Online Algorithms for Sum-Product Networks with Continuous Variables

Priyank Jaini[1] PJAINI@UWATERLOO.CA
Abdullah Rashwan[1] ARASHWAN@UWATERLOO.CA
Han Zhao[2] HAN.ZHAO@CS.CMU.EDU
Yue Liu[1] Y565LIU@UWATERLOO.CA
Ershad Banijamali[1] SBANIJAMALI@UWATERLOO.CA
Zhitang Chen[3] CHENZHITANG2@HUAWEI.COM
Pascal Poupart[1] PPOUPART@UWATERLOO.CA
[1]*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada*
[2]*Machine Learning Dept., School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA*
[3]*Noah's Ark Laboratory, Huawei Technologies, Hong Kong, China*

Abstract

Sum-product networks (SPNs) have recently emerged as an attractive representation due to their dual interpretation as a special type of deep neural network with clear semantics and a tractable probabilistic graphical model. We explore online algorithms for parameter learning in SPNs with continuous variables. More specifically, we consider SPNs with Gaussian leaf distributions and show how to derive an online Bayesian moment matching algorithm to learn from streaming data. We compare the resulting generative models to stacked restricted Boltzmann machines and generative moment matching networks on real-world datasets.

Keywords: Sum-product networks; continuous variables; online learning.

1. Introduction

Sum-product networks (SPNs) (Poon and Domingos, 2011) recently emerged as an attractive representation that can be seen as both a deep architecture and a tractable probabilistic graphical model. More precisely, SPNs are deep neural networks restricted to sum and product operators. While this limits their expressiveness, it allows us to attribute clear semantics to the nodes by interpreting the sub-SPN rooted at each node as a distribution over a subset of variables (under suitable conditions). In that sense, SPNs are also a type of probabilistic graphical model (PGM). In fact, SPNs are equivalent to arithmetic circuits (ACs) (Darwiche, 2003; Rooshenas and Lowd, 2014) and SPNs can be converted into Bayesian networks (BNs) (Zhao et al., 2015) (and vice-versa). SPNs (and ACs) differ from traditional PGMs such as BNs and Markov networks (MNs) from a computational perspective. SPNs (and ACs) are both a representation and an inference machine that allows exact inference to be performed in linear time with respect to the size of the network. This is particularly advantageous when learning a PGM from data since inference can always be performed exactly and tractably when the resulting PGM is an SPN or AC, but not when it is a BN or MN (unless the search space is restricted to a subclass of tractable PGMs such as bounded tree-width PGMs). Based on those advantages, SPNs have been used in many applications including image completion (Poon and Domingos, 2011), speech modeling (Peharz et al., 2014) and language modeling (Cheng et al., 2014).

To date, most of the algorithms for SPNs assume that the variables of interest are categorical. However, many real-world applications are best modeled by continuous variables. Hence, there is a need for SPN algorithms that work with continuous data. Several papers mention that it is possible to generalize SPNs to continuous variables by considering leaf distributions over continuous variables (Poon and Domingos, 2011; Gens and Domingos, 2012, 2013; Zhao et al., 2015), but only a recent paper by Peharz et al. (2016) proposes an EM algorithm for continuous SPNs and reports experiments with continuous data.

With the rise of large and streaming datasets, there is also a need for online algorithms. In this work, we investigate online algorithms for SPNs with Gaussian leaf distributions. More precisely, we describe how to learn the parameters of such continuous SPNs by extending the online Bayesian moment matching and online EM algorithms proposed by Rashwan et al. (2016) from categorical to Gaussian SPNs. We evaluate the algorithms with real-world datasets and compare their effectiveness to stacked restricted Boltzmann machines (Salakhutdinov and Hinton, 2009) and generative moment matching networks (Li et al., 2015), which are other types of generative deep neural networks.

2. Background

A sum-product network (SPN) (Poon and Domingos, 2011) is a rooted acyclic directed graph whose internal nodes are sums and products, and leaves are tractable distributions over some random variables. The edges linking each sum node to its children are labeled with non-negative weights. SPNs are equivalent to arithmetic circuits (Darwiche, 2003), which also consist of rooted acyclic directed graphs of sums and products, with the only difference that edges do not have weights, while leaves store numerical values. This is only a syntactic difference since edge weights in SPNs can always be transformed into leaves with corresponding numerical values in ACs (and vice-versa) (Rooshenas and Lowd, 2014).

The value of an SPN is the value computed by its root in a bottom up pass. More specifically, $V_{node}(\mathbf{x})$ denotes the value computed by a node based on evidence $\mathbf{x}$.

$$V_{node}(\mathbf{x}) = \begin{cases} p_{node}(\mathbf{x}) & node \text{ is a leaf} \\ \sum_{c \in children(node)} w_c V_c(\mathbf{x}) & node \text{ is a sum} \\ \prod_{c \in children(node)} V_c(\mathbf{x}) & node \text{ is a product} \end{cases} \tag{1}$$

Here, $p_{node}(\mathbf{x})$ denotes the probability density or mass of the evidence $\mathbf{x}$ in the distribution at a leaf node, depending on whether $\mathbf{x}$ is continuous or discrete. When the evidence is empty or does not instantiate any of the variables in the distribution at a leaf node then $p_{node}(\mathbf{x}) = 1$, which is equivalent to marginalizing out all the variables for which we do not have any evidence.

