Discriminative Training of Sum-Product Networks by Extended Baum-Welch

Abdullah Rashwan ARASHWAN@UWATERLOO.CA
Pascal Poupart PPOUPART@UWATERLOO.CA
University of Waterloo, Waterloo AI Institute, Ontario, Canada
Vector Institute, Toronto, Canada

Zhitang Chen CHENZHITANG2@HUAWEI.COM
Huawei Technologies, Hong Kong, China

Abstract

We present a discriminative learning algorithm for Sum-Product Networks (SPNs) (Poon and Domingos, 2011) based on the Extended Baum-Welch (EBW) algorithm (Baum et al., 1970). We formulate the conditional data likelihood in the SPN framework as a rational function, and we use EBW to monotonically maximize it. We derive the algorithm for SPNs with both discrete and continuous variables. The experiments show that this algorithm performs better than both generative Expectation-Maximization, and discriminative gradient descent on a wide variety of applications. We also demonstrate the robustness of the algorithm in the case of missing features by comparing its performance to Support Vector Machines and Neural Networks.

1. Introduction

Sum-Product networks (SPNs) were first proposed by Poon and Domingos (2011) as a new type of deep architecture that can be viewed as probabilistic graphical models that are equivalent to arithmetic circuits (ACs) (Darwiche, 2003). 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 marginal inference can be done without any approximation in linear time with respect to the size of the network. Marginal MAP inference is still intractable for SPNs.

Various generative learning algorithms have been designed to estimate the parameters of SPNs, including Gradient Descent and hard EM (Poon and Domingos, 2011), soft EM (Peharz, 2015), Bayesian moment matching (Rashwan et al., 2016), collapsed variational Bayes (Zhao et al., 2016), sequential monomial approximations and the concave-convex procedure (Zhao and Poupart, 2016). Discriminative training of SPNs by gradient descent was introduced by Gens and Domingos (2012). Although gradient descent is tractable, convergence can be quite slow since it uses first order approximations. Adel et al. (2015) introduced a novel discriminative algorithm for SPNs that learns the structure of the SPN while extracting features that are maximally correlated with the labels. The algorithm has been shown to perform well compared to generative structure learning algorithms.

However, it is a batch algorithm that recursively performs singular value decompositions that are very expensive.

In this paper, we present a novel algorithm to train SPNs discriminatively based on the Extended Baum-Welch technique. While Expectation Maximization and Baum-Welch are equivalent in generative training, they cannot be used directly in discriminative training and their extensions for maximizing conditional likelihoods are not the same. Extended Baum-Welch (for discriminative training) (Gopalakrishnan et al., 1991) is simpler both conceptually and computationally than conditional EM (for discriminative training) (Jebara and Pentland, 1998, 2000; Salojärvi et al., 2005) and therefore has become the most popular approach to train HMMs discriminatively. Extended Baum-Welch provides a general approach to optimize rational functions such as conditional likelihoods. It also offers faster convergence than gradient descent while guaranteeing monotonic improvement at each iteration (Normandin, 1991). In order to apply Extended Baum-Welch to discriminative SPNs, we will formulate the conditional distribution as a rational function. We will develop the algorithm for SPNs with both multinomial and univariate Gaussian distributions at the leaves.

The paper is structured as follows, Section 2 reviews SPNs, Baum-Welch and Extended Baum-Welch algorithms. Section 3 introduces discriminative SPNs and explains how to compute the conditional likelihood for a classification task. Then, update formulas for discriminative gradient descent and Extended Baum-Welch are derived for SPNs with discrete and continuous variables. Section 4 presents three sets of experiments that we carried out to evaluate the performance of our algorithm. Section 5 concludes the paper.

2. Background

2.1 Sum-Product Networks

Consider a set of random variables $\mathbf{X}$. 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 tractable distributions over a subset of variables. In this paper, we will consider leaves that consist of Bernoulli distributions over binary variables and univariate Gaussian distributions over continuous variables. Other works also consider SPNs with leaves that contain multivariate discrete distributions (Rooshenas and Lowd, 2014), Poisson distributions (Molina et al., 2017), multivariate Gaussian distributions (Jaini and Poupart, 2016; Hsu et al., 2017) distributions from the exponential family (Desana and Schnörr, 2016) and piecewise polynomial distributions (Molina et al., 2018). 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 encodes a function $F(\mathbf{x})$ that takes as input a variable assignment $\mathbf{X} = \mathbf{x}$ and produces an output at its root. This function is defined recursively at each node n as follows:

$$F_n(\mathbf{x}) = \begin{cases} P(\mathbf{X}_n = \mathbf{x}_n) & n \text{ is a leaf} \\ \prod_{m \in children(n)} F_m(\mathbf{x}) & n \text{ is a product} \\ \sum_{m \in children(n)} w_{nm} F_m(\mathbf{x}) & n \text{ is a sum} \end{cases} \tag{1}$$

Here, $\mathbf{X}_n = \mathbf{x}_n$ denotes the variable assignment restricted to the variables contained in leaf n. If none of the variables in leaf n are instantiated by $\mathbf{X} = \mathbf{x}$ then $P(\mathbf{X}_n = \mathbf{x}_n) = P(\emptyset) = 1$. Note also that if leaf n contains continuous variables, then $P(\mathbf{X}_n = \mathbf{x}_n)$ should be interpreted as a probability density function $pdf(\mathbf{X}_n = \mathbf{x}_n)$.

