Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks

Abdullah Rashwan
Computer Science
University of Waterloo
arashwan@uwaterloo.ca

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

Pascal Poupart
Computer Science
University of Waterloo
ppoupart@uwaterloo.ca

Abstract

Probabilistic graphical models provide a general and flexible framework for reasoning about complex dependencies in noisy domains with many variables. Among the various types of probabilistic graphical models, sum-product networks (SPNs) have recently generated some interest because exact inference can always be done in linear time with respect to the size of the network. This is particularly attractive since it means that learning an SPN from data always yields a tractable model for inference. However, existing parameter learning algorithms for SPNs operate in batch mode and do not scale easily to large datasets. In this work, we explore online algorithms to ensure that parameter learning can also be done tractably with respect to the amount of data. More specifically, we propose a new Bayesian moment matching (BMM) algorithm that operates naturally in an online fashion and that can be easily distributed. We demonstrate the effectiveness and scalability of BMM in comparison to online extensions of gradient descent, exponentiated gradient and expectation maximization on 20 classic benchmarks and 4 large scale datasets.

1 Introduction

Sum-Product networks (SPNs) were first proposed by [Poon and Domingos, 2011] as a new type of probabilistic graphical models. An SPN consists of an acyclic directed graph of sums and products that computes a non-linear function of its inputs. SPNs can be viewed as deep neural networks where non-linearity is achieved by products instead of sigmoid, softmax, hyperbolic tangent or rectified linear operations. They also have clear semantics in the sense that they encode a joint distribution over a set of leaf random variables in the form of a hierarchical mixture model. To better understand this distribution, SPNs can be converted into equivalent traditional probabilistic graphical models such as Bayesian networks (BNs) and Markov networks (MNs) by treating sum nodes as hidden variables [Zhao et al., 2015]. An important advantage of SPNs over BNs and MNs is that inference can be done without any approximation in linear time with respect to the size of the network. Hence, SPNs are gaining in popularity as a *tractable* class of probabilistic graphical models. SPNs have been used in image completion tasks [Poon and Domingos, 2011], activity recognition [Amer and Todorovic, 2012, Amer and Todorovic, 2015], speech modeling [Peharz et al., 2014b] and language modeling [Cheng et al., 2014].

A variety of algorithms have been proposed to learn the structure [Rooshenas and Lowd, 2014, Gens and Domingos, 2013] and the parameters [Poon and Domingos, 2011] of SPNs from data. Since existing algorithms operate in a batch way, they do not scale to large datasets. In this work, we explore online algorithms that can process a dataset in one sweep and can continuously refine the parameters of an SPN with a fixed structure based on streaming data. It is relatively easy to extend gradient descent (GD), exponentiated gradient and expectation maximization (EM) to the online setting by restricting the algorithms to a single iteration (i.e., single pass through the data) in which incremental updates to the parameters are performed after processing each data point. Since those algorithms maximize the likelihood of the data, they are subject to overfitting and local optima (the

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

optimization problem is non-convex). Furthermore, restricting the algorithms to a single iteration means that some information is lost. We propose a new Bayesian learning algorithm that does not frame the problem as optimization and therefore does not suffer from local optima. Furthermore, Bayesian learning is robust to overfitting and naturally operates in an online, asynchronous and distributed fashion since Bayes theorem allows us to incrementally update the posterior after each data point and the updates can be computed in any order and distributed on different machines. The main issue with Bayesian learning is that the posterior is often intractable, which is the case for parameter learning in SPNs. To that effect, we propose to approximate the posterior after each update with a tractable distribution that matches some moments of the exact, but intractable posterior. We call this algorithm *online Bayesian Moment Matching* (oBMM). The approximation introduced by moment matching yields a loss of information, however we show that oBMM performs better than online expectation maximization (oEM), online Exponentiated Gradient (oEG), and Stochastic gradient descent (SGD) on a suite of benchmarks and some large language modeling problems. We also demonstrate the scalability of oBMM by distributing the computation on many machines.

The paper is structured as follows. Section 2 reviews SPNs as well as parameter learning algorithms including gradient descent, exponentiated gradient and EM. It also explains how to apply online versions of GD and EM to parameter learning in SPNs. Section 3 describes oBMM to approximate Bayesian learning by moment matching. Section 4 demonstrates the effectiveness and scalability of oBMM in comparison to oEG, oEM, and SGD on a set of benchmarks and four large language modeling problems. Finally, Section 5 concludes and discusses future work.

