing from q. Now, the same maximum is attained multiplying by the constant $k = \frac{1}{\sum_i \beta_i^q}$. Then, defining $\bar{\beta}_i^q = k\beta_i^q$, we can equivalently find $\arg\max_{w^q} \sum_{i \in ch(q)} \bar{\beta}_i^q \log w_i^q$, where $\bar{\beta}_i^q$ is positive and sums to 1 and therefore can be interpreted as a discrete distribution. This is then the maximum of the cross entropy $\arg\max_{w^q} \left(-\mathbb{H}\left(\bar{\beta}_i^q, w_i^q\right)\right)$ defined e.g. in Murphy (2012, 2.8.2). The maximum is attained for $w_i^q = \bar{\beta}_i^q$, which corresponds to the following update:

$$w_j^{q*} = \beta_j^q / \sum_i \beta_i^q \tag{10}$$

Shared weights. In some SPN applications it is necessary to share weights between different sum nodes, for instance when a convolutional architecture is used (see e.g. Cheng et al. (2014)). To keep notation simple let us consider only two nodes q_1, q_2 with shared weights - the same reasoning generalizes immediately to an arbitrary number of nodes. We suppose the nodes are constrained to have identical weights, that is $w_i^{q_1} = w_i^{q_2}$ for every child i. Calling w^q the set of shared weights, we need to find $\arg\max_{w^q} \sum_{i \in ch(q)} \beta_i^{q_1} \log w_i^{q_1} + \sum_{i \in ch(q)} \beta_i^{q_2} \log w_i^{q_2}$. Then, employing the equality constraint, we rewrite $\arg\max_{w^q} \sum_{i \in ch(q)} (\beta_i^{q_1} + \beta_i^{q_2}) \log w_i^q + const$, where the constant includes terms not depending on w^q. We now apply the same reasoning as for the non-shared weight case, multiplying by the normalization constant $k = \frac{1}{\sum_{i \in ch(q)} (\beta_i^{q_1} + \beta_i^{q_2})}$ and noting that we end up maximizing the cross entropy $-\mathbb{H}\left(k\left(\beta_i^{q_1} + \beta_i^{q_2}\right), w_i^q\right)$. Then, generalizing for an arbitrary set of nodes Q_s that *share* the same weights of q, the weight update for node q is as follows:

$$w_j^{q*} = \left(\sum_{q_s \in Q_s} \beta_j^{q_s} \right) / \sum_i \left(\sum_{q_s \in Q_s} \beta_i^{q_s} \right) \tag{11}$$

Discussion. From the mixture point of view, the weight update corresponds to the EM update for the mixture coefficients λ_c in eq. 5. In both the shared and non shared cases, the EM weight update does not suffer from vanishing updates even in deep nodes thanks to the normalization term, a problem that prevented the use of direct likelihood maximization in classic SPN training methods SPNs with deep DAGs (see Gens and Domingos (2012)). Notice that this is a weight update procedure, and therefore it is similar to existing learning algorithms for SPNs which typically train only weights. Furthermore, this has no hyperparameters in contrast to classic SPN learning methods. Finally, we note that the same EM update for *non-shared weights only* was derived with radically different approaches in Peharz (2015); Zhao and Poupart (2016). Our approach however completes the derivation of EM by updating mixture components, as described below, and extends efficiently to the case of shared weights.

4.2. Leaf Parameters Update

From the mixture point of view, the weight update corresponds to the EM update for the mixture coefficients P_c in eq. 5. Simplifying $Q(W, \theta)$ through the use of Lemma 5 (Appendix A.3), the optimization problem reduces to:

$$\theta^* = \arg\max_\theta Q_\theta(\theta) \tag{12}$$

$$Q_\theta(\theta) = \sum_{l \in \mathcal{L}(S)} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\theta_l) \tag{13}$$

$$\alpha_{ln} = S(x_n)^{-1} \frac{\partial S(x_n)}{\partial S_l} S_l(x_n)$$

The evaluation of terms α_{ln}, which are constant coefficients in the optimization since they depend only on W_{old}, θ_{old}, is the *E step* of the EM algorithm and can be seen as computing the *responsibility* that leaf distribution φ_l assigns to the n-th data point, similarly to the responsibilities appearing in EM for classical mixture models. Importantly, we note that the maximization 14 is *concave* as long as $\varphi_l(X_l|\theta)$ is concave, in which case there is an *unique global optimum*.