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 variables according to the joint assignment.

$$P(\mathbf{X} = \mathbf{x}) = \frac{F_{root}(\mathbf{x})}{F_{root}(\emptyset)} \tag{2}$$

The normalization constant needed to obtain a probability is $F_{root}(\emptyset)$ where $\emptyset$ is the empty variable assignment, which means that all the variables are marginalized. Eq. 2 can also be used to compute the marginal probability of a partial assignment $\mathbf{Y} = \mathbf{y}$ where $\mathbf{Y} \subseteq \mathbf{X}$. Conditional probabilities can also be computed by evaluating two partial assignments:

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

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 (i.e., number of nodes, edges and parameters).[1]

An SPN is said to be *valid* when it represents a distribution and Eqs 2 and 3 can be used to answer inference queries correctly (Poon and Domingos, 2011). *Decomposability* and *completeness* are sufficient conditions that ensure validity (Darwiche, 2003; Poon and Domingos, 2011). 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 leaves 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 with base distribution $P(X_i = x_i)$, 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 Extended Baum-Welch Algorithm

The Baum-Welch algorithm was introduced in 1970 to estimate the parameters for HMMs (Baum et al., 1970). The algorithm was based on the Baum-Eagon inequality (Baum et al., 1967), which monotonically maximizes polynomials that satisfy certain conditions. The algorithm was then extended to maximizing rational functions (i.e., ratio of two polynomial functions) (Gopalakrishnan et al., 1991), which made it very useful in discriminative training settings. Extended Baum-Welch (EBW) was used to discriminatively train HMMs, GMMs, as well as discrete distributions

1. It is common to measure the complexity of probabilistic graphical models with respect to their tree-width, however tree-width is not a practical statistic since finding the tree-width of a graph is NP-hard. Instead, we describe the complexity of inference with respect to the size of the graph (number of nodes, edges and parameters), which is immediately available.

(Pernkopf and Wohlmayr, 2010; Klautau et al., 2003; Normandin and Morgera, 1991) In this Section, we will review the original Baum-Welch algorithm, then we will explain how it was extended to work for rational polynomials. Finally, we will show how to use EBW to train SPNs discriminatively in Section 3.

Theorem 4 *(Baum et al., 1967) Let $S(\theta)$ be a homogeneous degree d polynomial with non-negative coefficients. Let $\bar{\theta} = \{\bar{\theta}_{ij}\}$ be any point in the domain $\mathcal{D} : \sum_j \theta_{ij} = 1, \forall i$. Let $\hat{\theta} = T(\bar{\theta}) = T(\{\bar{\theta}_{ij}\})$ be a transformation function such that*

$$\hat{\theta}_{ij} = \frac{\bar{\theta}_{ij} \frac{\partial S}{\partial \theta_{ij}}(\bar{\theta})}{\sum_j \bar{\theta}_{ij} \frac{\partial S}{\partial \theta_{ij}}(\bar{\theta})}, \tag{4}$$

where $\sum_j \bar{\theta}_{ij} \frac{\partial S}{\partial \theta_{ij}}(\bar{\theta}) \neq 0$, $\frac{\partial S}{\partial \theta_{ij}}(\bar{\theta})$ is the value of $\frac{\partial S}{\partial \theta_{ij}}$ at $\bar{\theta}$. Then, $S(\hat{\theta}) > S(\bar{\theta})$ unless $T(\bar{\theta}) = \bar{\theta}$.

Theorem 4 is applied iteratively to optimize a polynomial $S(\theta)$. The transformation $T(\bar{\theta})$ is called a growth transform since it increases $S(\theta)$ monotonically. In discrete HMMs and other discrete mixture models, the likelihood function is a polynomial in the parameters θ and therefore Eq. 4 can be used to iteratively improve the parameters in a way that the likelihood monotonically improves. Interestingly, we obtain the same update formula as for Expectation-Maximization.

The growth transform can be extended to rational functions, $R_d(\theta) = \frac{Num(\theta)}{Den(\theta)}$, where both the numerator, $Num(\theta)$, and the denominator, $Den(\theta)$, are polynomials.