2 Background

2.1 Sum-Product Networks (SPNs)

Consider a set of random variables $\mathbf{X} = \{X_1, X_2, ..., X_n\}$. A sum-product network (SPN) [Poon and Domingos, 2011] is a probabilistic graphical model that can be used to express a joint distribution over those random variables. An SPN consists of a rooted acyclic directed graph where the interior nodes are sums or products and the leaves are indicator variables (e.g., $I_{X=x}$). There is one indicator variable per variable assignment that returns 1 when the assignment is true and 0 otherwise. The edges emanating from sum nodes are labeled with weights w_{nm} (where n is the source node, m is the destination node, and $w_{nm} > 0$). An SPN can be viewed as a function of the indicator variables that can be evaluated in a bottom up pass. For a joint assignment $\mathbf{e}$ of the indicator variables, the value $V_n(\mathbf{e})$ of a node n is computed recursively as follows:

$$V_n(\mathbf{e}) = \begin{cases} 1 & n \text{ is an indicator set to 1 in } \mathbf{e} \\ 0 & n \text{ is an indicator set to 0 in } \mathbf{e} \\ \prod_{m \in children(n)} V_m(\mathbf{e}) & n \text{ is a product} \\ \sum_{m \in children(n)} w_{nm} V_m(\mathbf{e}) & n \text{ is a sum} \end{cases} \tag{1}$$

An SPN can be used to encode a joint distribution over $\mathbf{X}$, which is defined by the graphical structure and the weights. The probability of a joint assignment $\mathbf{X} = \mathbf{x}$ is proportional to the value at the root of the SPN induced by setting the indicators according to the joint assignment.

$$\Pr(\mathbf{X} = \mathbf{x}) = \frac{V_{root}(\mathbf{e}(\mathbf{x}))}{V_{root}(\mathbf{1})} \tag{2}$$

Here $\mathbf{e}(\mathbf{x})$ indicates that $I_{X_i=x_i} = 1$ when $x_i \in \mathbf{x}$ and 0 otherwise. The normalization constant needed to obtain a probability is $V_{root}(\mathbf{1})$ where $\mathbf{1}$ is a vector of all 1's indicating that all the indicator variables are set to 1. Intuitively, by setting all the indicator variables to 1, the SPN will sum up the contributions of all the joint assignments. Eq. 2 can also be used to compute the marginal probability of a partial assignment $\mathbf{Y} = \mathbf{y}$ as long as we define the indicator assignment $\mathbf{e}(\mathbf{y})$ in such a way that $I_{X_i=x_i} = 1$ when $x_i \in \mathbf{y}$ or $X_i \notin \mathbf{Y}$, and 0 otherwise. This means that the indicators for the variables outside of the partial assignment are all set to 1 since we need to sum out the variables outside of the partial assignment. Conditional probabilities can also be computed by evaluating two partial assignments:

$$\Pr(\mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}) = \frac{\Pr(\mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z})}{\Pr(\mathbf{Z} = \mathbf{z})} \tag{3}$$

$$= \frac{V_{root}(\mathbf{e}(\mathbf{y}, \mathbf{z}))}{V_{root}(\mathbf{e}(\mathbf{z}))} \tag{4}$$

Since joint, marginal and conditional queries can all be answered by two network evaluations, exact inference takes linear time with respect to the size of the network. This is a remarkable property since inference in Bayesian and Markov networks may take exponential time in the size of the network.

An SPN is said to be *valid* when it represents a distribution and Eq. 2 and 4 can be used to answer inference queries correctly [Poon and Domingos, 2011]. *Decomposability* and *completeness* are sufficient conditions that ensure validity. Below we define decomposability and completeness in terms of the *scope* of a node.

Definition 1 (Scope) *The scope of a node n is the set of all variables that appear in the leaf indicators of the sub-SPN rooted at n.*

We can compute the scope of each node in a bottom up pass as follows. If n is a leaf indicator $I_{X_i=x_i}$, then $scope(n) = \{X_i\}$, otherwise $scope(n) = \cup_{m \in children(n)} scope(m)$.