Under suitable conditions, an SPN (or AC) encodes a joint distribution over a set of variables. Let $scope(node)$ be the set of variables in the leaves of the sub-SPN rooted at a node.

Definition 1 (Completeness/smoothness (Darwiche, 2002; Poon and Domingos, 2011)) *A sum node is complete/smooth when the scopes of its children are the same (i.e.,* $\forall c, c' \in children(sum)$*,* $scope(c) = scope(c')$*).*

Definition 2 (Decomposability (Darwiche, 2002; Poon and Domingos, 2011)) *A product node is decomposable when the scopes of its children are disjoint (i.e.,* $\forall c, c' \in children(sum)$ *and* $c \neq c', scope(c) \wedge scope(c') = \emptyset$*).*

When all sum nodes are complete/smooth and all product nodes are decomposable, an SPN is called valid, which ensures that the value computed by the SPN for some evidence is proportional to the probability of that evidence (i.e., $\Pr(\mathbf{x}) \propto V_{root}(\mathbf{x})$). This means that we can answer inference queries $\Pr(\mathbf{x}_1|\mathbf{x}_2)$ by computing two SPN evaluations $V_{root}(\mathbf{x}_1)$ and $V_{root}(\mathbf{x}_2)$:

$$\Pr(\mathbf{x}_1|\mathbf{x}_2) = \frac{V_{root}(\mathbf{x}_1, \mathbf{x}_2)}{V_{root}(\mathbf{x}_2)} \tag{2}$$

Since each SPN evaluation consists of a bottom up pass that is linear in the size of the network, the complexity of inference in SPNs is linear in the size of the SPN (same holds for ACs). In contrast, inference may be exponential in the size of the network for BNs and MNs.

We can interpret SPNs as hierarchical mixture models since each sum node can be thought as taking a mixture of the distributions encoded by its children, where the probability of each child component is proportional to the weight labeling the edge of that child. Completeness/smoothness ensures that each child is a distribution over the same set of variables. Similarly, we can think of each product node as factoring a distribution into a product of marginal distributions. Decomposability ensures that the marginal distributions are independent.

Several algorithms have been proposed to learn the structure and parameters of SPNs. Structure learning is important since it is difficult to specify a structure by hand that satisfies the completeness/smoothness and decomposability properties. Structure learning algorithms for SPNs are generally based on clustering techniques that greedily construct a structure by partitioning/grouping the data and the variables (Dennis and Ventura, 2012; Gens and Domingos, 2013; Peharz et al., 2013; Rooshenas and Lowd, 2014; Adel et al., 2015; Vergari et al., 2015).

Parameter learning techniques for SPNs include maximum likelihood by gradient ascent (Poon and Domingos, 2011), expectation maximization (Poon and Domingos, 2011; Peharz, 2015; Peharz et al., 2016) and signomial programming (Zhao and Poupart, 2016), as well as approximate Bayesian learning by moment matching (Rashwan et al., 2016) and collapsed variational inference (Zhao et al., 2016). Among those algorithms, gradient ascent, expectation maximization, moment matching and collapsed variational inference can learn in an online fashion with streaming data (Rashwan et al., 2016; Zhao et al., 2016).

2.1 Moment Matching

Since we will extend the Bayesian moment matching technique by Rashwan et al. (2016) to continuous SPNs with Gaussian leaves, we review some background about moment matching. In SPNs the parameters that need to be learned are the weights associated with each sum node. In Gaussian SPNs, the mean and covariance matrix of each leaf node also need to be learned. Let $\boldsymbol{\theta}$ denote the set of parameters that need to be learned based on some data $\mathbf{x}$. Bayesian learning proceeds by computing the posterior $\Pr(\boldsymbol{\theta}|\mathbf{x})$. When the posterior is intractable, we can approximate it by a simpler distribution after processing each data point x. This simpler distribution is chosen to match a subset of the moments of the exact intractable posterior, hence the name Bayesian moment matching, which is a form of assumed density filtering (Minka and Lafferty, 2002).

A moment is a quantitative measure of the shape of a distribution or a set of points. Let $f(\boldsymbol{\theta}|\boldsymbol{\phi})$ be a probability density function over a d-dimensional random variable $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_d\}$ parametrized by $\boldsymbol{\phi}$. The j^{th} order moments of $\boldsymbol{\theta}$ are defined as $M_{g_j(\boldsymbol{\theta})}(f) = \mathbb{E}_f\left[\prod_i \theta_i^{n_i}\right]$ where

$\sum_i n_i = j$ and g_j is a monomial of $\boldsymbol{\theta}$ of degree *j*.

$$M_{g_j(\boldsymbol{\theta})}(f) = \int_{\boldsymbol{\theta}} g_j(\boldsymbol{\theta}) f(\boldsymbol{\theta}|\boldsymbol{\phi}) d\boldsymbol{\theta}$$

For some distributions *f*, there exists a set of monomials *S(f)* such that knowing $M_g(f)\ \forall g \in S(f)$ allows us to calculate the parameters of *f*. For example, for a Gaussian distribution $\mathcal{N}(x; \mu, \sigma^2)$, the set of sufficient moments *S(f)* = $\{x, x^2\}$. This means knowing M_x and M_{x^2} allows us to estimate the parameters μ and σ^2 that characterize the distribution. The method of moments is used in the Bayesian Moment Matching (BMM) algorithm.