Theorem 5 *(Gopalakrishnan et al., 1991) Let $R_d(\theta) = \frac{Num(\theta)}{Den(\theta)}$ be a rational function. Let $\bar{\theta} = \{\bar{\theta}_{ij}\}$ be any point in the domain $\mathcal{D} : \sum_j \theta_{ij} = 1, \forall i$. Let's construct polynomials $Q(\theta)$ and $S(\theta)$ as follows*

$$\begin{aligned} Q(\theta) &= Num(\theta) - R_d(\bar{\theta})Den(\theta) \\ S(\theta) &= Q(\theta) + C(\theta) \end{aligned} \tag{5}$$

where $C(\theta) = c\left[\sum_{i,j} \theta_{ij} + 1\right]^d$, c is chosen such that it cancels all negative coefficients in $Q(\theta)$, and d is the degree of $Q(\theta)$. Based on the above construction, Theorem 4 can be applied to $S(\theta)$ such that $R_d(\hat{\theta}) > R_d(\bar{\theta})$, unless $T(\bar{\theta}) = \bar{\theta}$.

In discriminative learning, the conditional likelihood can typically be expressed as a rational function $R(\theta)$ (i.e., the data likelihood divided by the marginal of the inputs). In the next section, we will show how to construct a polynomial $S(\theta)$ from the rational function corresponding to the conditional likelihood of SPNs and then apply the growth function to improve the conditional likelihood monotonically.

3. Discriminative Sum-Product Networks

Let the training set be $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and the corresponding labels $\mathbf{Y} = \{y_1, ..., y_N\}$, where $\mathbf{x}_i \in \{0,1\}^M$, $y_i \in \{0, ..., Y\}$, N is the number of training examples, M is the feature size, and Y is the number of class labels. In this section, we first provide some preliminary derivations for discriminative SPNs, then extend the discriminative gradient descent technique proposed by

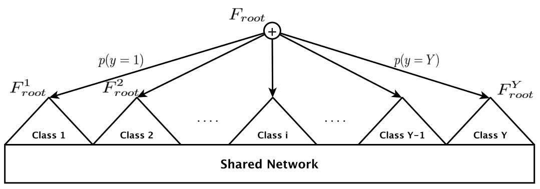


Figure 1: Discriminative SPN architecture.

Gens and Domingos (2012) to continuous SPNs with Gaussian leaves, and finally we describe our new discriminative learning technique based on Extended Baulm-Welch.

In discriminative training, we maximize the conditional probability distribution $P(y|\mathbf{x})$. To do that, we use the discriminative SPN architecture shown in Figure 1 where there is a sub-SPN, SPN_y, for each class y , and sub-SPNs can share part of the network. Each sub-SPN models the likelihood of an observation given the class label, $F^y_{root}(\mathbf{x}) = p(\mathbf{x}|y)$. Evaluating the whole network gives us the probability of an observation, $p(\mathbf{x}) = \sum_y p(y)F^y_{root}(\mathbf{x})$. According to Bayes rule, the conditional probability can be computed as follows:

$$P(y|\mathbf{x}) = \frac{p(y)p(\mathbf{x}|y)}{p(\mathbf{x})} = \frac{p(y)F^y_{root}(\mathbf{x})}{F_{root}(\mathbf{x})} \tag{6}$$

The label associated with the sub-SPN that maximizes the conditional probability distribution is selected.

$$\arg max_{y=1,\ldots,Y} p(y)F^y_{root}(\mathbf{x}) \tag{7}$$

In the training phase, the posterior $P(\mathbf{Y}|\mathbf{X})$ is maximized. The posterior is computed in terms of the prior and the likelihoods as follows

$$P(\mathbf{Y}|\mathbf{X}) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) = \prod_{n=1}^{N} \frac{P(y_n)F^{y_n}_{root}(\mathbf{x}_n)}{F_{root}(\mathbf{x}_n)} \tag{8}$$

Similarly $log(P(\mathbf{Y}|\mathbf{X}))$ is computed as follows.

$$log(P(\mathbf{Y}|\mathbf{X})) = \sum_{n=1}^{N} log(p(y_n)F^{y_n}_{root}(\mathbf{x}_n)) - log\big(F_{root}(\mathbf{x}_n)\big) \tag{9}$$

The above equations show that discriminative training aims at maximizing the likelihood of the correct class, similar to generative training, while minimizing the likelihoods of the remaining classes. Estimating $P(y)$ can be done easily and robustly by normalizing the class frequencies in the training data. Estimating the parameters of $P(\mathbf{x}|y)$, which are the weights of $F^y_{root}(\mathbf{x})$, is harder and usually doesn't have a closed form solution. Iterative methods are used in this case.

We will use $F^y_j(\mathbf{x})$ to refer to the value of the sub-SPN associated with class y at node j. Throughout the derivations, we assume that the SPN is always normalized to ensure that $P(\mathbf{x}|y) = F^y_{root}(\mathbf{x})$. This can be done by normalizing the weights after each iteration.