Definition 2 (Decomposability) *An SPN is decomposable when each product node has children with disjoint scopes.*

Definition 3 (Completeness) *An SPN is complete when each sum node has children with identical scope.*

2.2 Learning SPNs

Several algorithms have been proposed to learn the parameters and the structure of an SPN [Dennis and Ventura, 2012, Gens and Domingos, 2013, Peharz et al., 2013, Lee et al., 2013, Rooshenas and Lowd, 2014, Adel et al., 2015]. Parameter learning algorithms estimate the weights of an SPN for a fixed structure. Below, we review gradient descent (GD), and expectation maximization (EM) since they are the two most popular parameter learning algorithms for SPNs and they can be adapted easily to online learning. We also briefly introduce exponentiated gradient (EG) [Kivinen and Warmuth, 1997], which has been frequently used in machine learning to train convex models in the online setting.

2.2.1 Stochastic Gradient Descent (SGD)

Gradient descent has been proposed by [Poon and Domingos, 2011] to train SPNs in batch mode. The basic idea is to treat an SPN as a deep computational graph with probabilistic output, so that gradient descent can be applied to maximize the log-likelihood of the training data. Just as in other deep models, the gradient vector of model parameters in SPNs can be computed using the chain rule in a top-down fashion. Hence the computational complexity of GD for each instance scales linearly in the size of the network. Besides its conceptual simplicity, another advantage of GD is that it can be easily adapted to online learning by making updates after each observation, which is known as the stochastic gradient descent (SGD) algorithm. SGD is an off-the-shelf learning algorithm for many deep models. However, the model parameters need to be projected back into the feasible region after each iteration of SGD. Given a data instance $\mathbf{x}$, the update formula of SGD in SPNs can be expressed as follows

$$\mathbf{w}^{k+1} \leftarrow \left[\mathbf{w}^k - t_k \left(\nabla_{\mathbf{w}} V_{root}(\mathbf{e}(\mathbf{x})) - \nabla_{\mathbf{w}} V_{root}(\mathbf{1})\right)\right]_{\epsilon} \tag{5}$$

where t_k is the step size in the kth iteration, $[w]_\epsilon = \max\{\epsilon, w\}$ is the projection operator to $\epsilon-$positive region ($w \geq \epsilon > 0$).

2.2.2 Exponentiated Gradient (oEG)

Explicit projection is required in SGD in order to keep all the intermediate solutions in the feasible region. As an alternative, exponentiated gradient (EG) was proposed by [Kivinen and Warmuth, 1997] to train linear separators in the online setting. After that, EG has been widely applied in both online and batch settings to train convex models, including log-linear structured predictors [Globerson et al., 2007], conditional random fields and max-margin Markov networks [Collins et al., 2008], etc. EG has not been applied in training SPNs or other related nonconvex deep architectures. However, EG yields a natural training paradigm that admits a multiplicative update in each iteration so that there is no need to project intermediate gradients back to the feasible region. For this reason, we also treat EG as a candidate online learning algorithm for SPNs.

2.2.3 Expectation Maximization (oEM)

SGD is widely applicable to any deep models with differentiable objective functions. However, it fails to exploit the structure that the $V_{root}(\mathbf{e}(\mathbf{x}))$ can be normalized to obtain a joint probability in SPNs. To utilize the generative nature of SPNs, EM has been proposed [Peharz, 2015] as another approach to learn the parameters of SPNs. The key observation here is that we can treat sum nodes in SPNs as hidden variables in a mixture model. More specifically, for each sum node n in a SPN, n can be viewed as a multinomial random variable, where the number of values taken by n corresponds to the number of edges emanating from n. Such intuitive explanation will not affect the inference procedure for the observable variables in SPNs because all the sum nodes are summed out in the inference phase. This perspective leads to the possibility of applying the EM algorithm to learn the model parameters of SPNs. Again, EM can be extended to online computation by making an incremental update after each observation [Liang and Klein, 2009]. We refer interested readers to [Peharz, 2015] for more details about the EM algorithm in SPNs.

