Discriminative Training of Feed-Forward and Recurrent Sum-Product Networks by Extended Baum-Welch

Haonan Duan, Abdullah Rashwan, Pascal Poupart

University of Waterloo, Waterloo AI Institute, Ontario, Canada
Vector Institute, Toronto, Canada

Zhitang Chen

Huawei Technologies, Hong Kong, China

Abstract

We present a discriminative learning algorithm for feed-forward Sum-Product Networks (SPNs) (Poon and Domingos, 2011) and recurrent SPNs (Melibari et al., 2016) 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 and RSPNs 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.

Keywords:
Sum-product network, extended Baum-Welch, discriminative learning

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

Email addresses: h4duan@uwaterloo.ca (Haonan Duan), arashwan@uwaterloo.ca (Abdullah Rashwan), ppoupart@uwaterloo.ca (Pascal Poupart), chenzhitang2@huawei.com (Zhitang Chen)

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 (Conaty et al., 2017) (Mei et al., 2018).

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., 2016a), sequential monomial approximations and the concave-convex procedure (Zhao et al., 2016b). 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.

Recurrent sum-product networks (RSPNs), also known as dynamic SPNs, are a generalization of SPNs for modelling sequence data of varying length (Melibari et al., 2016). RSPNs can be used in sentiment analysis, speech recognition and protein sequencing. Similar to most other dynamic graphical models (Bilmes, 2010), an RSPN includes a template network that is repeated as many times as the number of time slices in a data sequence. The common generative learning algorithms for RSPNs are EM and gradient descent (Melibari et al., 2016). Kalra et al. (2018) introduced an online parameter and structure learning algorithm for RSPNs. However, no discriminative learning algorithm for RSPNs has been proposed yet.

In this paper, we present a novel algorithm to train SPNs and RSPNs discriminatively based on the Extended Baum-Welch technique. While Expectation Max-

imization 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, 1999, 2001; 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 and recurrent SPNs, we will formulate the conditional distribution as a rational function. We will develop the algorithm with both multinomial and univariate Gaussian distributions at the leaves.

The paper is structured as follows, Section 2 reviews SPNs, RSPNs, 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 describes how to extend the proposed techniques to RSPNs. Section 5 presents three sets of experiments that we carried out to evaluate the performance of our algorithms. Section 6 concludes the paper.

2. Background

2.1. Sum-Product Networks

A sum-product network (SPN) (Poon and Domingos, 2011) is a probabilistic graphical model that can be used to express a joint distribution over random variables.

Definition 2.1 (Sum-product network (Poon and Domingos, 2011)). An SPN over random variables $x_1, ..., x_n$ is a rooted acyclic directed graph where the interior nodes either compute products (product nodes) or weighted sum (sum nodes) of their inputs and the leaves are univariate probability distributions for one of the random variables $x_1, ..., x_n$. The edges emanating from sum nodes are labeled with weights θ_{ij} (where i is the source node, j is the destination node, and $\theta_{ij} > 0$).

Note that Definition 2.1 is slightly different from the one proposed by Poon and Domingos (2011) in that the random variables are not restricted to be binary.

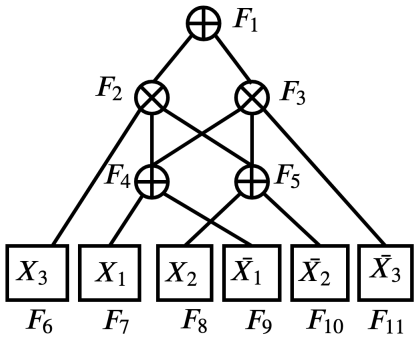


Figure 1: An example of SPN

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).

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 i as follows:

$$F_i(\mathbf{x}) = \begin{cases} P(\mathbf{X}_i = \mathbf{x}_i) & i \text{ is a leaf} \\ \prod_{j \in children(i)} F_j(\mathbf{x}) & i \text{ is a product} \\ \sum_{j \in children(i)} \theta_{ij} F_j(\mathbf{x}) & i \text{ is a sum} \end{cases} \tag{1}$$

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

Figure 1 shows an example of an SPN with three Bernoulli random variables, where $\bigoplus$ represents a sum node and $\bigotimes$ represents a product node. The root function $F(x_1, x_2, x_3) = \theta_{12}F_2 + \theta_{13}F_3 = \theta_{12}x_3F_4F_5 + \theta_{13}\bar{x}_3F_4F_5 = (\theta_{12}x_3 + \theta_{13}\bar{x}_3)F_4F_5 = (\theta_{12}x_3 + \theta_{13}\bar{x}_3)(\theta_{47}x_1 + \theta_{49}\bar{x}_1)(\theta_{58}x_2 + \theta_{510}\bar{x}_2)$