Finally, we will need to compute the partial derivative of the log likelihood with respect to an arbitrary parameter θ^y in the SPN, where superscript y indicates that θ^y is a parameter in the sub-SPN F^y_{root}. The derivative of the log likelihood can be obtained as follows:

$$\begin{aligned}\frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial \theta^y} &= \sum_{n=1}^{N} \frac{p(y_n)\frac{\partial F^{y_n}_{root}}{\partial \theta^y}(\mathbf{x}_n)}{p(y_n)F^{y_n}_{root}(\mathbf{x}_n)} - \frac{\frac{\partial F_{root}}{\partial \theta^y}(\mathbf{x}_n)}{F_{root}(\mathbf{x}_n)} \\ &= \sum_{n=1}^{N} \frac{1}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial \theta^y}(\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial \theta^y}(\mathbf{x}_n) \end{aligned} \tag{10}$$

Since we know that $\frac{\partial F^{y_n}_{root}}{\partial \theta^y} = 0$ when $y \neq y_n$, we can rewrite $\frac{\partial F^{y_n}_{root}}{\partial \theta^y}$ as follows:

$$\frac{\partial F^{y_n}_{root}}{\partial \theta^y} = \mathbb{1}_{[y \in SPN_{y_n}]} \frac{\partial F^{y}_{root}}{\partial \theta^y} \tag{11}$$

Also, since $F_{root}(\mathbf{x}_n) = \sum_y p(y) F^y_{root}(\mathbf{x}_n)$, we can rewrite $\frac{\partial F_{root}}{\partial \theta^y}(\mathbf{x}_n)$ as follows:

$$\frac{\partial F_{root}}{\partial \theta^y}(\mathbf{x}_n) = p(y) \frac{\partial F^{y}_{root}}{\partial \theta^y}(\mathbf{x}_n) \tag{12}$$

Finally, based on Eq. 11 and 12 we can rewrite Eq. 10 as follows:

$$\frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial \theta^y} = \sum_{n=1}^{N} \frac{\partial F^{y}_{root}}{\partial \theta^y}(\mathbf{x}_n) \left[\frac{\mathbb{1}_{[y=y_n]}}{F^{y_n}_{root}(\mathbf{x}_n)} - \frac{p(y)}{F_{root}(\mathbf{x}_n)} \right] = \sum_{n=1}^{N} \frac{\partial F^{y}_{root}}{\partial \theta^y}(\mathbf{x}_n) \gamma_n \tag{13}$$

where $\gamma_n \geq 0$. Throughout the rest of this section, we will use the above equation whenever we need the gradient of the log of the conditional likelihood.

3.1 Discriminative Learning for SPNs Using Gradient Descent

Gens and Domingos (2012) showed how to train edge weights for SPNs discriminatively by taking the gradient of the conditional log likelihood $log(P(\mathbf{Y}|\mathbf{X}))$. We briefly state how to compute discriminative gradients in continuous SPNs with Gaussian leaves.

Using Eq. 13 and knowing that $\frac{\partial F^{y}_{root}}{\partial w^y_{ij}}(\mathbf{x}) = F^y_j(\mathbf{x}_n) \frac{\partial F^{y}_{root}}{\partial F^y_i}(\mathbf{x}_n)$, the following formula can be used to update each edge weight w^y_{ij} by taking a small step η in the direction of the gradient.

$$w^y_{ij} \leftarrow w^y_{ij} + \eta \frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial w^y_{ij}} = w^y_{ij} + \eta \sum_n F^y_j(\mathbf{x}_n) \frac{\partial F^{y}_{root}}{\partial F^y_i}(\mathbf{x}_n) \gamma^y_n \tag{14}$$

Similarly, for leaf univariate Gaussian distributions, μ^y_{ij} and $(\sigma^2)^y_{ij}$ can be updated by taking a small step η in the direction of the gradient.

$$\mu^y_{ij} \leftarrow \mu^y_{ij} + \eta \sum_n \mathcal{N}^y_{ij}(x_{nj}) \frac{x_{nj} - \mu^y_{ij}}{(\sigma^2)^y_{ij}} \frac{\partial F^{y}_{root}}{\partial F^y_i}(\mathbf{x}_n) \gamma^y_n \tag{15}$$

$$(\sigma^2)^y_{ij} \leftarrow (\sigma^2)^y_{ij} + \eta \sum_n \frac{\mathcal{N}^y_{ij}(x_{nj})}{2(\sigma^2)^y_{ij}} \left[\frac{(x_{nj} - \mu^y_{ij})^2}{(\sigma^2)^y_{ij}} - 1 \right] \frac{\partial F^{y}_{root}}{\partial F^y_i}(\mathbf{x}_n) \gamma^y_n \tag{16}$$