3 Online Bayesian Moment Matching

We propose to use Bayesian learning to obtain a new online algorithm. As pointed out by [Broderick et al., 2013] Bayesian learning naturally lends itself to online learning and distributed computation. In the case of an SPN, the weights are the parameters to be learned. Bayesian learning starts by expressing a prior $\Pr(\mathbf{w})$ over the weights. Learning corresponds to computing the posterior distribution $\Pr(\mathbf{w}|data)$ based on the data observed according to Bayes' theorem:

$$\Pr(\mathbf{w}|data) \propto \Pr(\mathbf{w}) \Pr(data|\mathbf{w}) \tag{6}$$

Since the data consists of a set of instances $\mathbf{x}^{1:N} = \{\mathbf{x}^1, ..., \mathbf{x}^N\}$ that is assumed to be sampled identically and independently from some underlying distribution, we can rewrite Bayes' theorem in a recursive way that facilitates incremental online learning:

$$\Pr(\mathbf{w}|\mathbf{x}^{1:n+1}) \propto \Pr(\mathbf{w}|\mathbf{x}^{1:n}) \Pr(\mathbf{x}^{n+1}|\mathbf{w}) \tag{7}$$

The computation of the posterior in Bayes' theorem can also be distributed over several machines that each process a subset of the data. For example, suppose that we have K machines and a dataset of KN instances, then each machine (indexed by k) can compute a posterior $\Pr(\mathbf{w}|\mathbf{x}^{(k-1)N+1:kN})$ over N instances. Those posteriors can be combined to obtain the posterior of the entire dataset as follows:

$$\Pr(\mathbf{w}|\mathbf{x}^{1:KN}) = \Pr(\mathbf{w}) \prod_{k=1}^{K} \frac{\Pr(\mathbf{w}|\mathbf{x}^{(k-1)N+1:kN})}{\Pr(\mathbf{w})} \tag{8}$$

Hence, exact Bayesian learning can be performed naturally in an online and distributed fashion. Unfortunately, the computation of the posterior is often intractable. This is the case for parameter learning in SPNs. Thus, we propose to approximate the posterior obtained after each data instance with a tractable distribution by matching a set of moments. We first describe how to estimate the parameters of an SPN with a fixed structure by exact Bayesian learning. Then we discuss the exponential complexity of this approach and how to circumvent this intractability by Bayesian moment matching.

3.1 Exact Bayesian Learning

The parameters of an SPN consist of the weights associated with the edges emanating from each sum node. The first step is to define a prior over the weights. While the weights can be any non-negative number, any SPN can be transformed into an equivalent *normal* SPN with normalized weights (i.e., $w_{ij} \geq 0$ and $\sum_j w_{ij} = 1 \; \forall i \in \text{sumNodes}$) that correspond to local distributions [Peharz et al., 2015, Zhao et al., 2015]. Without loss of generality, we will restrict ourselves to normal SPNs since the likelihood of a data instance $\Pr(\mathbf{x})$ is obtained by a single network evaluation $V_{root}(\mathbf{e}(\mathbf{x}))$ (i.e., the normalization constant $V_{root}(\mathbf{1})$ is always 1 in Eq. 2) and this allows us to use a Dirichlet for the prior over each local distribution associated with the weights of each sum node. We start with a prior that consists of a product of Dirichlets with respect to the weights $\mathbf{w}_{i\cdot} = \{w_{ij} | j \in children(i)\}$ of each sum node i:

$$\Pr(\mathbf{w}) = \prod_{i \in \text{sumNodes}} Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) \tag{9}$$

The posterior is obtained by multiplying the prior by the likelihood $V_{root}(\mathbf{x})$ of each data instance:

$$\Pr(\mathbf{w}|\mathbf{x}^{1:n+1}) \propto \Pr(\mathbf{w}|\mathbf{x}^{1:n}) V_{root}(\mathbf{x}^{n+1}) \tag{10}$$

Since a network evaluation consists of an alternation of sums and products, we can rewrite $V_{root}(\mathbf{x})$ as a polynomial with respect to the weights. Furthermore, this polynomial can always be re-written as a sum of monomials that each consists of a product of weights:

$$V_{root}(\mathbf{x}) = \sum_c monomial_c^{\mathbf{x}}(\mathbf{w}) = \sum_c \prod_{ij \in monomial_c^{\mathbf{x}}} w_{ij} \tag{11}$$