The method of moments is a popular frequentist technique used to estimate the parameters of a probability distribution based on the evaluation of the empirical moments of a dataset. It has been previously used to estimate the parameters of latent Dirichlet allocation, mixture models and hidden Markov models (Anandkumar et al., 2012). The method of moments or moment matching can also be used in a Bayesian setting to compute a subset of the moments of an intractable posterior distribution. Subsequently, another distribution with the same moments is selected from a tractable family of distributions. Consider a Gaussian mixture model, which is a simple SPN where the root is a sum linked to a layer of product nodes that are each linked to Gaussian leaf distributions. We use the Dirichlet as a prior over the weights of the mixture and a Normal-Wishart distribution as a prior over each Gaussian component. We next give details about the Dirichlet and Normal-Wishart distributions, including their set of sufficient moments.

2.2 Family of Prior Distributions

In Bayesian Moment Matching, we project the posterior onto a tractable family of distributions by matching a set of sufficient moments. To ensure scalability, it is desirable to start with a family of distributions that are conjugate priors for multinomial distributions (for the set of weights) and Gaussian distributions with unknown mean and covariance matrix. The product of a Dirichlet distribution over the weights with a Normal-Wishart distribution over the mean and covariance matrix of each Gaussian component ensures that the posterior is a mixture of products of Dirichlet and Normal-Wishart distributions. Subsequently, we can approximate this mixture in the posterior with a single product of Dirichlet and Normal-Wishart distributions by using moment matching. We explain this in greater detail in Section 3, but first we describe briefly the Dirichlet and Normal-Wishart distribution along with some sets of sufficient moments.

2.2.1 DIRICHLET DISTRIBUTION

The Dirichlet distribution is a family of multivariate continuous probability distributions over the probability simplex. It is the conjugate prior probability distribution for the multinomial distribution and hence it is a natural choice of prior over the set of weights $\mathbf{w} = \{w_1, w_2, ..., w_M\}$ of a Gaussian mixture model. A set of sufficient moments for the Dirichlet distribution is $S = \{(w_i, w_i^2) : \forall i \in \{1, 2, ..., M\}\}$. Let $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_M\}$ be the parameters of the Dirichlet distribution over $\mathbf{w}$, then

$$\mathbb{E}[w_i] = \frac{\alpha_i}{\sum_j \alpha_j} \quad \forall i \in \{1, 2, ..., M\}$$

$$\mathbb{E}[w_i^2] = \frac{\alpha_i(\alpha_i + 1)}{\left(\sum_j \alpha_j\right)\left(1 + \sum_j \alpha_j\right)} \quad \forall i \in \{1, 2, ..., M\} \tag{3}$$

2.2.2 NORMAL WISHART PRIOR

The Normal-Wishart distribution is a multivariate distribution with four parameters. It is the conjugate prior of a multivariate Gaussian distribution with unknown mean and covariance matrix. This makes a Normal-Wishart distribution a natural choice for the prior over the unknown mean and precision matrix for our case.

Let $\boldsymbol{\mu}$ be a d-dimensional vector and $\boldsymbol{\Lambda}$ be a symmetric positive definite $d \times d$ matrix of random variables respectively. Then, a Normal-Wishart distribution over $(\boldsymbol{\mu}, \boldsymbol{\Lambda})$ given parameters $(\boldsymbol{\mu}_0, \kappa, \mathbf{W}, \nu)$ is such that $\boldsymbol{\mu} \sim \mathcal{N}_d\big(\boldsymbol{\mu}; \boldsymbol{\mu}_0, (\kappa\boldsymbol{\Lambda})^{-1}\big)$ where $\kappa > 0$ is real, $\boldsymbol{\mu}_0 \in \mathbb{R}^d$ and $\boldsymbol{\Lambda}$ has a Wishart distribution given as $\boldsymbol{\Lambda} \sim \mathcal{W}(\boldsymbol{\Lambda}; \mathbf{W}, \nu)$ where $\mathbf{W} \in \mathbb{R}^{d\times d}$ is a positive definite matrix and $\nu > d-1$ is real. The marginal distribution of $\boldsymbol{\mu}$ is a multivariate t-distribution i.e $\boldsymbol{\mu} \sim t_{\nu-d+1}\big(\boldsymbol{\mu}; \boldsymbol{\mu}_0, \frac{\mathbf{W}}{\kappa(\nu-d+1)}\big)$. The univariate equivalent for the Normal-Wishart distribution is the Normal-Gamma distribution.

