A Unified Approach for Learning the Parameters of Sum-Product Networks

Han Zhao
Machine Learning Department
Carnegie Mellon University
han.zhao@cs.cmu.edu

Pascal Poupart
David R. Cheriton School of Computer Science
University of Waterloo
ppoupart@uwaterloo.ca

Abstract

We present a unified approach for learning the parameters of Sum-Product networks (SPNs). We prove that any complete and decomposable SPN is equivalent to a mixture of trees where each tree corresponds to a product of univariate distributions. Based on the mixture model perspective, we characterize the objective function when learning SPNs based on the maximum likelihood estimation (MLE) principle and show that the optimization problem can be formulated as a signomial program. Both the projected gradient descent (PGD) and the exponentiated gradient (EG) in this setting can be viewed as first order approximations of the signomial program after proper transformation of the objective function. Based on the signomial program formulation, we construct two parameter learning algorithms for SPNs by using sequential monomial approximations (SMA) and the concave-convex procedure (CCCP), respectively. The two proposed methods naturally admit multiplicative updates, hence effectively avoiding the projection operation. With the help of the a unified framework, we also show an intrinsic connection between CCCP and Expectation Maximization (EM), where EM turns out to be another relaxation of the signomial program. Extensive experiments on 20 data sets demonstrate the effectiveness and efficiency of the two proposed approaches for learning SPNs. We also show that the proposed methods can improve the performance of structure learning and yield state-of-the-art results.

1 Introduction

Sum-product networks (SPNs) are new deep architectures that admit exact probabilistic inference in linear time in the size of the network [Poon and Domingos, 2011]. Different from traditional graphical models, where a compact representation does not necessarily lead to tractable inference, one of the most appealing advantages of SPNs is that they directly relate the inference complexity with the network size, thus leading to a notion of <u>inference-aware learning</u> [Peharz, 2015], where practitioners can explicitly control the inference cost during the learning of the model. This conceptual simplicity of SPNs does not sacrifice expressiveness. In fact, it has been shown recently that SPNs share the same modeling power as Bayesian networks (BNs) while at the same time being more flexible to encode context-specific independence in the network structure [Zhao et al., 2015]. Because of their flexibility and modeling power, SPNs have been widely applied in many fields of AI [Cheng et al., 2014, Amer and Todorovic, 2012, Peharz et al., 2014].

Similar to traditional graphical models, there are two main problems when learning SPNs: structure learning and parameter learning. In structure learning the goal is to infer the structure of SPNs directly from the data, and in general this process will also learn a set of parameters for the constructed SPNs (see Gens and Domingos [2013] for more details). However, the parameters obtained along with structure learning are often greedily or locally optimized in the sense that they do not necessarily maximize (minimize) the objective function of interest globally. As a result, structure learning

1

algorithms alone usually cannot fully exploit the expressiveness of the constructed model even if when an oracle provides the best structure. Hence an efficient parameter learning algorithm for SPNs can be helpful both as a fine-tuning step after structure learning and as a parameter estimation procedure by itself.

Poon and Domingos [2011], Gens and Domingos [2012] proposed both generative and discriminative learning algorithms for parameters in SPNs. At a high level, these approaches view SPNs as deep architectures and apply projected gradient descent (PGD) to optimize the data log-likelihood. There are several drawbacks associated with PGD. For example, the projection step in PGD hurts the convergence of the algorithm and it will often lead to solutions on the boundary of the feasible region. In [Poon and Domingos, 2011, Gens and Domingos, 2012], they also mentioned the possibility of applying EM algorithms to train SPNs by viewing sum nodes in SPNs as hidden variables. They presented an EM update formula without details. However, the update formula for EM given in [Poon and Domingos, 2011, Gens and Domingos, 2012] is incorrect, as pointed out by Peharz [2015].

In this paper we take a different perspective and present a unified framework for learning the parameters of SPNs. We prove that any complete and decomposable SPN is equivalent to a mixture of trees where each tree corresponds to a product of univariate distributions. Based on the mixture model perspective, we can precisely characterize the functional form of the objective function based on the network structure. We show that the optimization problem associated with learning the parameters of SPNs based on the MLE principle can be formulated as a signomial program (SP), where both PGD and the EG can be viewed as first order approximations of the signomial program after suitable transformations of the objective function. We also show that the signomial program formulation can be equivalently transformed into a difference of convex functions (DCP) formulation, where the objective function of the program can be naturally expressed as a difference of two convex functions. The DCP formulation allows us to develop two efficient optimization algorithms for learning the parameters of SPNs based on sequential monomial approximations (SMA) and the concave-convex procedure (CCCP), respectively. Both proposed approaches naturally admit multiplicative updates, hence effectively dealing with the constraints of the optimization. PGD, EG, SMA and CCCP can all be viewed as different levels of convex relaxation of the original SP. Hence the framework also provides an intuitive way to compare all four approaches. We conduct extensive experiments on 20 benchmark data sets to compare the empirical performance of PGD, EG, SMA and CCCP. Experimental results validate our theoretical analysis that CCCP is the best among all 4 approaches, showing that it converges consistently faster and with more stability than the other three methods. Furthermore, we use CCCP to boost the performance of LearnSPN [Gens and Domingos, 2013], showing that it can achieve results comparable to state-of-the-art structure learning algorithms using SPNs with much smaller sizes.

2 Background

2.1 Sum-Product Networks

To simplify the discussion of the main idea of our unified framework, we focus our attention on SPNs over boolean random variables. However, the framework presented here is general and can be easily extended into other discrete and continuous random variables. We first define the notion of network polynomial. We use $\mathbb{I}_x$ to denote an indicator variable that returns 1 when $X = x$ and 0 otherwise.

Definition 1 (Network Polynomial [Poon and Domingos, 2011])**.** Let $f(\cdot) \geq 0$ be an unnormalized probability distribution over a Boolean random vector $\mathbf{X}_{1:n}$. The network polynomial of $f(\cdot)$ is a multilinear function $\sum_{\mathbf{x}} f(\mathbf{x}) \prod_{i=1}^{n} \mathbb{I}_{\mathbf{x}_i}$ of indicator variables, where the summation is over all possible instantiations of the Boolean random vector $\mathbf{X}_{1:n}$.

Definition 2 (Sum-Product Network [Poon and Domingos, 2011])**.** A Sum-Product Network (SPN) over Boolean variables $\mathbf{X}_{1:n}$ is a rooted DAG whose leaves are the indicators $\mathbb{I}_{x_1}, \ldots, \mathbb{I}_{x_n}$ and $\mathbb{I}_{\bar{x}_1}, \ldots, \mathbb{I}_{\bar{x}_n}$ and whose internal nodes are sums and products. Each edge (v_i, v_j) emanating from a sum node v_i has a positive weight w_{ij}. The value of a product node is the product of the values of its children. The value of a sum node is $\sum_{v_j \in Ch(v_i)} w_{ij} val(v_j)$ where $Ch(v_i)$ are the children of v_i and $val(v_j)$ is the value of node v_j. The value of an SPN is the value of its root.

The scope of a node in an SPN is defined as the set of variables that have indicators among the node's descendants. For any node v in an SPN, if v is a terminal node, say, an indicator variable over X, then $\text{scope}(v) = \{X\}$, else $\text{scope}(v) = \bigcup_{\tilde{v} \in Ch(v)} \text{scope}(\tilde{v})$. Poon and Domingos [2011] also define the following properties of an SPN:

Definition 3 (Complete). An SPN is complete iff each sum node has children with the same scope.

Definition 4 (Decomposable). An SPN is decomposable iff for every product node v, $\text{scope}(v_i) \bigcap \text{scope}(v_j) = \varnothing$ where $v_i, v_j \in Ch(v), i \neq j$.

In this paper, we focus on complete and decomposable SPNs. For a complete and decomposable SPN $\mathcal{S}$, each node v in $\mathcal{S}$ defines a network polynomial $f_v(\cdot)$ which corresponds to the sub-SPN (subgraph) rooted at v. The network polynomial of $\mathcal{S}$, denoted by $f_{\mathcal{S}}$, is the network polynomial defined by the root of $\mathcal{S}$, which can be computed recursively from its children. The probability distribution induced by an SPN $\mathcal{S}$ is defined as $\Pr_{\mathcal{S}}(\mathbf{x}) \triangleq \frac{f_{\mathcal{S}}(\mathbf{x})}{\sum_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})}$.

2.2 Signomial Programming (SP)

Before introducing SP, we first introduce geometric programming (GP), which is a strict subclass of SP. A monomial is defined as a function $h : \mathbb{R}_{++} \mapsto \mathbb{R}$: $h(\mathbf{x}) = dx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$, where the domain is restricted to be the positive orthant, the coefficient d is positive and the exponents $a_i \in \mathbb{R}, \forall i$. A posynomial is a sum of monomials: $g(\mathbf{x}) = \sum_{k=1}^{K} d_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}$. One of the key properties of posynomials is its positivity, which allows us to transform it into the log-domain. A GP in standard form has the following form:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{K_0} d_{0k} \prod_{t=1}^{n} \mathbf{x}_t^{a_{0kt}} \\
\text{subject to} \quad & \sum_{k=1}^{K_i} d_{ik} \prod_{t=1}^{n} \mathbf{x}_t^{a_{ikt}} \leq 1, \quad i = 1, \ldots, p \\
& d_j \prod_{t=1}^{n} \mathbf{x}_t^{a_{jt}}, \quad j = 1, \ldots, q
\end{aligned}
\tag{1}
$$

where both the objective function and the inequality constraints are posynomials and the equality constraints are monomials. Note that in GP there is also an implicit constraint that $\mathbf{x} \in \mathbb{R}_{++}^n$. GP in its standard form is not a convex program since posynomials are not convex functions in general. However, we can effectively transform it into a convex problem by using the logarithmic transformation trick on $\mathbf{x}$, the multiplicative coefficients of each monomial and also each objective/constraint function [Chiang, 2005, Boyd et al., 2007]. We make the following change: let $\mathbf{y} = \log(\mathbf{x})$, $c_{ik} = \log(d_{ik}), \forall i, k$ and take the $\log(\cdot)$ of each function in (1). Then the standard GP form is equivalent to the following formulation:

$$
\begin{aligned}
\text{minimize} \quad & \log \left(\sum_{k=1}^{K_0} \exp(\mathbf{a}_{0k}^T \mathbf{y} + c_{0k}) \right) \\
\text{subject to} \quad & \log \left(\sum_{k=1}^{K_i} \exp(\mathbf{a}_{ik}^T \mathbf{y} + c_{ik}) \right) \leq 0, \quad i = 1, \ldots, p \\
& \mathbf{a}_j^T \mathbf{y} + c_j = 0, \quad j = 1, \ldots, q
\end{aligned}
\tag{2}
$$

which is a convex program since the log-sum-exp function is convex in its argument and $\mathbf{a}^T \mathbf{y} + c$ is affine in $\mathbf{y}$. Furthermore, in the convex formulation of GP we have $\mathbf{y} \in \mathbb{R}^n$, i.e., we naturally remove the positive constraint on $\mathbf{x}$ by taking the log transformation.