Intuitively, if we distribute the sums over the products in the network evaluation, we obtain a large sum of small products of weights where each product of weights is a monomial. The number of monomials is exponential in the number of sum nodes. Note that products of Dirichlets are conjugate priors with respect to monomial likelihood functions. This means that multiplying a monomial by a product of Dirichlets gives a distribution that is again a product of Dirichlets. However, since our likelihood function is a polynomial that consists of a sum of monomials, the posterior becomes a mixture of products of Dirichlets:

$$\Pr(\mathbf{w}|\mathbf{x}) \tag{12}$$

$$= \prod_{i \in \text{sumNodes}} Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) \sum_c monomial_c^{\mathbf{x}}(\mathbf{w}) \tag{13}$$

$$= \sum_c k_c \prod_{i \in \text{sumNodes}} Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot} + \delta_{i\cdot}^{\mathbf{x},c}) \tag{14}$$

$$\text{where } \delta_{ij}^{\mathbf{x},c} = \begin{cases} 1 & ij \in monomial_c^{\mathbf{x}} \\ 0 & ij \notin monomial_c^{\mathbf{x}} \end{cases} \tag{15}$$

The nice thing about the above derivation is that the posterior has a closed form (mixture of products of Dirichlets), however it is computationally intractable. The number of mixture components is exponential in the number of sum nodes after the first data instance (in the worst case). If we repeatedly update the posterior after each data instance according to Eq. 7, the

number of mixture components will grow exponentially with the amount of data and doubly exponentially with the number of sum nodes.

3.2 Moment Matching

Moment matching is a popular frequentist technique to estimate the parameters of a distribution based on the empirical moments of a dataset. For instance, it has been used to estimate the parameters of mixture models, latent Dirichlet allocation and hidden Markov models while ensuring consistency [Anandkumar et al., 2012]. Moment matching can also be used in a Bayesian setting to approximate an intractable posterior distribution. More precisely, by computing a subset of the moments of an intractable distribution, another distribution from a tractable family that matches those moments can be selected as a good approximation. Expectation propagation [Minka and Lafferty, 2002] is a good example of such an approach. We describe how to use Bayesian Moment Matching to approximate mixtures of products of Dirichlets obtained after processing each data instance by a single product of Dirichlets.

A Dirichlet $Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) \propto \prod_j (w_{ij})^{\alpha_{ij}-1}$ is defined by its hyper-parameters α. However it could also be defined by a set of moments. Consider the following first and second order moments which are expectations of w_{ij} and w_{ij}^2:

1. $M_{Dir}(w_{ij}) = \int_{w_{ij}} w_{ij} Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) dw_{ij}$
2. $M_{Dir}(w_{ij}^2) = \int_{w_{ij}} w_{ij}^2 Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) dw_{ij}$

We can express the hyperparameters α_{ij} in terms of the above moments as follows:

$$\alpha_{ij} = M_{Dir}(w_{ij}) \frac{M_{Dir}(w_{ij}) - M_{Dir}(w_{ij}^2)}{M_{Dir}(w_{ij}^2) - (M_{Dir}(w_{ij}))^2} \quad \forall ij \tag{16}$$

When approximating a distribution P by a Dirichlet, we can compute the moments $M_P(w_{ij})$ and $M_P(w_{ij}^2)$ of P and then use Eq. 16 to set the hyperparameters of the Dirichlet so that it has the same first and second order moments. More generally, since we are interested in approximating a joint distribution $P(\mathbf{w})$ by a *product* of Dirichlets, we can compute the moments $M_{P(\mathbf{w}_{i\cdot})}(w_{ij})$ and $M_{P(\mathbf{w}_{i\cdot})}(w_{ij}^2)$ of each marginal $P(\mathbf{w}_{i\cdot})$ in order to set the hyperparameters $\alpha_{i\cdot}$ of each Dirichlet in the product of Dirichlets. Hence, instead of explicitly computing the intractable posterior after each data instance, we compute the first and second order moments of the posterior which are sufficient to approximate the posterior by Bayesian moment matching with a product of Dirichlets. It turns out that we can exploit the structure of SPNs to compute efficiently the first and second order moments of the posterior. Algorithm 1 describes how to compute a moment of the posterior in one bottom-up pass. Since the number of moments is linear in the size of the network and computing each moment is also linear, the overall time to approximate the posterior by a product of Dirichlets is quadratic in the size of the network.