In Section 2.1, we defined *S* as a set of sufficient moments to characterize a distribution. In the case of the Normal-Wishart distribution, we would require at least four different moments to estimate the four parameters that characterize it. A set of sufficient moments in this case is $S = \{\boldsymbol{\mu}, \boldsymbol{\mu}\boldsymbol{\mu}^T, \boldsymbol{\Lambda}, \Lambda_{ij}^2\}$ where Λ_{ij}^2 is the $(i,j)^{th}$ element of the matrix $\boldsymbol{\Lambda}$. The expressions for sufficient moments are given by

$$\begin{aligned}
\mathbb{E}[\boldsymbol{\mu}] &= \boldsymbol{\mu}_0 \\
\mathbb{E}[(\boldsymbol{\mu}-\boldsymbol{\mu}_0)(\boldsymbol{\mu}-\boldsymbol{\mu}_0)^T] &= \frac{\kappa+1}{\kappa(\nu-d-1)}\mathbf{W}^{-1} \\
\mathbb{E}[\boldsymbol{\Lambda}] &= \nu\mathbf{W} \\
Var(\Lambda_{ij}) &= \nu(W_{ij}^2 + W_{ii}W_{jj})
\end{aligned} \tag{4}$$

3. Bayesian Moment Matching

We now discuss in detail the Bayesian Moment Matching (BMM) algorithm. Since Bayesian learning yields a posterior that consists of a mixture distribution with an exponentially growing number of mixture components, BMM approximates the posterior after each observation with fewer components in order to keep the computation tractable.

In Algorithm 1, we first describe a generic procedure to approximate the posterior P_n after each observation with a simpler distribution $\tilde{P_n}$ by moment matching. More precisely, a set of moments sufficient to define $\tilde{P_n}$ are matched with the moments of the exact posterior P_n. At each iteration, we first calculate the exact posterior $P_n(\boldsymbol{\Theta}|\mathbf{x}^{1:n})$. Then, we compute the set of moments *S(f)* that are sufficient to define a distribution in the family $f(\boldsymbol{\Theta}|\boldsymbol{\Phi})$. Next, we compute the parameter vector $\boldsymbol{\Phi}$ based on the set of sufficient moments. This determines a specific distribution $\tilde{P_n}$ in the family f that we use to approximate P_n. Note that the moments in the sufficient set *S(f)* of the approximate posterior are the same as that of the exact posterior. However, all the other moments outside this set of sufficient moments *S(f)* may not necessarily be the same.

3.1 BMM for multivariate Gaussian mixture model

In the previous section, we gave a generic version of the Bayesian moment matching algorithm. In this section, we briefly discuss an instantiation of the algorithm for a multivariate Gaussian mixture model, which is a simple 3-layer SPN consisting of a sum (top layer/root) of products (middle layer)

Algorithm 1 Generic Bayesian Moment Matching

```
**Input:** Data $\mathbf{x}^n$, $n \in \{1, 2, ..., N\}$
Let $f(\mathbf{\Theta}|\mathbf{\Phi})$ be a family of probability distributions with parameters $\mathbf{\Phi}$
Initialize a prior $\tilde{P}_0(\mathbf{\Theta})$
**for** $n = 1$ **to** $N$ **do**
    Compute $P_n(\mathbf{\Theta}|\mathbf{x}^{1:n})$ from $\tilde{P}_{n-1}(\mathbf{\Theta}|\mathbf{x}^{1:n-1})$ using Bayes' theorem
    $\forall\ g(\mathbf{\Theta}) \in S(f)$, evaluate $M_{g(\mathbf{\Theta})}(P_n)$
    Compute $\mathbf{\Phi}$ using $M_{g(\mathbf{\Theta})}(P_n)$'s
    Approximate $P_n$ with $\tilde{P}_n(\mathbf{\Theta}|\mathbf{x}^{1:n}) = f(\mathbf{\Theta}|\mathbf{\Phi})$
**end for**
Return $\hat{\mathbf{\Theta}} = \mathbb{E}_{\tilde{P}_n}[\mathbf{\Theta}]$
```

of multivariate Gaussians (bottom layer/leaves). The family of distributions for the prior in this case becomes $P_0(\mathbf{\Theta}) = Dir(\mathbf{w}|\mathbf{a}) \prod_{i=1}^{M} \mathcal{NW}_d(\boldsymbol{\mu}_i, \mathbf{\Lambda}_i|\boldsymbol{\alpha}_i, \kappa_i, \mathbf{W}_i, \nu_i)$ where $\mathbf{w} = (w_1, w_2, ..., w_M)$ and $\mathbf{a} = (a_1, a_2, ..., a_M)$. The set of sufficient moments for the posterior in this case would be given by $S(P(\mathbf{\Theta}|\mathbf{x})) = \{\boldsymbol{\mu}_i, \boldsymbol{\mu}_i\boldsymbol{\mu}_i^T, \mathbf{\Lambda}_i, \Lambda_{i_{kl}}^2, w_i, w_i^2 : \forall i \in 1, 2, .., M\}$ where $\Lambda_{i_{kl}}$ is the $(k, l)^{th}$ element of the matrix $\mathbf{\Lambda}_i$. Notice that, since $\mathbf{\Lambda}_i$ is a symmetric matrix, we only need to consider the moments of the elements on and above the diagonal of $\mathbf{\Lambda}_i$.