Non shared parameters. To the best of our knowledge, all existing SPN applications use leaf distributions with non shared parameters. Introducing the hypothesis that parameters θ_l are disjoint at each leaf l, we obtain separate maximizations in the form:

$$\theta_l^* = \arg\max_{\theta_l} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\theta_l) \tag{14}$$

This formulation can be recognized as the canonical form

of a *weighted maximum likelihood* problem. Therefore, the M step in the leaf updated translates into *a well studied problem*, where efficient solutions exist for several distributions - we consider in particular solutions for exponential families and graphical models in section 4.3, but the list of available methods goes well beyond the considered ones and therefore this step allows a great degree of *flexibility* in the choice of the leaf model, to be explored in future work.

Shared parameters. If θ_l is shared in more than one leaf the solution cannot be found analytically since a sum of logarithms appears in the optimization. However, the objective is still concave. Note that this case is not present in any existing SPN application to the best of our knowledge.

4.3. Exponential Family Leaves

A crucial property of exponential families is that eq. 12 is *concave* and therefore a *global optimum* can be reached (see e.g. Murphy (2012, 11.3.2)). Additionally, if the parameters are not shared (eq. 14) the solution is often available efficiently in *closed form*. Leaf distributions in the exponential family allow to have a wide degree of flexibility and power in modeling SPNs. The ability to learn them with EM was not investigated in applications of SPNs until now, and it should be matter of future applied work - a small example is provided in section 5. We will look in detail in the close form solution for some non-exhaustive cases of exponential family leaves.

Multivariate Gaussian Distributions. Suppose that a leaf distribution $\varphi_l(X_l)$ is a multivariate Gaussian. In this case the solution is obtained as in Murphy (2012, 11.4.2), and it is a simple adaptation of ML Gaussian fitting: $r_l = \sum_{n=1}^{N} \alpha_{ln}$, $\mu_l = \frac{\sum_{n=1}^{N} \alpha_{ln} x_n}{r_l}$, $\Sigma_l = \frac{\sum_{n=1}^{N} \alpha_{ln} (x_n - \mu_l)(x_n - \mu_l)^T}{r_l}$. It is easy to add a regularizer by adding a diagonal matrix to Σ_l, equivalent to assuming a prior distribution.

Tree graphical models. If a leaf distribution $\varphi_l(X_l)$ is a tree graphical model over discrete variables, the solution of eq. 14 can be found by constructing the tree with the Chow-Liu algorithm (Chow and Liu (1968)) adapted for weighted likelihood (see Meila and Jordan (2000)). The algorithm has a cost quadratic on the cardinality of X and allows to learn *jointly* the optimal tree structure and potentials. It can be extended to continuous variables and allows to add a regularizer term. The resulting SPN represents a very large mixture of tree graphical models, as in fig. 2. Mixture of trees are powerful architectures (see Meila and Jordan (2000)) and exploring the capabilities of this very large mixtures trained with EM should be subject of future applied work.

General Graphical Models. If $\varphi_l(X_l)$ is a graphical model (GM), eq. 14 reduces to the problem of fitting the structure *and* parameters of a graphical model through weighted maximum likelihood. This is a well known research field where results based on heuristics are available for several different structures, although the global optimum in close form is only available for trees - for instance Srebro (2003) proposes a method to find locally optimal structure and parameters for GMs with limited treewidth. The problem simplifies if the optimization is done over the model parameters keeping the GM structure fixed, although it remains NP hard (see Wainwright and Jordan (2008),Murphy (2012, 11.4.4) for a variety of methods). Remark: the idea of learning the structure of tractable GMs at the leaves has been exploited in the structure learning algorithm of Rooshenas and Lowd (2014), which however does not use EM and is based on a heuristic search.