3.2 Discriminative Learning for SPNs using Extended Baum-Welch

We first derive the Baum-Welch algorithm to maximize the log likelihood for SPNs in a generative way. Theorem 4 can be applied to SPNs assuming that the network is normalized. In that case, the polynomial $S(\theta)$ is the likelihood of the data, which corresponds to a product of network polynomials, i.e., $P(\mathbf{X}|\mathbf{Y}) = \prod_n F_{root}(\mathbf{x}_n)$. The parameters $\theta = \{w_{ij}\}$ of the polynomial satisfy the condition $\sum_j w_{ij} = 1$ since the sum of the weights for each sum node is one. To apply Theorem 4, we need to deal with the sum of the log-likelihoods, $log(P(\mathbf{X}|\mathbf{Y}))$, instead of the likelihood of the data, $P(\mathbf{X}|\mathbf{Y})$, since $log(P(\mathbf{X}|\mathbf{Y}))$ is easier to differentiate. We have

$$\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial w_{ij}} = \frac{1}{P(\mathbf{X}|\mathbf{Y})}\frac{\partial P(\mathbf{X}|\mathbf{Y})}{\partial w_{ij}} \tag{17}$$

Hence, we can rewrite Eq. 4 as follows.

$$\hat{w}_{ij} = \frac{w_{ij}P(\mathbf{X}|\mathbf{Y})\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial w_{ij}}}{\sum_j w_{ij}P(\mathbf{X}|\mathbf{Y})\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial w_{ij}}} = \frac{w_{ij}\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial w_{ij}}}{\sum_j w_{ij}\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial w_{ij}}} \tag{18}$$

The above formula is the same update formula obtained by Expectation-Maximization and the Convex-Concave Procedure (CCCP) (Zhao and Poupart, 2016).

In discriminative training for SPNs, applying Expectation-Maximization or CCCP doesn't lead to a closed form update formula (Gens and Domingos, 2012). Jebara and Pentland (1998, 2000) derived a conditional version of EM that turned out to be complicated and computationally demanding. Salojärvi et al. (2005) derived a simpler and faster update formula, but it requires second order derivatives, which is not tractable in large models with many parameters such as SPNs. In contrast, EBW can be applied to maximize the conditional likelihood with a closed form formula. Before applying Theorem 5 to SPNs, we will do the same as we did above to work with $\frac{\partial log R_d(\theta)}{\partial \theta_{ij}}$. We start by taking the derivative of $S(\theta)$ in Eq. 5 with respect to the parameters θ_{ij}

$$\frac{\partial S(\theta)}{\partial \theta_{ij}} = \frac{\partial Num(\theta)}{\partial \theta_{ij}} - R_d(\bar{\theta})\frac{\partial Den(\theta)}{\partial \theta_{ij}} + \frac{\partial C(\theta)}{\partial \theta_{ij}} \tag{19}$$

where $\frac{\partial C(\theta)}{\partial \theta_{ij}} = cd\Bigg[\sum_{i,j}\theta_{ij} + 1\Bigg]^{d-1}$ is also a constant. Since

$$\frac{\partial log(R_d(\theta))}{\partial \theta_{ij}} = \frac{1}{Num(\theta)}\frac{\partial Num(\theta)}{\partial \theta_{ij}} - \frac{1}{Den(\theta)}\frac{\partial Den(\theta)}{\partial \theta_{ij}} = \frac{1}{Num(\theta)}\left[\frac{\partial Num(\theta)}{\partial \theta_{ij}} - R_d(\theta)\frac{\partial Den(\theta)}{\partial \theta_{ij}}\right] \tag{20}$$

then we obtain the following equation.

$$\frac{\partial S(\bar{\theta})}{\partial \theta_{ij}} = Num(\bar{\theta})\left[\frac{\partial log R_d}{\partial \theta_{ij}}(\bar{\theta}) + \frac{1}{Num(\bar{\theta})}\frac{\partial C(\bar{\theta})}{\partial \theta_{ij}}\right] = Num(\bar{\theta})\left[\frac{\partial log R_d}{\partial \theta_{ij}}(\bar{\theta}) + D\right] \tag{21}$$

Substituting Eq. 21 in Equation 4, we get the following update formula

$$\hat{\theta}_{ij} = \frac{\bar{\theta}_{ij}\left[\frac{\partial log R_d}{\partial \theta_{ij}}(\bar{\theta}) + D\right]}{\sum_j\left[\bar{\theta}_{ij}\frac{\partial log R_d}{\partial \theta_{ij}}(\bar{\theta})\right] + D} \tag{22}$$

where $D = \frac{1}{Num(\bar{\theta})} \frac{\partial C(\bar{\theta})}{\partial \theta_{ij}}$. D controls how different $\hat{\theta}_{ij}$ is from $\bar{\theta}_{ij}$. Small values for D allow $\hat{\theta}_{ij}$ to change freely, while large values restrict the magnitude of changes. In practice, D is chosen such that $D < \frac{1}{Num(\bar{\theta})} \frac{\partial C(\bar{\theta})}{\partial \theta_{ij}}$ for faster convergence.