In Eq. 4 of Section 2.2.2, we presented the expressions for a set of sufficient moments of a Normal-Wishart distribution. Using those expressions we can again approximate a mixture of products of Dirichlet and Normal-Wishart distributions in the posterior with a single product of Dirichlet and Normal-Wishart distributions, as we did in the previous section. Finally, the estimate $\mathbf{\Theta} = \mathbb{E}_{\tilde{P}_n}[\mathbf{\Theta}]$ is obtained after observing the data $\mathbf{x}^{1:n}$.

3.2 Bayesian Moment Matching for Continuous SPNs

In this work, we consider SPNs that are trees (i.e., each node has a single parent) since many structure learning algorithms produce SPNs that are trees. When an SPN is a tree, it is possible to compute all the moments simultaneously in time that is linear in the size of the network. Two coefficients $coef_i^0, coef_i^1$ are computed at each node i in a bottom-up and top-down pass of the network. Algorithm 2 shows how those coefficients are computed. Once we have the coefficients, we can compute each moment as follows. For leaf nodes that model multivariate Gaussian distributions, the set of sufficient moments is $S_i = \{\boldsymbol{\mu}_i, \boldsymbol{\mu}_i\boldsymbol{\mu}_i^T, \mathbf{\Lambda}_i, \Lambda_{i_{kl}}^2\}$ $\forall i \in leafNodes$ and we get:

$$M_{P(\mathbf{\Theta}|\mathbf{x})}(s) = \int_s s\mathcal{NW}_i(\mu_i, \Lambda_i|\alpha_i, \kappa_i, W_i, \nu_i)\Big(coef_i^0 + coef_i^1\mathcal{N}(x|\mu_i, \Lambda_i^{-1})\Big)ds \ \ \forall s \in S_i$$

For the rest of the nodes:

$$M_{P(\mathbf{\Theta}|\mathbf{x})}(w_{ij}^k) = \int_{w_{i\cdot}} w_{ij}^k Dir(\mathbf{w}_{i\cdot}|\boldsymbol{\alpha}_{i\cdot})\Big(coef_i^0 + coef_i^1 \sum_{j'} w_{ij'}V_{j'}(\mathbf{x})\Big)dw_{i\cdot} \ \forall i \in sumNodes$$

At a high level, Alg. 2 maintains a posterior distribution over all the sum nodes in the subtree of a given node. This is achieved by a bottom-up pass of the network to compute all the $v_i(\mathbf{x})$ at each node followed by a top-down recursive pass to compute the required moments at all the sum nodes.

Algorithm 2 $computeCoefficients(node)$ based on $\mathbf{x}$ and weight prior $\prod_i Dir(w_{i\cdot}|\alpha_{i\cdot})$

```
1: if $isRoot(node)$ then
2:     $coef^0_{node} \leftarrow 0$
3:     $coef^1_{node} \leftarrow 1$
4: else if $isProduct(parent(node))$ then
5:     $coef^0_{node} \leftarrow coef^0_{parent}$
6:     $coef^1_{node} \leftarrow coef^1_{parent} \prod_{sibling} v_{sibling}(\mathbf{x})$
7: else if $isSum(parent(node))$ then
8:     $coef^0_{node} \leftarrow coef^0_{parent} + coef^1_{parent} \sum_{sibling} \frac{\alpha_{parent,sibling}}{\sum_j \alpha_{parent,j}} v_{sibling}(\mathbf{x})$
9:     $coef^1_{node} \leftarrow coef^1_{parent} \frac{\alpha_{parent,node}}{\sum_j \alpha_{parent,j}}$
10: end if
11: if $isNotLeaf(node)$ then $computeCoefficients(child)$ $\forall child$
```

In Alg. 2, $v_i(\mathbf{x})$ is defined as:

$$v_i(\mathbf{x}) = \int_{\mathbf{w}} \prod_j Dir(\mathbf{w}_j|\boldsymbol{\alpha}_j) V_i(\mathbf{x})\, d\mathbf{w} \tag{5}$$

where the integration is over all the model parameters $\mathbf{w}$ that appear in the sub-tree rooted at node i, and upper case $V_i(\mathbf{x})$ is the value computed at node i when the input instance is $\mathbf{x}$. Note that by utilizing the same network structure, all the $v_i(\mathbf{x})$ can be computed in a single bottom-up pass.

To see the correctness of Alg. 2, let r be the root node, p be a child product node of r, and s is a child sum node of p, we will show how to compute the posterior as we go down the tree while integrating the rest of the network above a certain node. We can write down the posterior as follows:

$$P(\mathbf{w}|\mathbf{x}) \propto \prod_{i \in subtree(r)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i) \left(coef^0_r + coef^1_r V_r(\mathbf{x})\right) \tag{6}$$

The base case is given by setting $coef^0_r = 0$ and $coef^1_r = 1$ at the root. We then expand $V_r(\mathbf{x})$ as:

$$V_r(\mathbf{x}) = w_{r,p} V_p(\mathbf{x}) + \sum_{s \in siblings(p)} w_{r,s} V_s(\mathbf{x}) \tag{7}$$

Substituting $V_r(\mathbf{x})$ into $P(\mathbf{w}|\mathbf{x})$, we get:

$$P(\mathbf{w}|\mathbf{x}) \propto \prod_{i \in subtree(r)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i) \left(coef^0_r + coef^1_r \left(w_{r,p} V_p(\mathbf{x}) + \sum_{i \in siblings(p)} w_{r,i} V_i(\mathbf{x})\right)\right) \tag{8}$$