Algorithm 1 Compute marginal moment for w_{ij}^k (k is an exponent indicating the order of the moment) in the posterior obtained after observing $\mathbf{x}$

```
if isLeaf(node) then
    return V_node(e(x))
else if isProduct(node) then
    return ∏_child computeMoment(child)
else if isSum(node) and node == i then
    return Σ_child M_Dir(w_ij^k w_{i,child})×
                computeMoment(child)
else
    return Σ_child w_{node,child} computeMoment(child)
end if
```

In practice, many structure learning algorithms produce SPNs that are trees (i.e., each node has a single parent) [Dennis and Ventura, 2012, Gens and Domingos, 2013, Peharz et al., 2013, Rooshenas and Lowd, 2014, Peharz et al., 2014a, Adel et al., 2015]. 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. The key is to compute two coefficients $coef_i^0, coef_i^1$ at each node i in a 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:

$$M_{posterior}(w_{ij}^k) = \int_{w_{i\cdot}} w_{ij}^k Dir(\mathbf{w}_{i\cdot}|\alpha_{i\cdot}) (coef_i^0 + coef_i^1 \sum_{j'} w_{ij'} V_{j'}(\mathbf{e}(\mathbf{x}))) dw_{i\cdot} \tag{17}$$

Here, $coef_i^0$ and $coef_i^1$ are additive and multiplicative coefficients respectively that capture quantities in the SPN above node i. We compute those coefficients in a top down fashion. At the root, $coef_{root}^0$ is initialized to 0 since there is nothing above the root to be added. Similarly, $coef_{root}^1$ is initialized to 1 since there is nothing above the root to be multiplied. Then, as we go down the SPN, the coefficients of the parent and network values of the siblings of node i are combined into $coef_i^0$ and $coef_i^1$.

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

if $isRoot(node)$ **then**
  $coef^0_{node} \leftarrow 0$
  $coef^1_{node} \leftarrow 1$
else if $isProduct(parent(node))$ **then**
  $coef^0_{node} \leftarrow coef^0_{parent}$
  $coef^1_{node} \leftarrow coef^1_{parent} \prod_{sibling} V_{sibling}(e(\mathbf{x}))$
else if $isSum(parent(node))$ **then**
  $coef^0_{node} \leftarrow coef^0_{parent} + \sum_{sibling} \frac{\alpha_{parent,sibling}}{\sum_j \alpha_{parent,j}} V_{sibling}(e(\mathbf{x}))$
  $coef^1_{node} \leftarrow coef^1_{parent} \frac{\alpha_{parent,node}}{\sum_j \alpha_{parent,j}}$
end if
if $isNotLeaf(node)$ **then**
  $computeCoefficients(child)\ \forall child$
end if

4 Experiments

We evaluated oBMM on 2 sets of datasets; 20 small datasets and 4 large datasets. The 20 datasets span diverse sets of domains, and the number of variables range from 16 to 1556 binary variables. These datasets were used before for comparisons in [Gens and Domingos, 2013, Rooshenas and Lowd, 2014]. The 4 large datasets are bag of words datasets and they are publicly available online from the UCI machine learning repository. Each dataset contains a collection of documents and the corresponding word counts. The number of variables (dictionary size) ranges from 6906 to 102660 variables. The variables are binarized by ignoring the word count and only used 0 or 1 for the absence or presence of a word in a document.

In order to evaluate the performance of oBMM, we evaluated it and compared it to three online algorithms: stochastic gradient descent (SGD), online exponentiated gradient (oEG) and online expectation maximization (oEM). We measure both the quality of the three algorithms in terms of average log-likelihood scores on the held-out test data sets and their scalability in terms of running time. To test the statistical significance of the results, we apply the Wilcoxon signed rank test [Wilcoxon, 1950] to compute the p-value and report statistical significance with p-value less than 0.05. The log-likelihood scores for the structural learning algorithm from LearnSPN [Gens and Domingos, 2013] are reported as a reference for the comparison of the three online parameter learning algorithms. The online algorithms are not expected to beat a structural learning algorithm, but we will show that online algorithms can outperform LearnSPN for the large datasets in terms of running time and accuracy.