To apply EBW to SPNs, we have to define the rational function R_d, which in this case is the posterior $P(\mathbf{Y}|\mathbf{X})$. The parameters θ of the posterior are the weights w^y_{ij}, the mean μ^y_{ij}, and the variance $(\sigma^2)^y_{ij}$.

For sum nodes, the weights will be updated as follows.

$$\hat{w}^y_{ij} = \frac{w^y_{ij}\left[\sum_n F^y_j(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right] + Dw^y_{ij}}{\left[\sum_{j \in Children(i)} \sum_n F^y_j(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right] + D} \tag{23}$$

Normandin and Morgera (1991) proposed a discrete approximation for univariate Gaussian distributions that allows us to update μ^y_{ij} and $(\sigma^2)^y_{ij}$ in the Gaussian leaves of SPNs as follows.

$$\hat{\mu}^y_{ij} = \frac{\left[\sum_n \left(\mathcal{N}^y_{ij}(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right)x_{nj}\right] + D\mu^y_{ij}}{\left[\sum_n \mathcal{N}^y_{ij}(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right] + D} \tag{24}$$

$$\hat{(\sigma^2)}^y_{ij} = \frac{\left[\sum_n \left(\mathcal{N}^y_{ij}(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right)x^2_{nj}\right] + D\left[(\sigma^2)^y_{ij} + (\mu^y_{ij})^2\right]}{\left[\sum_n \mathcal{N}^y_{ij}(\mathbf{x}_n)\frac{\partial F^y_{root}}{\partial F^y_i}(\mathbf{x}_n)\gamma^y_n\right] + D} - (\hat{\mu}^y_{ij})^2 \tag{25}$$

Constant D plays an important role in the discriminative training part. D tries to preserve the previous parameters. The larger D is, the stronger will be the influence of the previous parameters on the current ones. Choosing constant D is not trivial since small values can prevent convergence, while large values induce slow convergence. A rule of thumb is to start from a small value and to increase it after each epoch.

4. Experiments

To evaluate Discriminative SPNs using EBW, we carried out three sets of experiments. In the first experiment, we compared EBW to generative EM (discriminative EM requires second order derivatives (Salojärvi et al., 2005), which is not tractable for SPNs of 1 thousand to 1 million parameters) and discriminative gradient descent. The second experiment aims to illustrate the advantage of SPNs over Support Vector Machines (SVMs) and Neural Networks in the case of missing features. In a third experiment, we trained an SPN on MNIST images using generative EM and discriminative EBW. We sample images from the resulting SPNs, and we show the effect of using discriminative training on the model parameters. The implementation was coded in C++, and the code is publicly available at `https://github.com/arashwan/ebw_discriminative_spns`.

Table 1: The accuracies of EBW, generative EM, and discriminative GD on the test data of eight different datasets.

Dataset	Var#	Dataset Size	Classes#	Var Type	EBW	genEM	discGD
Banknote	4	1371	2	Binary	**86.13%**	83.94%	**86.13%**
Voice	20	3167	2	Cont	**97.15%**	96.20%	96.20%
Credit Card	29	284806	2	Cont	**99.92%**	99.38%	**99.92%**
Breast Cancer	30	865	2	Cont	**96.42%**	92.85%	91.07%
Sensorless Drive	48	85508	11	Cont	**99.44%**	99.36%	55.41%
Fault Detection	70	14354	41	Cont	**60.45%**	58.67%	58.12%
Activity Recognition	561	10299	6	Cont	**90.53%**	88.66%	76.45%
MNIST	784	70000	10	Binary	**95.07%**	93.35%	62.89%

For all experiments, we generated dense SPNs by using a variant of the algorithm proposed in (Poon and Domingos, 2011). We recursively construct the SPN structure in a top down fashion as follows. We treat the variables of each problem as a 1D array (or 2D array in the case of MNIST) based on the order of the features in the data. For each sum node, we construct children product nodes corresponding to all splits of the scope in two sub-arrays of variables (all vertical and horizontal splits in two 2D arrays in the case of MNIST). We stop when the scope has a single variable, in which case, a univariate leaf distribution is generated. To control the size of the network, we randomly skip some partitions.

4.1 EBW versus Other Parameter Learning Algorithms

In this experiment, we used eight different datasets[2] that span a wide spectrum of domains. The number of classes ranges from 2 to 41, and the number of variables ranges from 4 to 784. For the datasets that don't have training and testing splits, we use 10% of the data for testing and the rest for training. While we applied the algorithm on datasets where the variables are binary and continuous, our implementation can also handle categorical variables.

EBW has one hyper-parameter D. Initializing D to 0.1 and increasing it after each epoch by 0.1 produces the best results. Generative EM doesn't have any hyper-parameters. For gradient descent, we found that initializing the learning rate to 1 and decreasing it after each epoch by multiplying by 0.9 produces the best results. During the experiments, we limited the number of epochs to 20.