As we go down one level in the tree, we integrate $\mathbf{w}_r$ and the weights in all sibling subtrees of p:

$$\begin{aligned} &P(\{\mathbf{w}_i\}_{i \in subtree(p)}|\mathbf{x}) \\ &\propto \prod_{i \in subtree(p)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i) \left(coef^0_r + coef^1_r \left(\frac{\alpha_{r,p}}{\sum_i \alpha_{r,i}} V_p(\mathbf{x}) + \frac{\sum_{i \in siblings(p)} \alpha_{r,i} v_i(\mathbf{x})}{\sum_i \alpha_{r,i}}\right)\right) \\ &= \prod_{i \in subtree(p)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i) \left(\left(coef^0_r + coef^1_r \frac{\sum_{i \in siblings(p)} \alpha_{r,i} v_i(\mathbf{x})}{\sum_i \alpha_{r,i}}\right) + \left(coef^1_r \frac{\alpha_{r,p}}{\sum_i \alpha_{r,i}}\right) V_p(\mathbf{x})\right) \end{aligned}$$

where v_i is the value when integrating the weights. Let $coef_p^0 = coef_r^0 + coef_r^1 \frac{\sum_{i \in siblings(p)} \alpha_{r,i} v_i(\mathbf{x})}{\sum_i \alpha_{r,i}}$ and $coef_p^1 = coef_r^1 \frac{\alpha_{r,p}}{\sum_i \alpha_{r,i}}$, then we can write down $P(\mathbf{w}|\mathbf{x})$ as follows:

$$P(\{\mathbf{w}_i\}_{i \in subtree(p)}|\mathbf{x}) \propto \prod_{i \in subtree(p)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i)\left(coef_p^0 + coef_p^1 V_p(\mathbf{x})\right) \tag{9}$$

Another step down to sum node s by expanding $V_p(\mathbf{x})$:

$$P(\{\mathbf{w}_i\}_{i \in subtree(p)}|\mathbf{x}) \propto \prod_{i \in subtree(p)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i)\left(coef_p^0 + coef_p^1\left(V_s(\mathbf{x}) \prod_{i \in siblings(s)} V_i(\mathbf{x})\right)\right) \tag{10}$$

Again, we can integrate all weights of the sibling subtrees of node s, we get:

$$P(\{\mathbf{w}_i\}_{i \in subtree(s)}|\mathbf{x}) \propto \prod_{i \in subtree(s)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i)\left(coef_p^0 + \left(coef_p^1 \prod_{i \in siblings(s)} v_i(\mathbf{x})\right) V_s(\mathbf{x})\right) \tag{11}$$

Let $coef_s^0 = coef_p^0$ and $coef_s^1 = coef_p^1 \prod_{i \in siblings(s)} v_i(\mathbf{x})$, we get:

$$P(\{\mathbf{w}_i\}_{i \in subtree(s)}|\mathbf{x}) \propto \prod_{i \in subtree(s)} Dir(\mathbf{w}_i|\boldsymbol{\alpha}_i)\left(coef_s^0 + coef_s^1 V_s(\mathbf{x})\right) \tag{12}$$

The previous steps can be done recursively along each path from root to leaf since the underlying network structure is assumed to be a tree.

3.3 Distributed Bayesian Moment Matching

One of the major advantages of Bayes' theorem is that the computation of the posterior can be distributed over several machines, each of which processes a subset of the data. It is also possible to compute the posterior in a distributed manner using Bayesian moment matching algorithm. For example, let us assume that we have *T* machines and a data set with *TN* data points. Each machine t, can compute the approximate posterior $P_t(\mathbf{\Theta}|\mathbf{x}^{(t-1)N+1:tN})$ where $t \in 1, 2, .., T$ using Algorithm 1 over *N* data points. These partial posteriors $\{P_t\}_{t=1}^T$ can be combined to obtain a posterior over the entire data set $\mathbf{x}^{1:TN}$ according to the following equation:

$$P(\mathbf{\Theta}|\mathbf{x}^{1:TN}) \propto P(\mathbf{\Theta}) \prod_{t=1}^{T} \frac{P_t(\mathbf{\Theta}|\mathbf{x}^{(t-1)N+1:tN})}{P(\mathbf{\Theta})} \tag{13}$$

Subsequently, the estimate $\hat{\mathbf{\Theta}} = \mathbb{E}_{P(\mathbf{\Theta}|\mathbf{x}^{1:TN})}[\mathbf{\Theta}]$ is obtained over the whole data set. Therefore, we can use the Bayesian moment matching algorithm to perform Bayesian learning in an online and distributed fashion. Note that there is no loss incurred by distributing the computation of the posterior. Eq. 13 is exact and simply follows from Bayes' theorem and the assumption that data instances are sampled independently from some underlying distribution.