Since we don't do structural learning, a fixed structure is used to perform the experiments. We used a simple structure generator which can take 2 hyperparameters (number of variables, and the depth of the SPN) and generate the SPN accordingly. The generator keeps track of the scope at each node and it starts from a sum node as the root node that has all variables in the scope. Recursively, for each sum node, a number of children product nodes are generated with the same scope as the sum node, and for each product node, a number of children sum nodes are generated while randomly factoring the scope among the children. When the level at a product node is 2 or less, a number of children sum nodes are generated such that each child has only a single variable in its scope. Finally, a sum node that contains a single variable in its scope, two leaf nodes are generated corresponding to the indicators of the variables. For the small datasets, SPNs of depth 6 are used, while for the large datasets SPNs of depth 4 are used.

Online Bayesian Moment matching (oBMM) doesn't have any hyper-parameters to tune, except for the prior which we set randomly by selecting hyperparameters α_{ij} uniformly at random in $[0, 1]$. For the online Distributed Bayesian Moment Matching (oDMM), the training set is partitioned into 10 smaller sets, and each set is sent to a different machine to be processed. Once the machines finish, the output sum-product networks are collected and combined into a single sum-product network. For SGD and oEG, we use backtracking line search to shrink the step size if the log-likelihood scores on the training set decreases. The initial step size is set to 1.0 for all data sets. oEM does not require a learning rate nor weight shrinking. To avoid possible numeric underflow we use a smoothing constant equal to 0.01 in all the experiments.

Table 1 shows the average log-likelihoods on the test sets for various algorithms. oBMM outperforms the rest of the online algorithms in 19 out of the 20 datasets. The results show that oBMM has significantly better log-likelihood on most of the datasets. Surprisingly, oBMM matches the performance of LearnSPN in 11 datasets despite the fact that the structure used for oBMM is generated randomly. The advantage of online algorithms over batch algorithms including LearnSPN appears when the dataset size increases to the limit since it becomes hard to fit it in memory. Tables 2 and 3 show the likelihood scores and the running times of different methods on large datasets. Dashes are used to indicate that the algorithm didn't finish in a week or the training data could not fit in memory. We were able to run LearnSPN only on the KOS dataset since the rest of the

Table 1: Log-likelihood scores on 20 data sets. The best results among oBMM, SGD, oEM, and oEM are highlighted in bold font. ↑ (↓) indicates that the method has significantly better (worse) log-likelihoods than oBMM under Wilcoxon signed rank test with p value < 0.05.

Dataset	Var#	LearnSPN	oBMM	SGD	oEM	oEG
NLTCS	16	-6.11	**-6.07**	↓-8.76	↓-6.31	↓-6.85
MSNBC	17	-6.11	**-6.03**	↓-6.81	↓-6.64	↓-6.74
KDD	64	-2.18	**-2.14**	↓-44.53	↓-2.20	↓-2.34
PLANTS	69	-12.98	**-15.14**	↓-21.50	↓-17.68	↓-33.47
AUDIO	100	-40.50	**-40.7**	↓-49.35	↓-42.55	↓-46.31
JESTER	100	-53.48	**-53.86**	↓-63.89	↓-54.26	↓-59.48
NETFLIX	100	-57.33	**-57.99**	↓-64.27	↓-59.35	↓-64.48
ACCIDENTS	111	-30.04	**-42.66**	↓-53.69	-43.54	↓-45.59
RETAIL	135	-11.04	**-11.42**	↓-97.11	↓-11.42	↓-14.94
PUMSB-STAR	163	-24.78	**-45.27**	↓-128.48	↓-46.54	↓-51.84
DNA	180	-82.52	**-99.61**	↓-100.70	↓-100.10	↓-105.25
KOSAREK	190	-10.99	**-11.22**	↓-34.64	↓-11.87	↓-17.71
MSWEB	294	-10.25	**-11.33**	↓-59.63	↓-11.36	↓-20.69
BOOK	500	-35.89	**-35.55**	↓-249.28	↓-36.13	↓-42.95
MOVIE	500	-52.49	**-59.50**	↓-227.05	↓-64.76	↓-84.82
WEBKB	839	-158.20	**-165.57**	↓-338.01	↓-169.64	↓-179.34
REUTERS	889	-85.07	**-108.01**	↓-407.96	-108.10	↓-108.42
NEWSGROUP	910	-155.93	**-158.01**	↓-312.12	↓-160.41	↓-167.89
BBC	1058	-250.69	-275.43	↓-462.96	**-274.82**	↓-276.97
AD	1556	-19.73	**-63.81**	↓-638.43	↓-63.83	↓-64.11

Table 2: Log-lielihood scores on 4 large datasets. The best results are highlighted in bold font, and dashes "-" are used to indicate that an algorithm didn't finish or couldn't load the dataset into memory.