Table 1 shows that EBW always outperforms genEM. We also observed that discGD converges quickly to good solutions for small and shallow SPNs, but not deep SPNs, which suggests that it suffers from the gradient vanishing problem.

We explored the convergence speeds for EBW and discGD on the training data. We ran every algorithm for 20 epochs. Figure 2 shows the convergence speed and performance for both algorithms. Both algorithms take the same time per epoch. The figure shows that EBW converges faster to a better solution than discGD. Furthermore, discGD struggles to achieve good results consistently as we can see in the Activity Recognition plot where it didn't converge to a solution in 20 epochs while EBW was able to converge after a few epochs.

2. The datasets are publicly available at archive.ics.uci.edu/ml/ and kaggle.com except fault detection, which is a private dataset collected by Huawei.

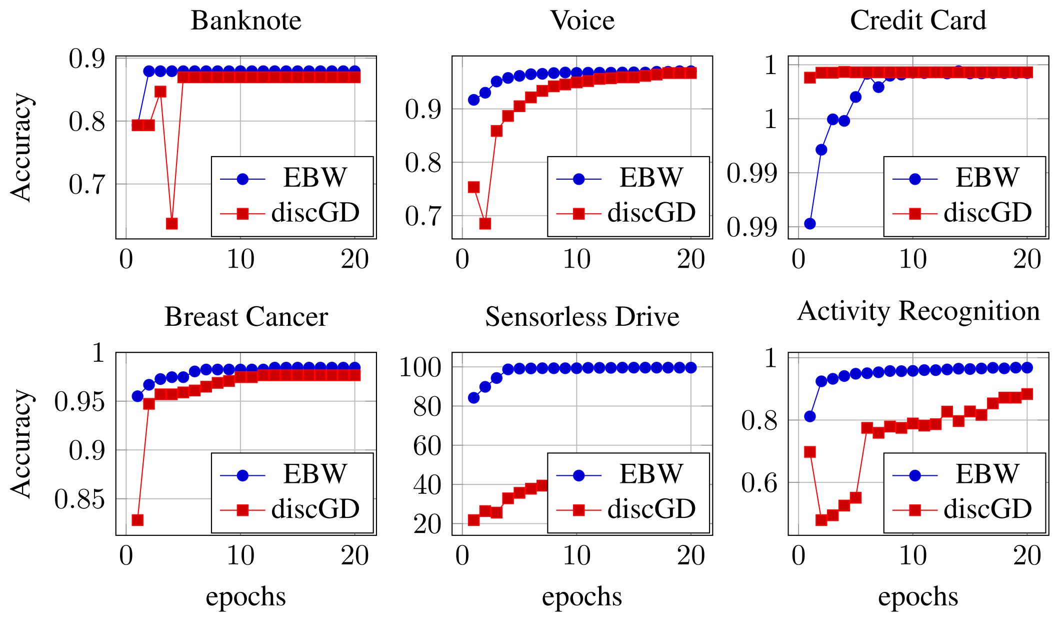


Figure 2: Accuracies on training data versus number of epochs for EBW and discriminative GD algorithms. The time per epoch is the same for both algorithms.

4.2 EBW for Problems with Missing Features

Missing features is a problem that commonly happens in wearable devices where sensors can fail frequently. SPNs can naturally handle missing features by summing out the corresponding unobserved variables when doing inference. We show the robustness of SPNs trained using EBW in the absence of some features. We compare the performance of SPNs to SVMs and Neural Networks.

We randomly set 50% of the features for each instance to be missing, SPNs can handle such missing features by summing/integrating out the leaf distributions. Since SVMs and NNs need values for all features, we set the missing features to their average.[3] For SVMs, a polynomial kernel was used and the penalty constant was tuned for best performance. For NNs, we limited the number of parameters to be equal to the number of parameters in SPNs. We used neural networks with two hidden layers and rectified linear units (ReLU) in each layer. We set the width of each layer such that the number of parameters for the NNs is the same as the SPNs. We used TensorFlow to build and train NNs. We set the number of epochs to 20. Table 2 shows that EBW-trained SPNs are consistently more robust to missing features than both SVMs and NNs.

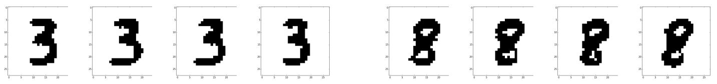
4.3 Visualizing the Parameters Learned by EBW

This experiment aims at visualizing the parameters learned by both EBW and generative EM by sampling different images from the learned SPNs. We chose two classes, ’3’ and ’8’, with visual similarities from the MNIST dataset. We trained an SPN using generative EM and a second SPN

3. It is possible to deal with missing features in kernel methods in a principled way by modifying the loss function to take into account the uncertainty induced by the missing features, however modeling assumptions are needed and the optimization problem is changed Pelckmans et al. (2005). Alternatively, one can also deal with missing features by casting kernel methods as estimation problems in exponential families, but this yields non-convex optimization problems Smola et al. (2005)

Table 2: The test accuracies of EBW-trained SPNs, SVMs and NNs on seven datasets.