SP has the same form as GP except that the multiplicative constant d inside each monomial is not restricted to be positive, i.e., d can take any real value. Although the difference seems to be small, there is a huge difference between GP and SP from the computational perspective. The negative multiplicative constant in monomials invalidates the log-log transformation trick frequently used in GP. As a result, SP cannot be reduced into a convex program and is believed to be hard to solve [Boyd et al., 2007].

3

3 Unified Approach for Learning

In this section we will first show that the parameter learning problem of SPNs based on the MLE principle can be formulated as an SP. Although SP is hard to solve in general, the structure of the problem at hand leads to efficient algorithms based on sequential convex approximations. More specifically, we will use a sequence of optimal monomial approximations combined with backtracking line search and the concave-convex procedure to tackle the SP.

3.1 Sum-Product Networks as a Mixture of Trees

In this section we introduce the notion of induced trees from SPNs and use it to show that every complete and decomposable SPN can be interpreted as a mixture of induced trees, where each induced tree corresponds to a product of univariate distributions. From this perspective, SPNs in general can be understood as a huge mixture model where the effective number of components in the mixture model is determined by its network structure.

Definition 5 (Induced SPN). Given a complete and decomposable SPN $\mathcal{S}$ over $X_{1:N}$, $\mathcal{T}$ is called an induced SPN from $\mathcal{S}$ if

1. $Root(\mathcal{S}) \in \mathcal{T}_V$.
2. If $v \in \mathcal{T}_V$ is a sum node, then exactly one child of v in $\mathcal{S}$ is in $\mathcal{T}_V$, with the corresponding edge be in $\mathcal{T}_E$.
3. If $v \in \mathcal{T}_V$ is a product node, then all the children of v in $\mathcal{S}$ are in $\mathcal{T}_V$, with the corresponding edges be in $\mathcal{T}_E$.

where in the definition above $\mathcal{T}_V$ is the node set of $\mathcal{T}$ and $\mathcal{T}_E$ is the edge set of $\mathcal{T}$.

For notational convenience we will call $\mathcal{T}$ an induced SPN by omitting the fact that $\mathcal{T}$ is induced from $\mathcal{S}$ if there is no confusion in the context.

Theorem 1. If $\mathcal{T}$ is an induced SPN from $\mathcal{S}$, then $\mathcal{T}$ is a tree that is complete and decomposable.

Proof. Argue by contradiction that $\mathcal{T}$ is not a tree, then there must exist a node $v \in \mathcal{T}$ such that v has more than one parent in $\mathcal{T}$. This means that there exists at least two paths $R, p_1, \ldots, v$ and $R, q_1, \ldots, v$ that connect the root of $\mathcal{S}(\mathcal{T})$, which we denote by R, and v. Let t be the last node in $R, p_1, \ldots, v$ and $R, q_1, \ldots, v$ such that $R, \ldots, t$ are common prefix of these two paths. By construction we know that such t must exist since these two paths start from the same root node R (R will be one candidate of such t). Also, we claim that $t \neq v$ otherwise these two paths overlap with each other, which contradicts the assumption that v has multiple parents. This shows that these two paths can be represented as $R, \ldots, t, p, \ldots, v$ and $R, \ldots, t, q, \ldots, v$ where $R, \ldots, t$ are the common prefix shared by these two paths and $p \neq q$ since t is the last common node. From the construction process defined in Def. 5, we know that both p and q are children of t in $\mathcal{S}$. Recall that for each sum node in $\mathcal{S}$, Def. 5 takes at most one child, hence we claim that t must be a product node, since both p and q are children of t. Then the paths that $t \to p \rightsquigarrow v$ and $t \to q \rightsquigarrow v$ indicate that $\text{scope}(v) \subseteq \text{scope}(p) \subseteq \text{scope}(t)$ and $\text{scope}(v) \subseteq \text{scope}(q) \subseteq \text{scope}(t)$, leading to $\varnothing \neq \text{scope}(v) \subseteq \text{scope}(p) \cap \text{scope}(q)$, which is a contradiction of the decomposability of the product node t. Hence as long as $\mathcal{S}$ is complete and decomposable, $\mathcal{T}$ must be a tree.

The completeness of $\mathcal{T}$ is trivially satisfied because each sum node has only one child in $\mathcal{T}$. It is also straightforward to verify that $\mathcal{T}$ satisfies the decomposability as $\mathcal{T}$ is an induced subgraph of $\mathcal{S}$, which is decomposable. ∎

As a result of Thm. 1, we will use the terms induced SPNs and induced trees interchangeably. With some abuse of notation, we use $\mathcal{T}(\mathbf{x})$ to mean the value of the network polynomial of $\mathcal{T}$ with input vector $\mathbf{x}$.

Theorem 2. If $\mathcal{T}$ is an induced tree from $\mathcal{S}$ over $X_{1:N}$, then $\mathcal{T}(\mathbf{x}) = \prod_{(v_i, v_i) \in \mathcal{T}_E} \theta_{ij} \prod_{n=1}^N \mathbb{I}_{x_n}$, where θ_{ij} is the edge weight of (v_i, v_j) provided v_i is a sum node and $\mathbb{I}_{x_n}$ is the leaf indicator variable in $\mathcal{T}$ of X_n.

Proof. First, the scope of $\mathcal{T}$ is the same as the scope of $\mathcal{S}$ because the root of $\mathcal{S}$ is also the root of $\mathcal{T}$. This shows that for each X_i there is at least one indicator $\mathbb{I}_{x_i}$ in the constant otherwise the scope

4

of the root node of $\mathcal{T}$ will be a strict subset of the scope of the root node of $\mathcal{S}$. Furthermore, for each variable X_i there is at most one indicator $\mathbb{I}_{x_i}$ in the constant. This is observed by the fact that there is at most one child collected from a sum node into $\mathcal{T}$ and if $\mathbb{I}_{x_i}$ and $\mathbb{I}_{\bar{x}_i}$ appear simultaneously in the constant, then their least common ancestor must be a product node. Note that the least common ancestor of $\mathbb{I}_{x_i}$ and $\mathbb{I}_{\bar{x}_i}$ is guaranteed to exist because of the tree structure of $\mathcal{T}$. However, this leads to a contradiction of the fact that $\mathcal{S}$ is decomposable. As a result, there is exactly one indicator $\mathbb{I}_{x_i}$ for each variable X_i in $\mathcal{T}$. Hence the multiplicative constant of the monomial admits the form $\prod_{i=1}^n \mathbb{I}_{x_i}$, which is a product of univariate distributions. More specifically, it is a product of indicator variables in the case of boolean input variables.

We have already shown that $\mathcal{T}$ is a tree and only product nodes in $\mathcal{T}$ can have multiple children. It follows that the functional form of $f_{\mathcal{T}}(\mathbf{x})$ must be a monomial, and only edge weights that are in $\mathcal{T}$ contribute to the monomial. Combing all the above, we know that $f_{\mathcal{T}}(\mathbf{x}) = \prod_{(v_i,v_i)\in\mathcal{T}_E} \theta_{ij} \prod_{i=1}^n \mathbb{I}_{x_i}$. ∎

Definition 6 (Network cardinality). The network cardinality $\tau_{\mathcal{S}}$ of an SPN $\mathcal{S}$ is the number of unique induced trees from $\mathcal{S}$.

Theorem 3. $\tau_{\mathcal{S}} = f_{\mathcal{S}}(\mathbf{1}|\mathbf{1})$ and $\mathcal{S}(\mathbf{x}) = \sum_{t=1}^{\tau_{\mathcal{S}}} \mathcal{T}_t(\mathbf{x})$, where $\mathcal{T}_t$ is the tth unique induced tree of $\mathcal{S}$ and $f_{\mathcal{S}}(\mathbf{1}|\mathbf{1})$ is the value of the network polynomial of $\mathcal{S}$ with input vector $\mathbf{1}$ and all edge weights set to be 1.

Proof. We prove by induction on the height of $\mathcal{S}$. If the height of $\mathcal{S}$ is 2, then depending on the type of the root node, we have two cases:

1. If the root is a sum node with K children, then there are $C_K^1 = K$ different subgraphs that satisfy Def. 5, which is exactly the value of the network by setting all the indicators and edge weights from the root to be 1.
2. If the root is a product node then there is only 1 subgraph which is the graph itself. Again, this equals to the value of $\mathcal{S}$ by setting all indicators to be 1.

Assume the theorem is true for SPNs with height $\leq h$. Consider an SPN $\mathcal{S}$ with height $h+1$. Again, depends on the type of the root node, we need to discuss two cases:

1. If the root is a sum node with K children, where the kth sub-SPN has $f_{\mathcal{S}_k}(\mathbf{1}|\mathbf{1})$ unique induced trees, then by Def. 5 the total number of unique induced trees of $\mathcal{S}$ is $\sum_{k=1}^K f_{\mathcal{S}_k}(\mathbf{1}|\mathbf{1}) = \sum_{k=1}^K 1\cdot f_{\mathcal{S}_k}(\mathbf{1}|\mathbf{1}) = f_{\mathcal{S}}(\mathbf{1}|\mathbf{1})$.
2. If the root is a product node with K children, then the total number of unique induced trees of $\mathcal{S}$ can then be computed by $\prod_{k=1}^K f_{\mathcal{S}_k}(\mathbf{1}|\mathbf{1}) = f_{\mathcal{S}}(\mathbf{1}|\mathbf{1})$.

The second part of the theorem follows by using distributive law between multiplication and addition to combine unique trees that share the same prefix in bottom-up order. ∎

Remark. The above three theorems prove the fact that an SPN $\mathcal{S}$ is essentially an ensemble of trees, where each tree computes an unnormalized distribution over $X_{1:N}$. The total number of unique trees in $\mathcal{S}$ is characterized by the network cardinality $\tau_{\mathcal{S}}$, which only depends on the structure of $\mathcal{S}$. Equivalently, this means that an SPN $\mathcal{S}$ can be treated as a mixture model with $\tau_{\mathcal{S}}$ effective components, where each component is a simple deterministic distribution. We illustrate the theorems above with a simple example shown in Fig. 1.
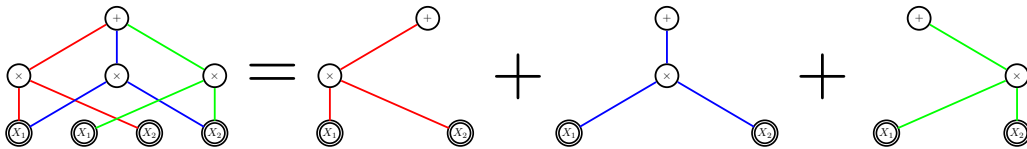

Figure 1: A complete and decomposable SPN is a mixture of induced trees. Double circle indicates univariate distributions over X_1 and X_2. Different colors are used to highlight unique induced trees where each induced tree is a product of univariate distributions over X_1 and X_2.