4.4. Complexity and Implementation

The pseudocode for the EM update step is given in algorithm 2. We consider the non-shared weight case for simplicity (eq. 10), but the algorithm can easily be adapted for shared weights using eq. 11. It is easy to see that the running time is $O(NE)$: In algorithm 1, the quantities in row 4 can be computed in a single $O(E)$ pass for all the nodes in the network (see section 2), and what follows is a for loop through each edge of the network. This is repeated for every data point, and the algorithm can immediately be executed in parallel for each data point. Memory requirement is $O(E)$ for β (one element β_i^q per edge), and $O(NE)$ for α, since α_{ln} must be computed for each leaf node l and sample n. If N is too large and this value cannot be stored in memory it is often possible to further optimize the code in order to have a memory cost $O(E)$ by using *iterative maximization procedures* whenever they are available for the particular kind of leaf distribution. For instance, both the maximum likelihood Gaussian fit and Chow-Liu algorithm discussed in section 4.3 are easily adapted to update the maximization iteratively for each data point, reducing the memory cost to $O(E)$ (see Murphy (2012, 11.4.2);Meila and Jordan (2000)).

5. Example Application

The aim of this section is to test the performance of the EM updates on W and θ in a controlled setting. The weight update on W can be easily compared to classical SPN learning algorithms which train weights (Gens and Domingos (2012)). The leaf parameter update on θ is difficult to compare since, at the best of our knowledge, leaf distributions were not learned until now in any application of Sum-Product Networks. An exception is structure learning for SPNs (Rooshenas and Lowd (2014)), where leaves however are not trained but constructed as Arithmetic Cir-

Algorithm 1 ComputeAlphaAndBeta

Input: SPN S, parameters $\{W, \theta\}$, data $\{x_1, x_2, ..., x_N\}$
set $\beta_i^q = 0$ for each $q \in \mathcal{N}(S)$ and $i \in ch(q)$
for each $x_n \in \{x_1, x_2, ..., x_N\}$ **do**
  compute $S_k(x_n)$, $\frac{\partial S(x_n)}{\partial S_k}$ for each node $k \in S$
  for each sum node $q \in \mathcal{N}(S)$ **do**
    for each node $i \in ch(q)$ **do**
      $\beta_i^q \leftarrow \beta_i^q + \frac{1}{N} w_i^q S_i(x_n) \frac{\partial S(x_n)}{\partial S_q} S(x_n)^{-1}$
    end for
  end for
  for each leaf node $l \in \mathcal{L}(S)$ **do**
    $\alpha_{ln} \leftarrow S(x_n)^{-1} \frac{\partial S(x_n)}{\partial S_l} S_l(x_n)$
  end for
end for

Algorithm 2 EMstep

Input: SPN S, parameters $\{W, \theta\}$, data $\{x_1, x_2, ..., x_N\}$
$[\alpha, \beta] \leftarrow$ ComputeAlphaAndBeta$(S, W, \theta, \{x_1, ..., x_N\})$
for each sum node $q \in \mathcal{N}(S)$ **do**
  $w_i^q \leftarrow \beta_i^q / \sum_{i \in ch(q)} \beta_i^q$
end for
for each leaf node $l \in \mathcal{L}(S)$ **do**
  $\theta_l \leftarrow \arg\max_{\theta_l} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\theta_l)$
end for

cuits along with the SPN. Therefore, in order to test if learning leaves is beneficial in Maximum Likelihood we introduce a minimal but non trivial synthetic application capturing the key properties of SPNs. While the focus of this paper is theoretical and the application clearly limited, the result show promise for real applications of the algorithm in future work.

The example that we build is a continuous version of the parity distribution from Poon and Domingos (2011) (figure 1) which we call "soft parity", where the deltas at the leaves are simply substituted with Gaussian distributions with mean centered on the "on" state of each delta, and weights are not uniform but randomly assigned. This is represented by a deep SPN with $O(n)$ nodes and as many layers as the number of variables, corresponding to a mixture model with 2^n components and 2^n modes. We create a target soft parity SPN with random weights and covariances, which is sampled to obtain a training and test set. A randomly initialize network is then trained to learn these samples with several different algorithms. Comparisons of results averaged over 10 random initializations and for soft parity SPNs of different depth are shown in table 1. Our limited empirical analysis agrees well with the extensive one performed independently by Zhao and Poupart (2016) on the EM weight update, which proved to be competitive with standard SPN learning methods. Not surprisingly, when also the leaf update is introduced the results are *significantly better*. It remains to be seen if this translates into real world applications of SPNs with complex leaves learned with EM, which should be matter of future applied work.