Dataset	Var#	LearnSPN	oBMM	oDMM	SGD	oEM	oEG
KOS	6906	-444.55	**-422.19**	-437.30	-3492.9	-538.21	-657.13
NIPS	12419	-	**-1691.87**	-1709.04	-7411.20	-1756.06	-3134.59
ENRON	28102	-	**-518.842**	-522.45	-13961.40	-554.97	-14193.90
NYTIMES	102660	-	-1503.65	-1559.39	-43153.60	**-1189.39**	-6318.71

Table 3: Running time in minutes on 4 large datasets. The best running time is highlighted in bold font, and dashes "-" are used to indicate that an algorithm didn't finish or couldn't load the dataset into memory.

Dataset	Var#	LearnSPN	oBMM	oDMM	SGD	oEM	oEG
KOS	6906	1439.11	89.40	**8.66**	162.98	59.49	155.34
NIPS	12419	-	139.50	**9.43**	180.25	64.62	178.35
ENRON	28102	-	2018.05	**580.63**	876.18	694.17	883.12
NYTIMES	102660	-	12091.7	**1643.60**	5626.33	5540.40	6895.00

datasets didn't finish in a week or they were too big to fit in memory. Both oBMM and oDMM outperform LearnSPN, which runs 16 times slower than oBMM and 166 times slower than oDMM. oBMM and oDMM also outperform SGD by a large margin on the large data sets. In terms of the running time, oDMM is significantly faster than the rest of the algorithms since we are distributing the training data over multiple machines.

5 Conclusion

SPNs gained popularity for being able to provide exact inference in linear time, however learning the parameters in a tractable way is still challenging. In this paper, we explored online algorithms for learning the parameters of an SPN tractably. We also proposed online Bayesian Moment Matching as an online framework for parameters learning, and we showed how to distribute the algorithm over many machines. We showed how oBMM outperforms the rest of the algorithms, we also showed that distributing the algorithm over multiple machines is very effective when running time is the bottleneck.

References

[Adel et al., 2015] Adel, T., Balduzzi, D., and Ghodsi, A. (2015). Learning the structure of sum-product networks via an svd-based algorithm.

[Amer and Todorovic, 2015] Amer, M. and Todorovic, S. (2015). Sum product networks for activity recognition.

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

[Anandkumar et al., 2012] Anandkumar, A., Hsu, D., and Kakade, S. M. (2012). A method of moments for mixture models and hidden markov models. *Journal of Machine Learning Research - Proceedings Track*, 23:33.1–33.34.

[Broderick et al., 2013] Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013). Streaming variational bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735.

[Cheng et al., 2014] Cheng, W.-C., Kok, S., Pham, H. V., Chieu, H. L., and Chai, K. M. A. (2014). Language modeling with sum-product networks. *INTERSPEECH*.

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

[Dennis and Ventura, 2012] Dennis, A. and Ventura, D. (2012). Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*.

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

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

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

[Lee et al., 2013] Lee, S.-W., Heo, M.-O., and Zhang, B.-T. (2013). Online incremental structure learning of sum–product networks. In *Neural Information Processing*, pages 220–227. Springer.

[Liang and Klein, 2009] Liang, P. and Klein, D. (2009). Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619. Association for Computational Linguistics.

[Minka and Lafferty, 2002] Minka, T. and Lafferty, J. (2002). Expectation-propagation for the generative aspect model. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 352–359. Morgan Kaufmann Publishers Inc.

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

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

[Peharz et al., 2014a] Peharz, R., Gens, R., and Domingos, P. (2014a). Learning selective sum-product networks. *ICML Workshop on Learning Tractable Probabilistic Models*.

[Peharz et al., 2014b] Peharz, R., Kapeller, G., Mowlaee, P., and Pernkopf, F. (2014b). 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.

[Peharz et al., 2015] Peharz, R., Tschiatschek, S., Pernkopf, F., Domingos, P., and BioTechMed-Graz, B. (2015). On theoretical properties of sum-product networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 744–752.

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

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

[Wilcoxon, 1950] Wilcoxon, F. (1950). Some rapid approximate statistical procedures. *Annals of the New York Academy of Sciences*, pages 808–814.

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