[Zhao et al., 2015] show that every complete and decomposable SPN is equivalent to a bipartite Bayesian network where a layer of hidden variables pointing to a layer of observable random variables. The number of hidden variables in the bipartite Bayesian network is equal to the number of sum nodes in $\mathcal{S}$. A naive expansion of such Bayesian network to a mixture model will lead to a huge mixture model with $O(2^M)$ components. Here we complement their theory and show that each complete and decomposable SPN is essentially a mixture of trees and the effective number of unique induced trees is given by $\tau_\mathcal{S}$. Note that $\tau_\mathcal{S} = f_\mathcal{S}(\mathbf{1}|\mathbf{1})$ depends only on the network structure. Without loss of generality assuming that in $\mathcal{S}$ layers of sum nodes are alternating with layers of product nodes, then $f_\mathcal{S}(\mathbf{1}|\mathbf{1}) = \Omega(2^h)$, where h is the height of $\mathcal{S}$. However, the exponentially many trees are recursively merged and combined in $\mathcal{S}$ such that the overall network size is still tractable.

3.2 Maximum Likelihood Estimation as SP

Without loss of generality, let's consider the likelihood function computed by an SPN $\mathcal{S}$ over n binary random variables with model parameters $\mathbf{w} \in \mathbb{R}_{++}^D$ and input vector $\mathbf{x} \in \{0,1\}^n$. Here the model parameters in $\mathcal{S}$ are edge weights from every sum node and we collect them together into a long vector $\mathbf{w}$, where D corresponds to the number of edges emanating from sum nodes in $\mathcal{S}$. By definition, the probability distribution induced by $\mathcal{S}$ can be computed by

$$\Pr_\mathcal{S}(\mathbf{x}|\mathbf{w}) \triangleq \frac{f_\mathcal{S}(\mathbf{x}|\mathbf{w})}{\sum_\mathbf{x} f_\mathcal{S}(\mathbf{x}|\mathbf{w})} = \frac{f_\mathcal{S}(\mathbf{x}|\mathbf{w})}{f_\mathcal{S}(\mathbf{1}|\mathbf{w})} \tag{3}$$

where $f_\mathcal{S}(\mathbf{1}|\mathbf{w})$ means the value of the network polynomial by setting all the indicators at the leaves of the network to be 1. The second equation in (3) comes from the fact that the partition function of SPNs can be computed efficiently by setting all the indicators to 1 and evaluating the network in a bottom-up fashion.

Theorem 4. Let $\mathcal{S}$ be an SPN with weights $\mathbf{w} \in \mathbb{R}_{++}^D$ over input vector $\mathbf{x} \in \{0,1\}^n$, the network polynomial $f_\mathcal{S}(\mathbf{x}|\mathbf{w})$ is a posynomial of the following form:

$$f_\mathcal{S}(\mathbf{x}|\mathbf{w}) = \sum_{l=1}^{f_\mathcal{S}(\mathbf{1}|\mathbf{1})} \prod_{i=1}^n \mathbb{I}_{x_i}^{(l)} \prod_{d=1}^D w_d^{\mathbb{I}_{w_d \in \mathcal{S}_l}} \tag{4}$$

where $\mathcal{S}_l$ is the lth unique induced tree taken from $\mathcal{S}$, and $\mathbb{I}_{x_i}^{(l)}$ is the indicator variable of X_i appearing in $\mathcal{S}_l$. $f_\mathcal{S}(\mathbf{1}|\mathbf{1})$ is the value of the network polynomial obtained by setting all indicators and weights to be 1.

This theorem is a direct corollary of Thm. 1 and Thm. 2. From the definition of network polynomial, we know that $f_\mathcal{S}$ is a multilinear function of the indicator variables. Thm. 4 works as a complement to characterize the functional form of a network polynomial in terms of $\mathbf{w}$. Each monomial in (4) corresponds exactly to a unique induced tree SPN from $\mathcal{S}$. It follows that the likelihood function $\mathcal{L}_\mathcal{S}(\mathbf{w}) \triangleq \Pr_\mathcal{S}(\mathbf{x}|\mathbf{w})$ can be expressed as the ratio of two posynomial functions. We now show that the optimization problem based on MLE is an SP.

Proposition 5. The MLE problem for SPNs is a signomial program.

Proof. Using the definition of $\Pr(\mathbf{x}|\mathbf{w})$ and Thm. 4, let $L = f_\mathcal{S}(\mathbf{1}|\mathbf{1})$, the MLE problem can be rewritten as

$$\text{maximize}_\mathbf{w} \quad \frac{f_\mathcal{S}(\mathbf{x}|\mathbf{w})}{f_\mathcal{S}(\mathbf{1}|\mathbf{w})} = \frac{\sum_{l=1}^L \prod_{i=1}^n \mathbb{I}_{x_i}^{(l)} \prod_{d=1}^D w_d^{\mathbb{I}_{w_d \in \mathcal{S}_l}}}{\sum_{l=1}^L \prod_{d=1}^D w_d^{\mathbb{I}_{w_d \in \mathcal{S}_l}}}$$

$$\text{subject to} \quad \mathbf{w} \in \mathbb{R}_{++}^D \tag{5}$$

which we claim is equivalent to:

$$\text{minimize}_{\mathbf{w},t} \quad -t$$

$$\text{subject to} \quad \sum_{l=1}^L t \prod_{d=1}^D w_d^{\mathbb{I}_{w_d \in \mathcal{S}_l}} - \sum_{l=1}^L \prod_{i=1}^n \mathbb{I}_{x_i}^{(l)} \prod_{d=1}^D w_d^{\mathbb{I}_{w_d \in \mathcal{S}_l}} \leq 0 \tag{6}$$

$$\mathbf{w} \in \mathbb{R}_{++}^D, t > 0$$

It is easy to check that both the objective function and constraint function in (6) are signomials. To see the equivalence of (5) and (6), let p^* be the optimal value of (5) achieved at $\mathbf{w}^*$. Choose $t = p^*$ and $\mathbf{w} = \mathbf{w}^*$ in (6), then $-t$ is also the optimal solution of (6) otherwise we can find feasible $(t', \mathbf{w}')$ in (6) which has $-t' < -t \Leftrightarrow t' > t$. Combined with the constraint function in (6), we have $p^* = t < t' \leq \frac{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}')}{f_{\mathcal{S}}(\mathbf{1}|\mathbf{w}')}$, which contradicts the optimality of p^*. In the other direction, let t^*, $\mathbf{w}^*$ be the solution that achieves optimal value of (6), then we claim that t^* is also the optimal value of (5), otherwise there exists a feasible $\mathbf{w}$ in (5) such that $t \triangleq \frac{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{f_{\mathcal{S}}(\mathbf{1}|\mathbf{w})} > t^*$. Since $(\mathbf{w}, t)$ is also feasible in (6) with $-t < -t^*$, this contradicts the optimality of t^*. ∎

The transformation from (5) to (6) does not make the problem any easier to solve. Rather, it destroys the structure of (5), i.e., the objective function of (5) is the ratio of two posynomials. However, the equivalent transformation does reveal some insights about the intrinsic complexity of the optimization problem, which indicates that it is hard to solve (5) efficiently with the guarantee of achieving a globally optimal solution.

3.3 Sequential Convex Approximations

PGD optimizes the log-likelihood by projecting the intermediate solution back to the positive orthant after each gradient update. Since the constraint in (5) is an open set, we need to manually create a closed set on which the projection operation can be well defined. One feasible choice is to project on to $\mathbb{R}^n_\epsilon \triangleq \{\mathbf{x} \in \mathbb{R}^n_{++} \mid x_i \geq \epsilon, \forall i\}$ where $\epsilon > 0$ is assumed to be very small. To avoid the projection, one direct solution is to use the exponentiated gradient (EG) method[Kivinen and Warmuth, 1997], which was first applied in an online setting and latter successfully extended to batch settings when training with convex models [Collins and McAllester, 2005, Globerson et al., 2007, Collins et al., 2008]. EG admits a multiplicative update at each iteration and hence avoids the need for projection in PGD. Both PGD and EG are first-order methods and they can be viewed as approximating the SP after applying a log-transformation to the objective function only.

Notice that although (5) is a signomial program, its objective function is expressed as the ratio of two posynomials. Hence, we can still apply the logarithmic transformation trick used in geometric programming to its objective function and to the variables to be optimized. More concretely, let $w_d = \exp(y_d), \forall d$ and take the $\log$ of the objective function, it becomes equivalent to maximize the following new objective without any constraint on $\mathbf{y}$:

$$\text{maximize} \qquad \log\left(\sum_{l=1}^{L(\mathbf{x})} \exp(\sum_{d=1}^{D} y_d \mathbb{I}_{y_d \in \mathcal{S}_l})\right) - \log\left(\sum_{l=1}^{L} \exp(\sum_{d=1}^{D} y_d \mathbb{I}_{y_d \in \mathcal{S}_l})\right) \qquad (7)$$

Note that in the first term of Eq. 7 the upper index $L(\mathbf{x}) \leq L \triangleq f_{\mathcal{S}}(\mathbf{1}|\mathbf{1})$ depends on the current input $\mathbf{x}$ because not all the induced tree SPNs take positive value for the current input $\mathbf{x}$. By transforming into the log-space, we naturally guarantee the positivity of the solution at each iteration, hence transforming a constrained optimization problem into an unconstrained optimization problem without any sacrifice. Furthermore, both terms in Eq. 7 are convex functions in $\mathbf{y}$ after the transformation. Hence, the transformed objective function is now expressed as the difference of two convex functions, which is called a DC function [Hartman et al., 1959]. The class of DC functions is a super class of convex functions and is closed under most of the operations that can be encountered in mathematical optimization [Piot et al., 2014]. The optimization problem with respect to DC functions is known as DCP. The DCP formulation helps us to design two efficient algorithms to solve the problem based on the general idea of sequential convex approximations for nonlinear programming.

3.3.1 Sequential Monomial Approximation

Let's consider the linearization of both terms in Eq. 7 in order to apply first-order methods in the transformed space. To compute the gradient with respect to different components of $\mathbf{y}$, we view each node of an SPN as an intermediate function of the network polynomial and apply the chain rule to back propagate the gradient. The differentiation of $f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})$ with respect to the root node of the network is set to be 1. The differentiation of the network polynomial with respect to a partial function at each node can then be computed in two passes of the network: the bottom-up pass evaluates the values of all partial functions given the current input $\mathbf{x}$ and the top-down pass differentiates the
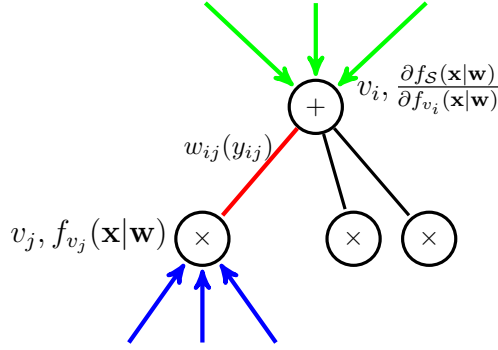
Figure 2: Information flow about the computation of the gradient of log-network polynomial with respect to $y_{ij}(w_{ij})$. Each edge $y_{ij}(w_{ij})$ collects the evaluation value from v_j in bottom-up pass and also differentiation value from v_i in top-down pass.

network polynomial with respect to each partial function. Since the model parameters $\mathbf{y}(\mathbf{w})$ are only associated with sum nodes, they can be easily computed once we have obtained the differentiations for each node:

$$\frac{\partial \log f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial y_{ij}} = \frac{\partial \log f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})} f_{v_j}(\mathbf{x}|\mathbf{w}) w_{ij} \tag{8}$$

where v_i is restricted to be a sum node and v_j is a child of v_i. Following the evaluation-differentiation passes, the gradient of the objective function in (7) can be computed in $O(|\mathcal{S}|)$. Furthermore, although the computation is conducted in $\mathbf{y}$, the results are fully expressed in terms of $\mathbf{w}$, which suggests that in practice we do not need to explicitly construct $\mathbf{y}$ from $\mathbf{w}$. An illustration of the process is provided in Fig. 2.