4. Related Work

The expectation maximization (EM) algorithm for estimating the parameters of SPNs, including both the edge weights as well as the parameters of the leaf distributions, has recently been proposed by Peharz et al. (2016). In the E-step, the authors apply the differential approach developed in Darwiche (2003) to efficiently propagate the sufficient statistics for both edge parameters as well as leaf parameters in time linear in the size of the network. In the M-step, marginalization can be done in closed form for edge parameters, where the edge parameters incident to each sum node form a categorical distribution. When the leaf distributions are chosen from the exponential family, the update formula in the M-step for the leaf parameters can be computed by integration over the natural parameters of the corresponding distribution. When leaf distributions are Gaussians, as the case in this paper, a closed form update formula can be obtained as well. We refer the readers to Peharz et al. (2016) for more detailed discussions of batch EM for SPNs with continuous leaf nodes.

Most deep neural networks are function approximators without any generative capability and cannot be interpreted as probabilistic graphical models. However, Li et al. (2015) recently introduced Generative Moment Matching Networks (GenMMNs), which are deep neural networks that take as input random values and output samples mimicking a desired dataset. Hence those networks can be viewed as generative models that can approximate sampling from the underlying distribution of any dataset. GenMMNs are trained by the so-called empirical maximum mean discrepancy (MMD) (Gretton et al., 2006) between the outputs of the network and a training set. Using the kernel trick, MMD measures the maximum discrepancy between all empirical moments of a dataset and the generative model encoded by a neural network. This is a form of frequentist moment matching that leads to an objective that can be optimized by backpropagation (Li et al., 2015).

5. Experiments

We performed experiments on 7 real-world datasets from the UCI machine learning repository (Lichman, 2013) and the function approximation repository (Guvenir and Uysal, 2000) that span diverse domains with 3 to 48 attributes. We compare the performance of online algorithms for Sum-Product Networks with continuous variables as well as other generative deep networks like Stacked Restricted Boltzman Machines (SRBMs) and Generative Moment Matching Networks (GenMMNs). We evaluated the performance of SPNs using both online EM (oEM) and online Bayesian Moment Matching (oBMM). Stochastic gradient descent and exponentiated gradient descent could also be used, but since their convergence rate was shown to be much slower than that of oEM (Rashwan et al., 2016; Zhao and Poupart, 2016), we do not report results for them. Since there are no structure learning algorithms for continuous SPNs, we used a random structure and a structure equivalent to a Gaussian Mixture Model for the experiments using both oEM and oBMM. In both cases, the leaves are multivariate Gaussian distributions. Before discussing the results, we provide additional details regarding the training and likelihood computation for oEM, SRBM and GenMMN.

Online EM. Peharz et al. (2016) describe how to estimate the leaf parameters in a batch mode by EM. To adapt their algorithm to the online setting, we follow the framework introduced by Neal and Hinton (1998) where increments of the sufficient statistics are accumulated in a streaming fashion. More specifically, we sample one data instance at a time to compute the corresponding local sufficient statistics from that instance. Then we accumulate the local sufficient statistics as increments to the globally maintained sufficient statistics. The update formula is obtained by renormalization according to the global sufficient statistics. This online variant of EM is also known as incremental EM Liang and Klein (2009). There are no hyperparameters to be tuned in the online EM (oEM) algorithm for SPNs. In the experiments, we randomly initialize the weights of SPNs and apply oEM to estimate the parameters. For each data set, oEM processes each data instance exactly once.

SRBM. We trained stacked restricted Boltzmann machine (SRBM) with 3 hidden layers of binary nodes where the first and second hidden layers have the same number of nodes as the dimensionality of the data and the third hidden layer has four times as many nodes as the dimensionality of the data. This yielded a number of parameters that is close to the number of parameters of the random SPNs for a fair comparison. This $1 : 1 : 4$ structure is taken from (Hinton and Salakhutdinov, 2006).Computing the exact likelihood of SRBMs is intractable. Hence, we used Gibbs sampling to approximate the log-likelihood. We averaged 1000 runs of 500 iterations of Gibbs sampling where in each iteration the value of each hidden node is re-sampled given the values of the adjacent nodes. At the end of each run, the log-likelihood of each data instance is computed given the last values sampled for the hidden nodes in the layer above the leaf nodes.

Gen MMN. We trained fully-connected networks with multiple layers. The number of layers and hidden units per layer depends on the training dataset and was selected to match the number of parameters in random SPNs for a fair comparison. Inputs to the networks are vectors of elements drawn independently and uniformly at random in $[-1, 1]$. The activation function for the hidden layers is ReLU and for the output layer is sigmoid. We used the Gaussian kernel in MMD and followed the algorithm described in (Li et al., 2015) for training. To compute the average log-likelihood reported in Table 1, we generate $10,000$ samples using the trained model and fit a kernel density estimator to these samples. The average log-likelihood of the test set is then evaluated using this estimated density. The kernel for our density estimator is Gaussian and its parameter is set to maximize the log-likelihood of the validation set. This technique was used since there is no explicit probability density function for this model. However, as shown in (Theis et al., 2015) this method only provides an approximate estimate of the log-likelihood and therefore the log-likelihood reported for GenMMNs in Table 1 may not be directly comparable to the likelihood of other models.