Table 1. Log likelihood of test data for soft parity distributions of increasing depth. EM-W, θ and EM-W is our EM algorithm updating W, θ and only W respectively. SoftEM, HardEM, SoftGD and HardGD are described in (Gens and Domingos, 2012).

NVARS	EM-W, θ	EM-W	SOFTGD	SOFTEM	HARDGD	HARDEM
10	$\mathbf{-8.01}$	-10.6	-11.0	-10.6	-11.0	-10.9
20	$\mathbf{-16.1}$	-21.1	-22.0	-21.1	-22.1	-21.6
40	$\mathbf{-32.5}$	-43.4	-45.4	-43.5	-45.3	-44.7
80	$\mathbf{-66.0}$	-85.7	-89.4	-85.9	-89.3	-88.2

6. Conclusions

We presented the first derivation of Expectation Maximization (EM) for the very large mixture encoded by a SPN which includes updates on both mixture *coefficients* and *components*. The EM *weight update* does not suffer from vanishing updates and therefore can be applied without approximations to deep SPNs, in contrast with standard SPN weight learning methods (Gens and Domingos (2012)). The *leaf distribution update* is new in SPN literature and allows to efficiently learn leaves exploiting exponential family tools. This capability has never been used in SPN applications at the best of our knowledge, and should be focus of future applications such as learning very large mixtures of tree graphical models with EM. A synthetic but non trivial test case confirms the potential of this approach.

A. Appendix - Proofs

Preliminars. Consider some subnetwork σ_c of S including the edge (q, i) (fig. 4). Remembering that σ_c is a tree, we divide σ_c in three disjoint subgraphs: the edge (q, i), the tree $\sigma_{h(c)}^{d(i)}$ corresponding to "descendants" of i, and the remaining tree $\sigma_{g(c)}^{a(q)}$. Notice that $g(c)$ could be the same for two different subnetworks σ_1 and σ_2, meaning that the subtree $\sigma_{g(c)}^{a(q)}$ is in common. Similarly, the subtree $\sigma_{h(c)}^{d(i)}$ could be in common between several subnetworks.

We now observe that the the coefficient λ_c and component P_c (eqs. 3 and 4) factorize in terms corresponding to $\sigma_{g(c)}^{a(q)}$ and to $\sigma_{h(c)}^{d(i)}$ as follows: $\lambda_c = w_i^q \lambda_{h(c)}^{d(i)} \lambda_{g(c)}^{a(q)}$ and $P_c = P_{h(c)}^{d(i)} P_{g(c)}^{a(q)}$, where $\lambda_{h(c)}^{d(i)} = \prod_{(m,n) \in \mathcal{L}\left(\sigma_{h(c)}^{d(i)}\right)} w_n^m$, $P_{h(c)}^{d(i)} = \prod_{l \in \mathcal{L}\left(\sigma_{h(c)}^{d(i)}\right)} \varphi_l$ and similarly for $a(q)$. With this notation, for *each* subnetwork σ_c including (q, i) we write:

$$\lambda_c P_c = w_i^q \left(\lambda_{g(c)}^{a(q)} P_{g(c)}^{a(q)}\right)\left(\lambda_{h(c)}^{d(i)} P_{h(c)}^{d(i)}\right) \tag{15}$$

Let us now consider the sum over *all* the subnetworks σ_c of S that include (q,i). The sum can be rewritten as two nested sums, the external one over all terms $\sigma_g^{a(q)}$ (red part, fig. 4) and the internal one over all subnets $\sigma_h^{d(i)}$ (blue part, fig. 4). This is intuitively easy to grasp: we can think of the sum over all trees σ_c as first keeping the subtree $\sigma_g^{a(q)}$ fixed and varying all possible subtrees $\sigma_h^{d(i)}$ below i (inner sum), then iterating this for choice of $\sigma_g^{a(q)}$ (outer sum). Exploiting the factorization 15 we obtain the following formulation:

$$\sum_{c:(q,i)\in\mathcal{E}(\sigma_c)} \lambda_c P_c = w_i^q \sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)} \sum_{h=1}^{C_{d(i)}} \lambda_h^{d(i)} P_h^{d(i)} \tag{16}$$

where $C_{d(i)}$ and $C_{a(q)}$ denote the total number of different trees $\sigma_h^{d(i)}$ and $\sigma_g^{a(q)}$ in S.