Let $f(\mathbf{y}) = \log f_{\mathcal{S}}(\mathbf{x}|\exp(\mathbf{y})) - \log f_{\mathcal{S}}(\mathbf{1}|\exp(\mathbf{y}))$, consider the optimal first-order approximation of $f(\mathbf{y})$ at point $\mathbf{y}^{(k)}$:

$$\hat{f}(\mathbf{y}) = f(\mathbf{y}^{(k)}) + \nabla_{\mathbf{y}} f(\mathbf{y}^{(k)})^T (\mathbf{y} - \mathbf{y}^{(k)}) \tag{9}$$

which is equivalent to

$$\exp(\hat{f}(\mathbf{w})) = \exp\{f(\mathbf{y}^{(k)}) + \nabla_{\mathbf{y}} f(\mathbf{y}^{(k)})^T (\mathbf{y} - \mathbf{y}^{(k)})\} = C_1 \prod_{d=1}^{D} w_d^{\nabla_{y_d} f(\mathbf{y}^{(k)})} \tag{10}$$

in the original space, where C_1 is a positive constant irrelevant of $\mathbf{y}(\mathbf{w})$. Eq. 10 suggests that approximating $\hat{f}(\mathbf{y})$ with the best linear function is equivalent to using the best monomial approximation of the signomial program (5). This leads to a sequential monomial approximation of the original SP formulation: at each iteration $\mathbf{y}^{(k)}$, we linearize both terms in Eq. 7 and form the optimal monomial function in terms of $\mathbf{w}^{(k)}$. The additive update of $\mathbf{y}^{(k)}$ in Eq. 9 leads to a multiplicative update of $\mathbf{w}^{(k)}$. We use a backtracking line search to determine the step size of the update in each iteration.

3.3.2 Concave-convex Procedure

Sequential monomial approximation is a general approximation technique that is applicable to any SP [Boyd et al., 2007, Chiang, 2005]. It linearizes both terms in $f(\mathbf{y})$ to obtain an approximation of (7) and then applies gradient descent in the transformed space. However, it fails to utilize the structure of the problem when learning SPNs. Here we propose another approach based on the concave-convex procedure (CCCP) [Yuille et al., 2002] to utilize the fact that the objective function is expressed as the difference of two convex functions. At a high level CCCP solves a sequence of concave surrogate functions until convergence. In most of the cases, the maximum of a concave surrogate function can only be solved using other convex solvers and as a result the efficiency of the CCCP highly depends on the choice of the convex solvers. However, we show that by a suitable transformation of the network we can compute the maximum of the concave surrogate in closed

form in time that is linear in the network size, which leads to a very efficient algorithm for learning the parameters of SPNs. We also prove the convergence properties of our algorithm.

Consider the objective function to be maximized in DCP: $f(\mathbf{y}) = \log f_{\mathcal{S}}(\mathbf{x}|\exp(\mathbf{y})) - \log f_{\mathcal{S}}(\mathbf{1}|\exp(\mathbf{y})) \triangleq f_1(\mathbf{y}) + f_2(\mathbf{y})$ where $f_1(\mathbf{y}) \triangleq \log f_{\mathcal{S}}(\mathbf{x}|\exp(\mathbf{y}))$ is a convex function and $f_2(\mathbf{y}) \triangleq -\log f_{\mathcal{S}}(\mathbf{1}|\exp(\mathbf{y}))$ is a concave function. We can linearize only the convex part $f_1(\mathbf{y})$ to obtain a surrogate function

$$\hat{f}(\mathbf{y}, \mathbf{z}) = f_1(\mathbf{z}) + \nabla_{\mathbf{y}} f_1(\mathbf{z})^T (\mathbf{y} - \mathbf{z}) + f_2(\mathbf{y}) \tag{11}$$

for $\forall \mathbf{y}, \mathbf{z} \in \mathbb{R}^D$. Due to the convexity of $f_1(\mathbf{y})$ we have $f_1(\mathbf{y}) \geq f_1(\mathbf{z}) + \nabla_{\mathbf{y}} f_1(\mathbf{z})^T (\mathbf{y} - \mathbf{z}), \forall \mathbf{y}, \mathbf{z}$ and as a result the following two properties always hold for $\forall \mathbf{y}, \mathbf{z}$:

$$\hat{f}(\mathbf{y}, \mathbf{z}) \leq f(\mathbf{y}) \quad \text{and} \quad \hat{f}(\mathbf{y}, \mathbf{y}) = f(\mathbf{y}) \tag{12}$$

Hence, $\hat{f}(\mathbf{y}, \mathbf{z})$ is a concave function in $\mathbf{y}$. CCCP updates $\mathbf{y}$ at each iteration k by solving

$$\mathbf{y}^{(k)} \in \arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)}) \tag{13}$$

unless we already have $\mathbf{y}^{(k-1)} \in \arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$, in which case a generalized fixed point $\mathbf{y}^{(k-1)}$ has been found and the algorithm stops. It is easy to show that at each iteration of CCCP we always have $f(\mathbf{y}^{(k)}) \geq f(\mathbf{y}^{(k-1)})$. Note also that $f(\mathbf{y})$ is computing the log-likelihood of input $\mathbf{x}$ and therefore it is bounded above by 0. By the monotone convergence theorem, $\lim_{k \to \infty} f(\mathbf{y}^{(k)})$ exists and the sequence $\{f(\mathbf{y}^{(k)})\}$ converges.

We now discuss how to compute a closed form solution for the maximization of the concave surrogate $\hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$. Since $\hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$ is concave for any fixed $\mathbf{y}^{(k-1)}$, a sufficient and necessary condition to find its maximum is

$$\nabla_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)}) = \nabla_{\mathbf{y}} f_1(\mathbf{y}^{(k-1)}) + \nabla_{\mathbf{y}} f_2(\mathbf{y}) = 0 \tag{14}$$

In the above equation, if we consider only the partial derivative with respect to $y_{ij}(w_{ij})$, we obtain

$$\frac{w_{ij}^{(k-1)} f_{v_j}(\mathbf{x}|\mathbf{w}^{(k-1)})}{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})} \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w}^{(k-1)})} = \frac{w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})}{f_{\mathcal{S}}(\mathbf{1}|\mathbf{w})} \frac{\partial f_{\mathcal{S}}(\mathbf{1}|\mathbf{w})}{\partial f_{v_i}(\mathbf{1}|\mathbf{w})} \tag{15}$$

Since there are D partial derivatives in the gradient, this leads to a system of D nonlinear equations, which is hard to solve in closed form. However, if we do a change of variable by considering locally normalized weights w'_{ij} (i.e., $w'_{ij} \geq 0$ and $\sum_j w'_{ij} = 1 \ \forall i$), then a solution can be easily computed. As described in [Peharz et al., 2015, Zhao et al., 2015], any SPN can be transformed into an equivalent <u>normal</u> SPN with normalized weights in a bottom up pass as follows:

$$w'_{ij} = \frac{w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})}{\sum_j w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})} \tag{16}$$

We can then replace $w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})$ in the above equation by the expression it is equal to in Eq. 15 to obtain a closed form solution:

$$w'_{ij} \propto w_{ij}^{(k-1)} \frac{f_{v_j}(\mathbf{x}|\mathbf{w}^{(k-1)})}{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})} \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w}^{(k-1)})} \tag{17}$$

Eq. 17 also suggests that in order to obtain a solution to Eq. 15, for each edge weight w_{ij}, the sufficient statistics include only three terms, i.e, the evaluation value at v_j, the differentiation value at v_i and the previous edge weight $w_{ij}^{(k-1)}$, all of which can be obtained in two passes of the network for each input $\mathbf{x}$ as illustrated in Fig. 2. Thus the computational complexity to obtain a maximum of the concave surrogate is $O(|\mathcal{S}|)$.

We discussed before that the sequence of function values $\{f(\mathbf{y}^{(k)})\}$ converges to a limiting point. However, this fact alone does not necessarily indicate that $\{f(\mathbf{y}^{(k)})\}$ converges to $f(\mathbf{y}^*)$ where $\mathbf{y}^*$ is a stationary point of $f(\cdot)$ nor does it imply that the sequence $\{\mathbf{y}^{(k)}\}$ converges as $k \to \infty$. <u>Zangwill's global convergence theory</u> [Zangwill, 1969] has been successfully applied to study the convergence properties of many iterative algorithms frequently used in machine learning, including EM [Wu, 1983], generalized alternating minimization [Gunawardana and Byrne, 2005] and also CCCP [Lanckriet and Sriperumbudur, 2009]. Here we also apply Zangwill's theory and combine the analysis from Lanckriet and Sriperumbudur [2009] to show the following theorem:

Theorem 6. Let $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ be any sequence generated using Eq. 17 from any positive initial point, then all the limiting points of $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ are stationary points of the DCP in (7). In addition, $\lim_{k\to\infty} f(\mathbf{y}^{(k)}) = f(\mathbf{y}^*)$, where $\mathbf{y}^*$ is some stationary point of (7).

We delay the proof of this theorem to appendix. It is worth to point out that the above theorem <u>does not</u> imply the convergence of the sequence $\{\mathbf{w}_{(k)}\}_{k=1}^{\infty}$. Thm. 6 only indicates that all the limiting points of $\{\mathbf{w}_{(k)}\}_{k=1}^{\infty}$, i.e., the limits of subsequences of $\{\mathbf{w}_{(k)}\}_{k=1}^{\infty}$, are stationary points of the DCP in (7). We also present a negative example in the appendix that invalidates the application of Zangwill's global convergence theory on the analysis in this case.

The convergence rate of general CCCP is still an open problem [Lanckriet and Sriperumbudur, 2009]. Salakhutdinov et al. [2002] studied the convergence rate of unconstrained bound optimization algorithms with differentiable objective functions, of which our problem is a special case. The conclusion is that depending on the curvature of f_1 and f_2 (which are functions of the training data), CCCP will exhibit either a quasi-Newton behavior with superlinear convergence or first-order convergence. We show in experiments that CCCP normally exhibits a fast, superlinear convergence rate compared with PGD, EG and SMA. Both CCCP and EM are special cases of a more general framework known as Majorization-Maximization Hunter and Lange [2004]. We show that in the case of SPNs these two algorithms coincide with each other, i.e., they lead to the same update formulas despite the fact that they start from totally different perspectives. We refer interested readers to the appendix for a detailed derivation and discussion. We summarize all 4 algorithms and highlight their connections and differences in Table 1.