Algorithm	EBW		NN		SVM	
Dataset	0%	50%	0%	50%	0%	50%
Banknote	86.13%	94.90%	86.13%	62.04%	86.13%	54.74%
Voice	97.15%	88.60%	97.46%	88.60%	96.20%	77.53%
Credit Card	99.92%	99.80%	99.87%	99.80%	99.96%	99.73%
Breast Cancer	96.24%	89.28%	94.60%	83.92%	94.64%	87.50%
Sensorless Drive	99.44%	52.03%	96.03%	47.90%	75.50%	12.40%
Fault Detection	60.45%	48.40%	50.01%	46.80%	57.04%	47.09%
Activity Recognition	90.53%	88.59%	93.82%	81.57%	96.23%	61.14%


(a) Sampled images for digits '3' and '8' from generatively trained SPNs

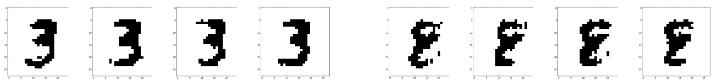

(b) Sampled images for digits '3' and '8' from discriminatively trained SPNs

Figure 3: The above samples show the effect of training SPNs discriminatively. The sampled images for digit '8' in the discriminative training case illustrate that the SPN for digit '8' was tuned to focus on the parts that discriminate the digit '8' from the digit '3'.

using discriminative EBW. We sampled images from the resulting SPNs to analyze the effect of discriminative EBW on the learned parameters.

As shown in Figure 3, the sampled images from the generatively trained SPN resemble the appearance of digit '3' and digit '8'. On the other hand, the sampled images from the discriminatively trained SPN show the parts of the digits '3' and '8' that are discriminative. We know that the left part of digit '8' differentiates it from digit '3', which is what was learned by the SPN.

5. Conclusion

We described a framework to train SPNs discriminatively using Extended Baum-Welch. We did so by formulating the conditional likelihood as a rational function and applied Extended Baum-Welch to maximize this function. We derived the update formulas for cases where the leaf nodes are either multinomial or univariate normal distributions. The experiments show that EBW outperforms generative EM and discriminative gradient descent in a wide variety of applications. We demonstrated the advantage of SPNs for classification tasks when some features are missing. We also illustrated the effect of learning the parameters of SPNs using EBW.

References

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

L. E. Baum, J. A. Eagon, et al. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Amer. Math. Soc*, 1967.

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 1970.

A. Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 2003.

M. Desana and C. Schnörr. Expectation maximization for sum-product networks as exponential family mixture models. *arXiv:1604.07243*, 2016.

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

P. S. Gopalakrishnan, D. Kanevsky, A. Nádas, and D. Nahamoo. An inequality for rational functions with applications to some statistical estimation problems. *IEEE Transactions on Information Theory*, 1991.

W. Hsu, A. Kalra, and P. Poupart. Online structure learning for sum-product networks with Gaussian leaves. *arXiv preprint arXiv:1701.05265*, 2017.

P. Jaini and P. Poupart. Online and distributed learning of Gaussian mixture models by Bayesian moment matching. *arXiv preprint arXiv:1609.05881*, 2016.

T. Jebara and A. Pentland. Maximum conditional likelihood via bound maximization and the CEM algorithm. In *NIPS*, 1998.

T. Jebara and A. Pentland. On reversing Jensen's inequality. In *NIPS*, 2000.

A. Klautau, N. Jevtic, and A. Orlitsky. Discriminative Gaussian mixture models: A comparison with kernel classifiers. In *ICML*, 2003.

A. Molina, S. Natarajan, and K. Kersting. Poisson sum-product networks: A deep architecture for tractable multivariate Poisson distributions. In *AAAI*, 2017.

A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, and K. Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *AAAI*, 2018.

Y. Normandin. *Hidden Markov models, maximum mutual information estimation, and the speech recognition problem*. PhD thesis, McGill University, Montreal, 1991.

Y. Normandin and S. D. Morgera. An improved MMIE training algorithm for speaker-independent, small vocabulary, continuous speech recognition. In *ICASSP*, 1991.

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

K. Pelckmans, J. De Brabanter, J. A. Suykens, and B. De Moor. Handling missing values in support vector machine classifiers. *Neural Networks*, 18(5-6):684–692, 2005.

F. Pernkopf and M. Wohlmayr. *Large Margin Learning of Bayesian Classifiers Based on Gaussian Mixture Models*. Springer Berlin Heidelberg, 2010.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *UAI*, 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*, 2014.

J. Salojärvi, K. Puolamäki, and S. Kaski. Expectation maximization algorithms for conditional likelihoods. In *ICML*, 2005.

A. J. Smola, S. Vishwanathan, and T. Hofmann. Kernel methods for missing variables. In *AISTATS*, 2005.

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.