Lemma 6. $\frac{\partial S(X)}{\partial S_q} = \sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)}$.

Proof. First let us separate the sum in eq. 5 in two sums, one over subnetworks including q and one over subnetworks not including q: $S(X) = \sum_{k:q\in\sigma_k} \lambda_k P_k + \sum_{l:q\notin\sigma_l} \lambda_l P_l$. The second sum does not involve S_q so for $\frac{\partial S(X)}{\partial S_q}$ it becomes a constant $\hat{k}$. Then, $S = \sum_{k:q\in\sigma_k} \lambda_k P_k + \hat{k}$. As in eq. 16, we divide the sum $\sum_{k:q\in\sigma_k}(\cdot)$ in two nested sums acting over disjoint terms:

$$S = \left(\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)}\right)\left(\sum_{h=1}^{C_{d(q)}} \lambda_h^{d(q)} P_h^{d(q)}\right) + \hat{k}$$

We now notice that $\sum_{k=1}^{C_{d(q)}} \lambda_k^{d(q)} P_k^{d(q)} = S_q$ by Proposition 4, since $\lambda_k^{d(q)} P_k^{d(q)}$ refer to the subtree of σ_c rooted in i and the sum is taken over all such subtrees. So we write:

$$S = \left(\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)}\right) S_q + \hat{k}$$

Taking the partial derivative leads to the result. □

A.1. Proof of Lemma 5

We start by writing the sum on the left-hand side of eq. 6 as in eq. 16. Now, first we notice that $\sum_{k=1}^{C_{d(i)}} \lambda_k^{d(i)} P_k^{d(i)}$ equals $S_i(X)$ by Proposition 4, since $\lambda_k^{d(i)} P_k^{d(i)}$ refer to the subtree of σ_c rooted in i and the sum is taken over all such subtrees. Second, $\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)} = \frac{\partial S(X)}{\partial S_q}$ for Lemma 6. Substituting in 16 we get the result. □

A.2. EM step on W

Starting from eq. 7 and collecting terms not depending on W in a constant in view of the maximization, we obtain:

$$\begin{aligned} Q(W) &= \sum_{n=1}^{N}\sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \ln \lambda_c(W) + \text{const} \\ &= \sum_{n=1}^{N}\sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{(q,i)\in\mathcal{E}(\sigma_c)} \ln w_i^q + \text{const} \end{aligned}$$

We now drop the constant and move out $\sum_{(q,i)\in\mathcal{E}(\sigma_c)}$ by introducing a delta $\delta_{(q,i),c}$ which equals 1 if $(q,i)\in\mathcal{E}(\sigma_c)$ and 0 otherwise and summing over *all* edges $\mathcal{E}(S)$:

$$\begin{aligned} Q(W) &= \sum_{n=1}^{N}\sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{(q,i)\in\mathcal{E}(S)} \ln w_i^q \delta_{(q,i),c} \\ &= \sum_{(q,i)\in\mathcal{E}(S)} \sum_{n=1}^{N} \frac{\sum_{c=1}^{C} \lambda_c P_c(x_n)\,\delta_{(q,i),c}}{S(x_n)} \ln w_i^q \\ &= \sum_{(q,i)\in\mathcal{E}(S)} \sum_{n=1}^{N} \frac{\sum_{c:(q,i)\in\mathcal{E}(\sigma_c)} \lambda_c P_c(x_n)}{S(x_n)} \ln w_i^q \end{aligned}$$

Applying Lemma 5 to $\sum_{c:(q,i)\in\mathcal{E}(\sigma_c)} \lambda_c P_c(x_n)$ we get:

$$Q(W) = \sum_{(q,i)\in\mathcal{E}(S)} \left(\sum_{n=1}^{N} \frac{w_{i,old}^q \frac{\partial S(x_n)}{\partial S_q} S_i(x_n)}{S(x_n)}\right) \ln w_i^q$$