Table 1: Summary of PGD, EG, SMA and CCCP. Var. means the optimization variables.

Algo	Var.	Update Type	Approx. Type	Update Formula
PGD	$\mathbf{w}$	Additive	Linearization	$w_d^{(k+1)} \leftarrow P_{\mathbb{R}_{++}^\epsilon}\left\{w_d^{(k)} + \gamma(\nabla_{w_d}f_1(\mathbf{w}^{(k)}) - \nabla_{w_d}f_2(\mathbf{w}^{(k)}))\right\}$
EG	$\mathbf{w}$	Multiplicative	Linearization	$w_d^{(k+1)} \leftarrow w_d^{(k)}\exp\{\gamma(\nabla_{w_d}f_1(\mathbf{w}^{(k)}) - \nabla_{w_d}f_2(\mathbf{w}^{(k)}))\}$
SMA	$\log\mathbf{w}$	Multiplicative	Monomial	$w_d^{(k+1)} \leftarrow w_d^{(k)}\exp\{\gamma w_d^{(k)} \times (\nabla_{w_d}f_1(\mathbf{w}^{(k)}) - \nabla_{w_d}f_2(\mathbf{w}^{(k)}))\}$
CCCP	$\log\mathbf{w}$	Multiplicative	Log-sum-exp+Affine	$w_{ij}^{(k+1)} \propto w_{ij}^{(k)} \times \nabla_{v_i}f_{\mathcal{S}}(\mathbf{w}^{(k)}) \times f_{v_j}(\mathbf{w}^{(k)})$

4 Experiments

4.1 Experimental Setting

We conduct experiments on 20 benchmark data sets from various domains to compare and evaluate the convergence performance of the four algorithms: PGD, EG, SMA and CCCP. These 20 data sets are widely used in [Gens and Domingos, 2013, Rooshenas and Lowd, 2014] to assess different SPNs for the task of density estimation. The domain of these 20 data sets include click-through logs, plant habitats, nucleic acid sequences, text documents, movie rates and many others. All the features in the 20 data sets are binary features. The 20 data sets cover three typical statistical estimation settings where the number of training instances is either greater, similar or much smaller than the number of optimization variables. Hence, they enable a thorough experimental comparison to test these 4 algorithms under different applications and statistical scenarios. Detailed information about these 20 datasets and the SPNs used in the experiments are provided in Table. 2 and Table. 5. All the SPNs that are used for comparisons of PGD, EG, SMA and EM are trained using LearnSPN [Gens and Domingos, 2013]. We throw away the weights returned by LearnSPN and use random weights as initial model parameters. The random weights are determined by the same random seed in all four algorithms.

4.2 Parameter Learning

We implement all four algorithms in C++ and test them on a compute server with 32 cores. Each core is Intel Xeon(R) CPU E5 2.00GHz. For each algorithm, we set the maximum number of iterations to 50. If the absolute difference in the training log-likelihood at two consecutive steps is less than 0.001, the algorithms are stopped. We combine PGD, EG and SMA with backtracking line

Table 2: Statistics of data sets and models. N is the number of variables modeled by the network, $|\mathcal{S}|$ is the size of the network and p is the number of parameters to be estimated in the network. $N \times n/p$ means the ratio of training instances times the number of variables to the number parameters.

Data set	N	$|\mathcal{S}|$	p	Train	Valid	Test	$N \times n/p$
NLTCS	16	13,733	1,716	16,181	2,157	3,236	150.871
MSNBC	17	54,839	24,452	291,326	38,843	58,265	202.541
KDD 2k	64	48,279	14,292	180,092	19,907	34,955	806.457
Plants	69	132,959	58,853	17,412	2,321	3,482	20.414
Audio	100	739,525	196,103	15,000	2,000	3,000	7.649
Jester	100	314,013	180,750	9,000	1,000	4,116	4.979
Netflix	100	161,655	51,601	15,000	2,000	3,000	29.069
Accidents	111	204,501	74,804	12,758	1,700	2,551	18.931
Retail	135	56,931	22,113	22,041	2,938	4,408	134.560
Pumsb-star	163	140,339	63,173	12,262	1,635	2,452	31.638
DNA	180	108,021	52,121	1,600	400	1,186	5.526
Kosarak	190	203,321	53,204	33,375	4,450	6,675	119.187
MSWeb	294	68,853	20,346	29,441	3,270	5,000	425.423
Book	500	190,625	41,122	8,700	1,159	1,739	105.783
EachMovie	500	522,753	188,387	4,524	1,002	591	12.007
WebKB	839	1,439,751	879,893	2,803	558	838	2.673
Reuters-52	889	2,210,325	1,453,390	6,532	1,028	1,540	3.995
20 Newsgrp	910	14,561,965	8,295,407	11,293	3,764	3,764	1.239
BBC	1058	1,879,921	1,222,536	1,670	225	330	1.445
Ad	1556	4,133,421	1,380,676	2,461	327	491	2.774

search and use a weight shrinking coefficient set at 0.8. The learning rates are initialized to 1.0 for all three methods. For PGD, we set the projection margin ϵ to 0.01. There is no learning rate and no backtracking line search in CCCP. We set the smoothing parameter to 0.001 in CCCP to avoid numerical issues.

We show in Fig. 3 the average log-likelihood scores on 20 training data sets to evaluate the convergence speed and stability of PGD, EG, SMA and CCCP. Clearly, CCCP wins by a large margin over PGD, EG and SMA, both in convergence speed and solution quality. Furthermore, among the four algorithms, CCCP is the most stable due to its guarantee that the log-likelihood (on training data) will not decrease after each iteration. These 20 experiments also clearly show that CCCP often converges in a few iterations, exhibiting a superlinear convergence speed. On the other hand, PGD, EG and SMA are on par with each other since they are all first-order methods. SMA is more stable than PGD and EG and often achieves better solutions than PGD and EG. On large data sets, SMA also converges faster than PGD and EG. Surprisingly, EG performs worse than PGD in some cases and is quite unstable despite the fact that it admits multiplicative updates. The "hook shape" curves of PGD in some data sets are due to the projection operations, which makes PGD the most unstable one among the four algorithms. The computational complexity per update is $O(|\mathcal{S}|)$ in all four algorithms. The constant involved in the $|\mathcal{S}|$ term of CCCP is slightly larger than those of the other three algorithms as there are more $\exp(\cdot)$ calls in CCCP. However, in practice, CCCP often takes less time than the other three algorithms because it takes fewer iterations to converge. We list detailed running time statistics for all four algorithms on the 20 data sets in Table. 3.

4.3 Boosting Structure Learning

In this experiment, we combine CCCP as a "fine tuning" procedure with the structure learning algorithm LearnSPN and compare it to the state-of-the-art structure learning algorithm ID-SPN [Rooshenas and Lowd, 2014]. More concretely, we keep the model parameters learned from LearnSPN and use them to initialize CCCP. We then update the model parameters globally using CCCP as a fine tuning technique. This normally helps to obtain a better generative model since the original parameters are learned greedily and locally during the structure learning algorithm. The final CCCP procedure adjusts all the parameters globally toward a better solution. In this experiment, we use the validation set log-likelihood score to avoid overfitting. The algorithm returns the set of
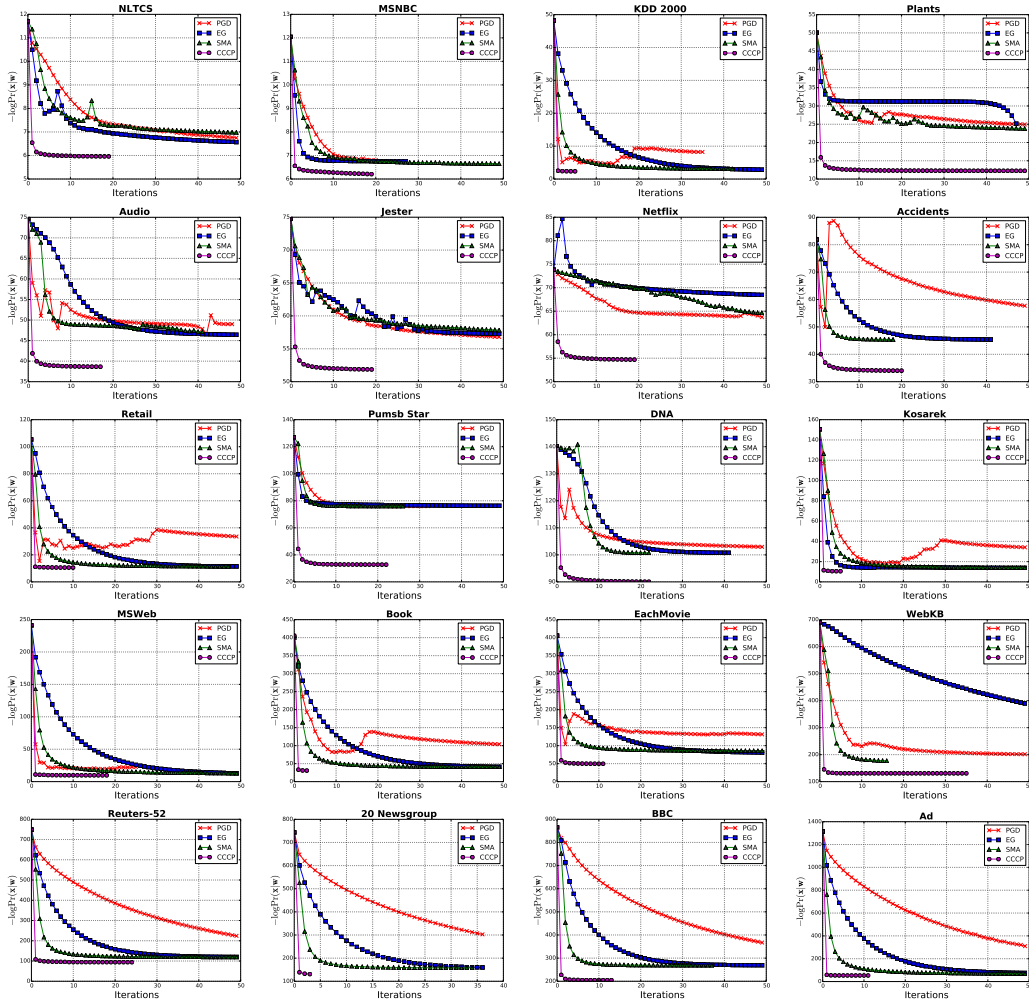
Figure 3: Negative log-likelihood values on training data versus number of iterations for PGD, EG, SMA and CCCP.

parameters that achieve the best validation set log-likelihood score as output. Experimental results are reported in Table. 4.