5.1 Results

Table 1 reports the average log-likelihoods (with standard error) of each model on the 7 real-world datasets using 10-fold cross-validation. No results are reported for random SPNs trained by oBMM and oEM when the number of attributes is too small (less than 5) to consider a structure that is more complex than GMMs. No results are reported for GMM SPNs trained by oBMM and oEM when the number of attributes is large (48) in comparison to the amount of data since the number of parameters to be trained in the covariance matrix of each Gaussian component is square in the number of attributes. We can think of GMMs as a baseline structure for SPNs when the dimensionality of the data is low and random SPNs as more data efficient models for high-dimensional datasets. Note also that since the data is continuous, the log-likelihood can be positive when a model yields a good fit as observed for the Flow Size dataset. The best results for each dataset are highlighted in bold. oBMM consistently outperforms oEM on each dataset. In comparison to other deep models, the results are mixed and simply suggest that continuous SPNs trained by oBMMs are competitive with SRBMs and GenMMNs. Note that it is remarkable that SPNs with a random or basic GMM structure can compete with other deep models in the absence of any structure learning. Note also

Table 1: Log-likelihood scores on real-world data sets. The best results are highlighted in bold font.

Dataset # of vars	Flow Size 3	Quake 4	Banknote 4	Abalone 8	Kinematics 8	CA 22	Sensorless Drive 48
oBMM (random)	-	-	-	-1.82 ± 0.19	-11.19 ± 0.03	-2.47 ± 0.56	**1.58** ± 1.28
oEM (random)	-	-	-	-11.36 ± 0.19	-11.35 ± 0.03	-31.34 ± 1.07	-3.40 ± 6.063
oBMM (GMM)	**4.80** ± 0.67	-3.84 ± 0.16	-4.81 ± 0.13	**-1.21** ± 0.36	-11.24 ± 0.04	**-1.78** ± 0.59	-
oEM (GMM)	-0.49 ± 3.29	-5.50 ± 0.41	-4.81 ± 0.13	-3.53 ± 1.68	-11.35 ± 0.03	-21.39 ± 1.58	-
SRBM	-0.79 ± 0.004	**-2.38** ± 0.01	-2.76 ± 0.001	-2.28 ± 0.001	**-5.55** ± 0.02	-4.95 ± 0.003	-26.91 ± 0.03
GenMMN	0.40 ± 0.007	-3.83 ± 0.21	**-1.70** ± 0.03	-3.29 ± 0.10	-11.36 ± 0.02	-5.41 ± 0.14	-29.41 ± 1.185

that SPNs constitute a tractable probabilistic model in which inference can be done exactly in linear time. This is not the case for SBRMs and GenMMNs where inference must be approximated.

6. Conclusion

We presented the first online algorithm for parameter learning in continuous SPNs with Gaussian leaves. We also reported on the first comparison between continuous SPNs and other generative deep models such as SRBMs and GenMMNs. In the future, we plan to devise a structure learning algorithm for continuous SPNs. Structure learning has the potential to significantly improve the modeling ability of continuous SPNs.

Acknowledgments

This research was funded by a grant from Huawei Noah's Ark Lab in Hong Kong.

References

T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm. In *UAI*, 2015.

A. Anandkumar, D. Hsu, and S. M. Kakade. A method of moments for mixture models and hidden Markov models. In *COLT*, volume 1, page 4, 2012.

W.-C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai. Language modeling with sum-product networks. In *Annual Conf. of the Int. Speech Communication Association*, 2014.

A. Darwiche. A logical approach to factoring belief networks. *KR*, 2:409–420, 2002.

A. Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 50(3):280–305, 2003.

A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *NIPS*, 2012.

R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *NIPS*, pages 3248–3256, 2012.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *ICML*, pages 873–880, 2013.

A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola. A kernel method for the two-sample-problem. In *NIPS*, pages 513–520, 2006.

H. A. Guvenir and I. Uysal. Bilkent university function approximation repository. 2000.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Y. Li, K. Swersky, and R. Zemel. Generative moment matching networks. In *ICML*, pages 1718–1727, 2015.

P. Liang and D. Klein. Online EM for unsupervised models. In *HLT-NAACL*, pages 611–619, 2009.

M. Lichman. UCI machine learning repository, 2013.

T. Minka and J. Lafferty. Expectation-propagation for the generative aspect model. In *UAI*, pages 352–359, 2002.

R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

R. Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Medical University of Graz, 2015.

R. Peharz, B. C. Geiger, and F. Pernkopf. Greedy part-wise learning of sum-product networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 612–627. Springer, 2013.

R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP*, pages 3699–3703, 2014.

R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *arXiv:1601.06180*, 2016.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *UAI*, pages 2551–2558, 2011.

A. Rashwan, H. Zhao, and P. Poupart. Online and Distributed Bayesian Moment Matching for Sum-Product Networks. In *AISTATS*, 2016.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, pages 710–718, 2014.

R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *AISTATS*, pages 448–455, 2009.

L. Theis, A. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv:1511.01844*, 2015.

A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML-PKDD*, pages 343–358. 2015.

H. Zhao and P. Poupart. A unified approach for learning the parameters of sum-product networks. *arXiv:1601.00318*, 2016.

H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and Bayesian networks. In *ICML*, 2015.

H. Zhao, T. Adel, G. Gordon, and B. Amos. Collapsed variational inference for sum-product networks. In *ICML*, 2016.