And defining $\beta_i^q = w_{i,old}^q \sum_{n=1}^{N} S^{-1}(x_n) \frac{\partial S(x_n)}{\partial S_q} S_i(x_n)$ we can write $Q(W) = \sum_{q\in\mathcal{N}(S)} \sum_{i\in ch(q)} \beta_i^q \ln w_i^q$. □

A.3. EM step on θ

Starting from eq. 7, as in A.2 we expand $\ln P_c$ as a sum of logarithms and obtain: $Q(\theta) = \sum_{n=1}^{N}\sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{l\in\mathcal{L}(\sigma_c)} \ln \varphi_l(x_n|\theta_l) + const$. Introducing $\delta_{l,c}$ which equals 1 if $l\in\mathcal{L}(\sigma_c)$ and 0 otherwise, dropping the constant and performing the sum $\sum_{l\in\mathcal{L}(S)}$ over all leaves in S we get:

$$
\begin{aligned}
Q(\theta) &= \sum_{n=1}^{N}\sum_{c=1}^{C}\frac{\lambda_c P_c(x_n)}{S(x_n)}\sum_{l\in\mathcal{L}(S)}\ln\varphi_l(x_n|\theta_l)\,\delta_{l,c} \\
&= \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\frac{\sum_{c=1}^{C}\lambda_c P_c(x_n)}{S(x_n)}\ln\varphi_l(x_n|\theta_l)\,\delta_{l,c} \\
&= \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\frac{\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c(x_n)}{S(x_n)}\ln\varphi_l(x_n|\theta_l) \\
&= \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\alpha_{ln}\ln\varphi_l(x_n|\theta_l)
\end{aligned}
$$

Where $\alpha_{ln} = S(x_n)^{-1}\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c(x_n)$. To compute α_{ln} we notice that the term P_c in this sum always contains a factor φ_l (eq. 4), and $\varphi_l = S_l$ by def. 1. Then, writing $P_{c\setminus l} = \left(\prod_{k\in\mathcal{L}(\sigma_c)\setminus l}\varphi_k\right)$ we obtain: $\alpha_{ln} = S(x_n)^{-1} S_l\left(\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l}(x_n)\right)$. Finally, since $S = \sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c + \sum_{k:l\notin\mathcal{L}(\sigma_k)}\lambda_k P_k = S_l\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l} + \hat{k}$ (where $\hat{k}$ does not depend on S_l), taking the derivative we get $\frac{\partial S}{\partial S_l} = \sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l}$. Substituting we get: $\alpha_{ln} = S(x_n)^{-1}\frac{\partial S(X)}{\partial S_l}S_l(x_n)$. □

References

Mohamed Amer and Sinisa Todorovic. Sum product networks for activity recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI 2015)*, 2015. 1

Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Chai. Language Modeling with Sum-Product Networks. *Annual Conference of the International Speech Communication Association 15 (INTERSPEECH 2014)*, 2014. 1, 1, 4.1

C. I. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968. 4.3

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. doi: 10.2307/2984875. 1

Aaron Dennis and Dan Ventura. Greedy structure search for sum-product networks. 2015. 3, 3, 3

Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *NIPS*, pages 3248–3256, 2012. 1, 2, 3, 4.1, 5, 1, 6

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *ICML (3)*, pages 873–880, 2013. 2

Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1: 1–48, 2000. 4.3, 4.4

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029. 4, 4.1, 4.3, 4.4

Robert Peharz. Foundations of sum-product networks for probabilistic modeling. (phd thesis). *Researchgate:273000973*, 2015. 1, 1, 2, 4.1

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 337–346, 2011. 1, 1, 1, 2, 2, 3, 3, 5

Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 710–718. JMLR Workshop and Conference Proceedings, 2014. 1, 4.3, 5

Nathan Srebro. Maximum likelihood bounded tree-width markov networks. *Artif. Intell.*, 143(1):123–138, January 2003. ISSN 0004-3702. doi: 10.1016/S0004-3702(02)00360-0. 4.3

Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, January 2008. ISSN 1935-8237. doi: 10.1561/2200000001. 4.3

Han Zhao and Pascal Poupart. A unified approach for learning the parameters of sum-product networks. *ArXiv e-prints, arXiv:1601.00318*, 2016. 1, 1, 3, 4.1, 5