As shown in Table 4, the use of CCCP after LearnSPN always helps to improve the model performance. By optimizing model parameters on these 20 data sets, we boost LearnSPN to achieve better results than state-of-the-art ID-SPN on 7 data sets, where the original LearnSPN only outperforms ID-SPN on 1 data set. Note that the sizes of the SPNs returned by LearnSPN are much smaller than those produced by ID-SPNs. Hence, it is remarkable that by fine tuning the parameters with CCCP, we can achieve better performance despite the fact that the models are smaller. For a fair comparison, we also list the size of the SPNs returned by ID-SPN in Table. 5. As a result, we suggest using CCCP after structure learning algorithms to fully exploit the expressiveness of the constructed model.

5   Conclusion

In this paper we show that the network polynomial of an SPN is a posynomial function of the model parameters and that learning the parameter by maximum likelihood yields a signomial program. We propose two convex relaxations to solve the SP based on sequential monomial approximations and the concave-convex procedure. We analyze the convergence properties of CCCP for learning SPNs. Extensive experiments are conducted to evaluate the proposed approaches and current methods.

Table 3: Running time of 4 algorithms on 20 data sets, measured in seconds.

Data set	PGD	EG	SMA	CCCP
NLTCS	438.35	718.98	458.99	206.10
MSNBC	2720.73	2917.72	8078.41	2008.07
KDD 2k	46388.60	22154.10	27101.50	29541.20
Plants	12595.60	10752.10	7604.09	13049.80
Audio	19647.90	3430.69	12801.70	14307.30
Jester	6099.44	6272.80	4082.65	1931.41
Netflix	29573.10	27931.50	15080.50	8400.20
Accidents	14266.50	3431.82	5776.00	20345.90
Retail	28669.50	7729.89	9866.94	5200.20
Pumsb-star	3115.58	13872.80	4864.72	2377.54
DNA	599.93	199.63	727.56	1380.36
Kosarak	122204.00	112273.00	49120.50	42809.30
MSWeb	136524.00	13478.10	65221.20	45132.30
Book	190398.00	6487.84	69730.50	23076.40
EachMovie	30071.60	32793.60	17751.10	60184.00
WebKB	123088.00	50290.90	44004.50	168142.00
Reuters-52	13092.10	5438.35	20603.70	1194.31
20 Newsgrp	151243.00	96025.80	173921.00	11031.80
BBC	20920.60	18065.00	36952.20	3440.37
Ad	12246.40	2183.08	12346.70	731.48

Table 4: Average log-likelihoods on test data. Highest average log-likelihoods are highlighted in bold.

Data set	CCCP	LearnSPN	ID-SPN
NLTCS	**-6.029**	-6.099	-6.050
MSNBC	**-6.045**	-6.113	-6.048
KDD 2k	**-2.134**	-2.233	-2.153
Plants	-12.872	-12.955	**-12.554**
Audio	-40.020	-40.510	**-39.824**
Jester	**-52.880**	-53.454	-52.912
Netflix	-56.782	-57.385	**-56.554**
Accidents	-27.700	-29.907	**-27.232**
Retail	**-10.919**	-11.138	-10.945
Pumsb-star	-24.229	-24.577	**-22.552**
DNA	-84.921	-85.237	**-84.693**
Kosarak	-10.880	-11.057	**-10.605**
MSWeb	-9.970	-10.269	**-9.800**
Book	-35.009	-36.247	**-34.436**
EachMovie	-52.557	-52.816	**-51.550**
WebKB	-157.492	-158.542	**-153.293**
Reuters-52	-84.628	-85.979	**-84.389**
20 Newsgrp	-153.205	-156.605	**-151.666**
BBC	**-248.602**	-249.794	-252.602
Ad	**-27.202**	-27.409	-40.012

Table 5: Sizes of SPNs produced by LearnSPN and ID-SPN.

Data set	LearnSPN	ID-SPN
NLTCS	13,733	24,690
MSNBC	54,839	579,364
KDD 2k	48,279	1,286,657
Plants	132,959	2,063,708
Audio	739,525	2,643,948
Jester	314,013	4,225,471
Netflix	161,655	7,958,088
Accidents	204,501	2,273,186
Retail	56,931	60,961
Pumsb-star	140,339	1,751,092
DNA	108,021	3,228,616
Kosarak	203,321	1,272,981
MSWeb	68,853	1,886,777
Book	190,625	1,445,501
EachMovie	522,753	2,440,864
WebKB	1,439,751	2,605,141
Reuters-52	2,210,325	4,563,861
20 Newsgrp	14,561,965	3,485,029
BBC	1,879,921	2,426,602
Ad	4,133,421	2,087,253

References

M. R. Amer and S. Todorovic. Sum-product networks for modeling activities with stochastic structure. In Computer Vision and Pattern Recognition, 2012.

S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. Optimization and engineering, 8(1):67–127, 2007.

W.-C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai. Language modeling with sum-product networks. INTERSPEECH, 2014.

M. Chiang. Geometric programming for communication systems. Now Publishers Inc, 2005.

M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. The Journal of Machine Learning Research, 9:1775–1822, 2008.

P. L. B. M. Collins and B. T. D. McAllester. Exponentiated gradient algorithms for large-margin structured classification. In Advances in Neural Information Processing Systems 17: Proceedings of the 2004 Conference, volume 17, page 113. MIT Press, 2005.

A. Darwiche. A differential approach to inference in bayesian networks. Journal of the ACM (JACM), 50(3):280–305, 2003.

R. Gens and P. Domingos. Discriminative learning of sum-product networks. In Advances in Neural Information Processing Systems, pages 3248–3256, 2012.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In Proceedings of The 30th International Conference on Machine Learning, pages 873–880, 2013.

A. Globerson, T. Y. Koo, X. Carreras, and M. Collins. Exponentiated gradient algorithms for log-linear structured prediction. In Proceedings of the 24th international conference on Machine learning, pages 305–312. ACM, 2007.

A. Gunawardana and W. Byrne. Convergence theorems for generalized alternating minimization procedures. The Journal of Machine Learning Research, 6:2049–2073, 2005.

P. Hartman et al. On functions representable as a difference of convex functions. Pacific J. Math, 9 (3):707–713, 1959.

D. R. Hunter and K. Lange. A tutorial on mm algorithms. The American Statistician, 58(1):30–37, 2004.

J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Information and Computation, 132(1):1–63, 1997.

G. R. Lanckriet and B. K. Sriperumbudur. On the convergence of the concave-convex procedure. pages 1759–1767, 2009.

D.-I. R. Peharz. Foundations of Sum-Product Networks for Probabilistic Modeling. PhD thesis, Aalborg University, 2015.

R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, pages 3699–3703. IEEE, 2014.

R. Peharz, S. Tschiatschek, F. Pernkopf, P. Domingos, and B. BioTechMed-Graz. On theoretical properties of sum-product networks. pages 744–752, 2015.

B. Piot, M. Geist, and O. Pietquin. Difference of convex functions programming for reinforcement learning. pages 2519–2527, 2014.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In Proc. 12th Conf. on Uncertainty in Artificial Intelligence, pages 2551–2558, 2011.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In Proceedings of The 31st International Conference on Machine Learning, pages 710–718, 2014.

R. Salakhutdinov, S. Roweis, and Z. Ghahramani. On the convergence of bound optimization algorithms. Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence, pages 509–516, 2002.

C. J. Wu. On the convergence properties of the em algorithm. The Annals of statistics, pages 95–103, 1983.

A. L. Yuille, A. Rangarajan, and A. Yuille. The concave-convex procedure (cccp). Advances in neural information processing systems, 2:1033–1040, 2002.

W. I. Zangwill. Nonlinear programming: a unified approach, volume 196. Prentice-Hall Englewood Cliffs, NJ, 1969.

H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and bayesian networks. In Proceedings of The 32nd International Conference on Machine Learning, 2015.

A Convergence of CCCP for SPNs

Theorem 6. Let $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ be any sequence generated using Eq. 17 from any positive initial point, then all the limiting points of $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ are stationary points of the DCP in (7). In addition, $\lim_{k \to \infty} f(\mathbf{y}^{(k)}) = f(\mathbf{y}^*)$, where $\mathbf{y}^*$ is some stationary point of (7).

Proof. We will use Zangwill's global convergence theory for iterative algorithms [Zangwill, 1969] to show the convergence in our case. Before showing the proof, we need to first introduce the notion of "point-to-set mapping", where the output of the mapping is defined to be a set. More formally, a point-to-set map Φ from a set $\mathcal{X}$ to $\mathcal{Y}$ is defined as $\Phi : \mathcal{X} \mapsto \mathcal{P}(\mathcal{Y})$, where $\mathcal{P}(\mathcal{Y})$ is the power set of $\mathcal{Y}$. Suppose $\mathcal{X}$ and $\mathcal{Y}$ are equipped with the norm $||\cdot||_{\mathcal{X}}$ and $||\cdot||_{\mathcal{Y}}$, respectively. A point-to-set map Φ is said to be <u>closed</u> at $x^* \in \mathcal{X}$ if $x_k \in \mathcal{X}, \{x_k\}_{k=1}^{\infty} \to x^*$ and $y_k \in \mathcal{Y}, \{y_k\}_{k=1}^{\infty} \to y^*, y_k \in \Phi(x_k)$ imply that $y^* \in \Phi(x^*)$. A point-to-set map Φ is said to be closed on $S \subseteq \mathcal{X}$ if Φ is closed at every point in S. The concept of <u>closedness</u> in the point-to-set map setting reduces to <u>continuity</u> if we restrict that the output of Φ to be a set of singleton for every possible input, i.e., when Φ is a point-to-point mapping.

Theorem 7 (Global Convergence Theorem [Zangwill, 1969])**.** Let the sequence $\{x_k\}_{k=1}^{\infty}$ be generated by $x_{k+1} \in \Phi(x_k)$, where $\Phi(\cdot)$ is a point-to-set map from $\mathcal{X}$ to $\mathcal{X}$. Let a solution set $\Gamma \subseteq \mathcal{X}$ be given, and suppose that:

1. all points x_k are contained in a compact set $S \subseteq \mathcal{X}$.
2. Φ is closed over the complement of Γ.
3. there is a continuous function α on $\mathcal{X}$ such that:
   (a) if $x \notin \Gamma$, $\alpha(x') > \alpha(x)$ for $\forall x' \in \Phi(x)$.
   (b) if $x \in \Gamma$, $\alpha(x') \geq \alpha(x)$ for $\forall x' \in \Phi(x)$.

Then all the limit points of $\{x_k\}_{k=1}^{\infty}$ are in the solution set Γ and $\alpha(x_k)$ converges monotonically to $\alpha(x^*)$ for some $x^* \in \Gamma$.

Let $\mathbf{w} \in \mathbb{R}_+^D$. Let $\Phi(\mathbf{w}^{(k-1)}) = \exp(\arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)}))$ and let $\alpha(\mathbf{w}) = f(\log \mathbf{w}) = f(\mathbf{y}) = \log f_{\mathcal{S}}(\mathbf{x}|\exp(\mathbf{y})) - \log f_{\mathcal{S}}(\mathbf{1}|\exp(\mathbf{y}))$. Here we use $\mathbf{w}$ and $\mathbf{y}$ interchangeably as $\mathbf{w} = \exp(\mathbf{y})$ or each component is a one-to-one mapping. Note that since the $\arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$ given $\mathbf{y}^{(k-1)}$ is achievable, $\Phi(\cdot)$ is a well defined point-to-set map for $\mathbf{w} \in \mathbb{R}_+^D$.

Specifically, in our case given $\mathbf{w}^{(k-1)}$, at each iteration of Eq. 17 we have

$$w'_{ij} = \frac{w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})}{\sum_j w_{ij} f_{v_j}(\mathbf{1}|\mathbf{w})} \propto w_{ij}^{(k-1)} \frac{f_{v_j}(\mathbf{x}|\mathbf{w}^{(k-1)})}{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})} \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w}^{(k-1)})}$$

i.e., the point-to-set mapping is given by

$$\Phi_{ij}(\mathbf{w}^{(k-1)}) = \frac{w_{ij}^{(k-1)} f_{v_j}(\mathbf{x}|\mathbf{w}^{(k-1)}) \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w}^{(k-1)})}}{\sum_{j'} w_{ij'}^{(k-1)} f_{v_{j'}}(\mathbf{x}|\mathbf{w}^{(k-1)}) \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w}^{(k-1)})}}$$

Let $S = [0,1]^D$, the D dimensional hyper cube. Then the above update formula indicates that $\Phi(\mathbf{w}^{(k-1)}) \in S$. Furthermore, if we apply Alg. **??** to $\mathbf{w}^{(1)}$ before the update, we can guarantee that $\{\mathbf{w}_k\}_{k=1}^{\infty} \subseteq S$, which is a compact set in $\mathbb{R}_+^D$.

The solution to $\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$ is not unique. In fact, there are infinitely many solutions to this nonlinear equations. However, as we define above, $\Phi(\mathbf{w}^{(k-1)})$ returns <u>one</u> solution to this convex program in the D dimensional hyper cube. Hence in our case $\Phi(\cdot)$ reduces to a point-to-point map, where the definition of <u>closedness</u> of a point-to-set map reduces to the notion of <u>continuity</u> of a point-to-point map. Define $\Gamma = \{\mathbf{w}^* \mid \mathbf{w}^* \text{ is a stationary point of } \alpha(\cdot)\}$. Hence we only need to verify the continuity of $\Phi(\mathbf{w})$ when $\mathbf{w} \in S$. To show this, we first characterize the functional form of $\frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})}$ as it is used inside $\Phi(\cdot)$. We claim that for each node v_i, $\frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})}$ is again, a posynomial function of $\mathbf{w}$. A graphical illustration is given in Fig. 4 to explain the process. This can also be derived from
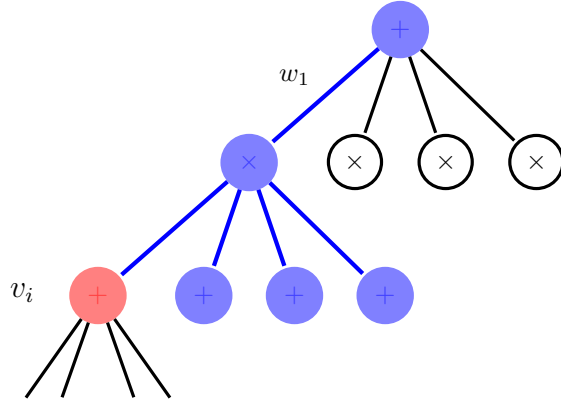
Figure 4: Graphical illustration of $\frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})}$. The partial derivative of f_S with respect to f_{v_i} (in red) is a posynomial that is a product of edge weights lying on the path from root to v_i and network polynomials from nodes that are children of product nodes on the path (highlighted in blue).

the sum rules and product rules used during top-down differentiation. More specifically, if v_i is a product node, let $v_j, j = 1, \ldots, J$ be its parents in the network, which are assumed to be sum nodes, the differentiation of f_S with respect to f_{v_i} is given by $\frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})} = \sum_{j=1}^{J} \frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_j}(\mathbf{x}|\mathbf{w})} \frac{\partial f_{v_j}(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})}$. We reach

$$\frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})} = \sum_{j=1}^{J} w_{ij} \frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_j}(\mathbf{x}|\mathbf{w})} \tag{18}$$

Similarly, if v_i is a sum node and its parents $v_j, j = 1, \ldots, J$ are assumed to be product nodes, we have

$$\frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})} = \sum_{j=1}^{J} \frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_j}(\mathbf{x}|\mathbf{w})} \frac{f_{v_j}(\mathbf{x}|\mathbf{w})}{f_{v_i}(\mathbf{x}|\mathbf{w})} \tag{19}$$

Since v_j is a product node and v_j is a parent of v_i, so the last term in Eq. 19 can be equivalently expressed as

$$\frac{f_{v_j}(\mathbf{x}|\mathbf{w})}{f_{v_i}(\mathbf{x}|\mathbf{w})} = \prod_{h \neq i} f_{v_h}(\mathbf{x}|\mathbf{w})$$

where the index is range from all the children of v_j except v_i. Combining the fact that the partial differentiation of f_S with respect to the root node is 1 and that each f_v is a posynomial function, it follows by induction in top-down order that $\frac{\partial f_S(\mathbf{x}|\mathbf{w})}{\partial f_{v_i}(\mathbf{x}|\mathbf{w})}$ is also a posynomial function of $\mathbf{w}$.

We have shown that both the numerator and the denominator of $\Phi(\cdot)$ are posynomial functions of $\mathbf{w}$. Because posynomial functions are continuous functions, in order to show that $\Phi(\cdot)$ is also continuous on $S\backslash\Gamma$, we need to guarantee that the denominator is not a degenerate posynomial function, i.e., the denominator of $\Phi(\mathbf{w}) \neq 0$ for all possible input vector $\mathbf{x}$. Recall that $\Gamma = \{\mathbf{w}^* \mid \mathbf{w}^*$ is a stationary point of $\alpha(\cdot)\}$, hence $\forall \mathbf{w} \in S\backslash\Gamma, \mathbf{w} \notin \text{bd } S$, where bd S is the boundary of the D dimensional hyper cube S. Hence we have $\forall \mathbf{w} \in S\backslash\Gamma \Rightarrow \mathbf{w} \in \text{int } S \Rightarrow \mathbf{w} > 0$ for each component. This immediately leads to $f_v(\mathbf{x}|\mathbf{w}) > 0, \forall v$. As a result, $\Phi(\mathbf{w})$ is continuous on $S\backslash\Gamma$ since it is the ratio of two strictly positive posynomial functions.

We now verify the third property in Zangwill's global convergence theory. At each iteration of CCCP, we have the following two cases to consider:

1. If $\mathbf{w}^{(k-)} \notin \Gamma$, i.e., $\mathbf{w}^{(k-1)}$ is not a stationary point of $\alpha(\mathbf{w})$, then $\mathbf{y}^{(k-1)} \notin \arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$, so we have $\alpha(\mathbf{w}^{(k)}) = f(\mathbf{y}^{(k)}) \geq \hat{f}(\mathbf{y}^{(k)}, \mathbf{y}^{(k-1)}) > \hat{f}(\mathbf{y}^{(k-1)}, \mathbf{y}^{(k-1)}) = f(\mathbf{y}^{(k-1)}) = \alpha(\mathbf{w}^{(k-1)})$, where the first inequality and the third equality comes from (12).
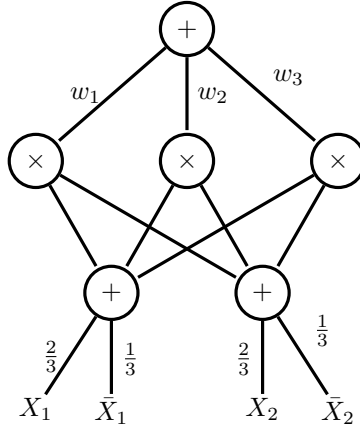
Figure 5: An SPN over two binary random variables.

2. If $\mathbf{w}^{(k-1)} \in \Gamma$, i.e., $\mathbf{w}^{(k-1)}$ is a stationary point of $\alpha(\mathbf{w})$, then $\mathbf{y}^{(k-1)} \in \arg\max_{\mathbf{y}} \hat{f}(\mathbf{y}, \mathbf{y}^{(k-1)})$, so we have $\alpha(\mathbf{w}^{(k)}) = f(\mathbf{y}^{(k)}) \geq \hat{f}(\mathbf{y}^{(k)}, \mathbf{y}^{(k-1)}) = \hat{f}(\mathbf{y}^{(k-1)}, \mathbf{y}^{(k-1)}) = f(\mathbf{y}^{(k-1)}) = \alpha(\mathbf{w}^{(k-1)})$.

By Zangwill's global convergence theory, we now conclude that all the limit points of $\{\mathbf{w}_k\}_{k=1}^{\infty}$ are in Γ and $\alpha(\mathbf{w}_k)$ converges monotonically to $\alpha(\mathbf{w}^*)$ for some stationary point $\mathbf{w}^* \in \Gamma$. ∎

Remark 1. Technically we need to choose $\mathbf{w}_0 \in \text{int } S$ to ensure the continuity of $\Phi(\cdot)$. This initial condition combined with the fact that inside each iteration of CCCP the algorithm only applies positive multiplicative update and renormalization, ensure that after any finite k steps, $\mathbf{w}_k \in \text{int} S$. Theoretically, only in the limit it is possible that some components of $\mathbf{w}_{\infty}$ may become 0. However in practice, due to the numerical precision of float numbers on computers, it is possible that after some finite update steps some of the components in $\mathbf{w}_k$ become 0. So in practical implementation we recommend to use a small positive number ϵ to smooth out such 0 components in $\mathbf{w}_k$ during the iterations of CCCP. Such smoothing may hurt the monotonic property of CCCP, but this can only happens when $\mathbf{w}_k$ is close to $\mathbf{w}^*$ and we can use early stopping to obtain a solution in the interior of S.

Remark 2. Thm. 6 only implies that any limiting point of the sequence $\{\mathbf{w}_k\}_{k=1}^{\infty}(\{\mathbf{y}_k\}_{k=1}^{\infty})$ must be a stationary point of the log-likelihood function and $\{f(\mathbf{y})_k\}_{k=1}^{\infty}$ must converge to some $f(\mathbf{y}^*)$ where $\mathbf{y}^*$ is a stationary point. Thm. 6 does not imply that the sequence $\{\mathbf{w}_k\}_{k=1}^{\infty}(\{\mathbf{y}_k\}_{k=1}^{\infty})$ is guaranteed to converge. Lanckriet and Sriperumbudur [2009] studies the convergence property of general CCCP procedure. Under more strong conditions, i.e., the strict concavity of the surrogate function or that $\Phi()$ to be a contraction mapping, it is possible to show that the sequence $\{\mathbf{w}_k\}_{k=1}^{\infty}(\{\mathbf{y}_k\}_{k=1}^{\infty})$ also converges. However, none of such conditions hold in our case. In fact, in general there are infinitely many fixed points of $\Phi(\cdot)$, i.e., the equation $\Phi(\mathbf{w}) = \mathbf{w}$ has infinitely many solutions in S. Also, for a fixed value t, if $\alpha(\mathbf{w}) = t$ has at least one solution, then there are infinitely many solutions. Such properties of SPNs make it generally very hard to guarantee the convergence of the sequence $\{\mathbf{w}_k\}_{k=1}^{\infty}(\{\mathbf{y}_k\}_{k=1}^{\infty})$. We give a very simple example below to illustrate the hardness in SPNs in Fig. 5. Consider applying the CCCP procedure to learn the parameters on the SPN given in Fig. 5 with three instances $\{(0,1), (1,0), (1,1)\}$. Then if we choose the initial parameter $\mathbf{w}_0$ such that the weights over the indicator variables are set as shown in Fig. 5, then any assignment of (w_1, w_2, w_3) in the probability simplex will be equally optimal in terms of likelihood on inputs. In this example, there are uncountably infinite equal solutions, which invalidates the finite solution set requirement given in [Lanckriet and Sriperumbudur, 2009] in order to show the convergence of $\{\mathbf{w}_k\}_{k=1}^{\infty}$. However, we emphasize that the convergence of the sequence $\{\mathbf{w}_k\}_{k=1}^{\infty}$ is not as important as the convergence of $\{\alpha(\mathbf{w})_k\}_{k=1}^{\infty}$ to desired locations on the log-likelihood surface as in practice any $\mathbf{w}^*$ with equally good log-likelihood may suffice for the inference/prediction task.

17

B   Expectation Maximization

It is clear that an SPN is a generative model and maintains a joint distribution over $\{X_1, \ldots, X_n\}$. However when it was first proposed it was not clear what is the exact joint distribution and probabilistic independence properties encoded in the structure of an SPN. Recent work shows that SPNs are essentially equivalent to bipartite Bayesian networks if we allow more compact data structures, i.e., directed acyclic graphs, to represent local conditional distributions in BNs [Zhao et al., 2015]. This finding leads us to use EM algorithms for training SPNs. It should be noted that although the idea of treating sum nodes in SPNs as hidden variables is not new [Poon and Domingos, 2011, Gens and Domingos, 2012], a correct EM algorithm for learning the parameters of SPNs is not available in the literature because of the misconception about the joint probability distribution represented by SPNs. To the best of our knowledge, the only correct EM algorithm for SPN so far is derived this year, independently, by Peharz [2015] using his augmented SPN argumentation. The main difference lies in the fact that we obtain the algorithm based on the bipartite BN interpretation of an SPN and make a connection to the probabilistic semantics of differential operations in BN [Darwiche, 2003] without seeking to change or augment the original SPN. Hence we provide an alternative, perhaps easier and more concise, derivation of EM algorithm for SPNs.

A latent random variable interpretation of sum nodes in SPNs was pointed out in [Poon and Domingos, 2011], where the authors suggest a hard EM parameter learning algorithm. Later, Gens et al. summarize both the hard and soft versions of the EM algorithm in [Gens and Domingos, 2012, table 2], where the marginal posterior of a hidden variable H is given as

$$\Pr(H = h_k|\mathbf{x}, \mathbf{w}) \propto w_k \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w})} \tag{20}$$

where $\mathcal{S}_H$ is a sub-SPN rooted at a sum node H, interpreted to be a hidden variable. It is then suggested that Eq. 20 should be used for weight updates in soft EM algorithm and a variant of Eq. 20, which assigns all the credit, which is 1, to the winning child of a sum node for hard EM weight updates. We point out here that although the marginal posterior of a hidden variable can indeed be used for weight updates in soft EM algorithm, which will become clear later using the bipartite BN perspective of an SPN, the expression in Eq. 20 is incorrect. A direct and intuitive explanation of Eq. 20 is that the marginal posterior of a hidden variable keeps as same with its prior since the term $\frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w})}{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w})}$ is a constant for all the children of the sum node H. This further means that if we do weight updates in EM based on Eq. 20, the data log-likelihood on training set will keep the same after every EM iteration since the weights of sum nodes keep the same. However, one of the useful properties of EM algorithm applied to generative models lies in the fact that it will improve the training data log-likelihood monotonously until convergence. It should be noted here that the same conclusion has been observed independently by [Peharz, 2015].

We now proceed to derive the EM algorithm for learning the parameters of SPNs based on the bipartite BN perspective. Given an SPN $\mathcal{S}$, we first construct an equivalent bipartite BN as shown in Fig. 6. The joint probability distribution of $\mathcal{S}$ over both observable variables $\mathbf{X}$ and hidden variables
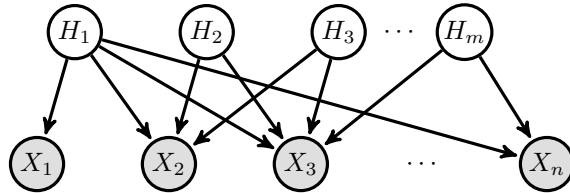

Figure 6: A bipartite BN representation of a given SPN $\mathcal{S}$ with m hidden variables (sum nodes) and n observable variables.

(sum nodes) $\mathbf{H}$ can be written as:

$$\Pr(\mathbf{X}, \mathbf{H}|\mathbf{w}) = \left(\prod_{i=1}^{m} \Pr(H_i|w_i)\right)\left(\prod_{t=1}^{n} \Pr(X_t|Pa(X_t))\right)$$

Consider the expected complete data log-likelihood function

$$\mathcal{Q}(\mathbf{w}, \mathbf{w}^{(k-1)}) \triangleq \mathbb{E}_{\Pr(\mathbf{h}|\mathbf{x}, \mathbf{w}^{(k-1)})}[\log \Pr(\mathbf{x}, \mathbf{h}|\mathbf{w})] \tag{21}$$

Substituting the joint distribution into Eq. 21, we reach

$$\mathcal{Q}(\mathbf{w}, \mathbf{w}^{(k-1)}) = \sum_{i=1}^{m} \sum_{j=1}^{|H_i|} \mathbb{E}_{\Pr(\mathbf{h}|\mathbf{x}, \mathbf{w}^{(k-1)})}[\mathbb{I}_{H_{ij}} \log w_{ij}] \tag{22}$$

where $\mathbb{I}_{H_{ij}}$ is the indicator function of the events "hidden variable H_i takes its jth value". It is worthwhile to point out here that due to the bipartite topology of the BN, the terms including model parameters of hidden variables are independent of each other, which makes the following optimization easier to deal with. Further define

$$\gamma_{ij} \triangleq \mathbb{E}_{\Pr(\mathbf{h}|\mathbf{x}, \mathbf{w}^{(k-1)})}[\mathbb{I}_{H_{ij}}] = \Pr(H_i = j|\mathbf{x}, \mathbf{w}^{(k-1)}) \tag{23}$$

to be the sufficient statistics for model parameter w_{ij}. In order to maximize Eq. 22 with respect to w_{ij}, we augment Eq. 22 using Lagrange multiplier method with the constraints $\sum_j w_{ij} = 1, \forall i$, differentiate the augmented Lagrangian with respect to w_{ij} and then set the derivatives to 0, we achieve the weight update formula for EM algorithm as $w_{ij}^{(k)} = \frac{\gamma_{ij}}{\sum_j \gamma_{ij}}$, where the sufficient statistics is the marginal probability of each hidden variable (sum node) in SPN,

$$\gamma_{ij} = \Pr(H_i = j|\mathbf{x}, \mathbf{w}^{(k-1)}) = \frac{f_{\mathcal{S}}(\mathbf{x}, H_i = j|\mathbf{w}^{(k-1)})}{f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})} \tag{24}$$

where the denominator is simply the value of the SPN $\mathcal{S}$ with current model parameters $\mathbf{w}^{(k-1)}$ given $\mathbf{x}$ as input. To compute the numerator $f_{\mathcal{S}}(\mathbf{x}, H_i = j|\mathbf{w}^{(k-1)})$, we need to introduce a theorem about the probabilistic semantics of differentiation in Bayesian network:

Theorem 8 ([Darwiche, 2003]). Let $\mathcal{B}$ be a Bayesian network representing probability distribution Pr with network polynomial function $f_{\mathcal{B}}$. For every variable H and evidence $\mathbf{x}$, we have

$$\frac{\partial f_{\mathcal{B}}}{\partial \mathbb{I}_{H_j}}(\mathbf{x}) = \Pr(H = j, \mathbf{x} \backslash H)$$

where $\mathbf{x} \backslash H$ means excluding variable H from evidence $\mathbf{x}$. For an example, for evidence $\mathbf{x} = \mathbf{x}_{1:n}$, $\frac{\partial f_{\mathcal{B}}(\mathbf{x}_{1:n})}{\partial \mathbb{I}_{X_{i,j}}} = \Pr(X_i = j, \mathbf{x}_{1:i-1}, \mathbf{x}_{i+1:n})$. As another example, for $\mathbf{x} = \mathbf{x}_{1:n}$ and $H_i \notin \mathbf{X}_{1:n}$, we have $\frac{\partial f_{\mathcal{B}}(\mathbf{x}_{1:n})}{\partial \mathbb{I}_{H_{ij}}} = \Pr(H_i = j, \mathbf{x}_{1:n})$. Thm. 8 is particularly useful here to compute the numerator in Eq. 24, which gives us

$$\Pr(\mathbf{x}, H_i = j|\mathbf{w}^{(k-1)}) = \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial \mathbb{I}_{H_{ij}}}$$

where

$$\frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial \mathbb{I}_{H_{ij}}} = \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w}^{(k-1)})} \frac{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial \mathbb{I}_{H_{ij}}}$$

$$= w_{ij} \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w}^{(k-1)})} f_{\mathcal{S}_{H_{ij}}}(\mathbf{x}|\mathbf{w}^{(k-1)}) \tag{25}$$

Combine all the above results, the formula for computing the sufficient statistics γ_{ij} is given by

$$\gamma_{ij} \propto w_{ij}^{(k-1)} f_{\mathcal{S}_{H_{i,j}}}(\mathbf{x}|\mathbf{w}^{(k-1)}) \frac{\partial f_{\mathcal{S}}(\mathbf{x}|\mathbf{w}^{(k-1)})}{\partial f_{\mathcal{S}_H}(\mathbf{x}|\mathbf{w}^{(k-1)})} \tag{26}$$

Comparing γ_{ij} in EM algorithm with the update formula Eq. 17 in CCCP, they share exactly the same form. Although the development of CCCP and EM for SPNs comes from different perspectives, where CCCP treats SPNs as deep models and optimize the transformed signomial program while EM treats SPNs as a special kind of probabilistic graphical models and optimize the expected complete data log-likelihood. This is not a coincidence. In fact, both CCCP and EM are special instances of a more general bound optimization framework frequently used in statistics and machine learning, known as minorization maximization (MM) or majorization minimization depending on the context [Hunter and Lange, 2004].