The value of an SPN is the value of its root, which can be naturally evaluated in a bottom-up fashion. Computing partial derivatives of an SPN with respect

to any node can be done efficiently using the backpropogation algorithm of a computation graph, which is an inexpensive procedure that applies the chain rule in calculus recursively (Goodfellow et al., 2016). The backpropagation algorithm applied on an SPN to compute partial derivatives is given in Algorithm 1 (Gens and Domingos, 2012):

Algorithm 1: Backpropogation algorithm for partial derivatives of SPNs

Input: A valid SPN and an array F which stores the evaluation of each node, where F_i denotes the value of node i

Output: Partial derivatives with respect to all nodes, $\frac{\partial F_{root}}{\partial F_i}$

```
Initialize ∂F_root/∂F_i = 0 except ∂F_root/∂F_root = 1
for all nodes i in top down order do
    if i is a sum node then
        for all children j of node i do
            ∂F_root/∂F_j += θ_ij ∂F_root/∂F_i
    else if i is a product node then
        for all children j of node i do
            ∂F_root/∂F_j += ∂F_root/∂F_i F_i/F_j
    else if i is a leaf node then
        continue
```

F_i is the value of the sum node based on θ_{ij} and F_j is the value of the child node. Using the chain rule, we obtain:

$$\frac{\partial F_{root}}{\partial \theta_{ij}} = F_j \frac{\partial F_{root}}{\partial F_i} \tag{2}$$

Using Equation 2 and Algorithm 1, we can calculate $\frac{\partial F_{root}}{\partial \theta_{ij}}$ in linear time.

An SPN is said to be *valid* when the root node always correctly computes a value proportional to the probability of evidence for any evidence (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 2.2 (**Scope**). If node n is a leaf with base distribution $P(X = x)$, then $scope(n) = \{X\}$, otherwise $scope(n) = \cup_{j \in children(n)} scope(j)$.

In other words, the scope of a node n is the set of all variables that appear in

the leaves of the sub-SPN rooted at n. For example, in Figure 1, the scope of node 9 is $\{X_1\}$ and the scope of node 1 is $\{X_1, X_2, X_3\}$

Definition 2.3 (**Decomposability**)**.** An SPN is decomposable when each product node has children with disjoint scopes.

Definition 2.4 (**Completeness**)**.** An SPN is complete when each sum node has children with identical scope.

Theorem 2.1. (Poon and Domingos, 2011) If an SPN is decomposable and complete, then it is valid.

In the following discussion of the paper, all SPNs refer to decomposable and complete SPNs unless otherwise stated.

A valid 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.

$$P(\mathbf{X} = \mathbf{x}) = \frac{F_{root}(\mathbf{x})}{F_{root}(\emptyset)} \tag{3}$$

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. Equation 3 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:

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{Z} = \mathbf{z}) = \frac{P(\mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z})}{P(\mathbf{Z} = \mathbf{z})} = \frac{F_{root}(\mathbf{y}, \mathbf{z})}{F_{root}(\mathbf{z})} \tag{4}$$

For a valid SPN, 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]

[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.

2.2. *Recurrent Sum-Product Network*

Static graphical models, such as SPNs and Bayesisan networks, can only be used to learn data of fixed length. However, in some domains like speech recognition and sentiment analysis, we are often given sequence data of varying length. Dynamic graphical models consist of a static graphical model $G(V, E)$ (often called template) that is repeated as many times as the length of a data sequence (Bilmes, 2010). Dynamic Bayesian Networks (Murphy, 2002) are examples of dynamic graphical models. DBNs generalize BNs in a sense that BNs are used as templates for DBNs. Any static graphical model can be generalized into a dynamic graphical model (Bilmes, 2010).

RSPNs are dynamic graphical models with templates being SPNs. Thus, one advantage of RSPNs compared with most other dynamic graphical models is that (marginal) inference complexity for RSPNs is linear in the length of data sequences and the size of the template network.

Consider a data sequence of T time slices with each time slice containing n variables. This sequence can be modeled by an RSPN which consists of $T - 1$ template networks on top of a bottom network and capped by a top network (Melibari et al., 2016). The definitions of template network, bottom network and top network are given below.

Definition 2.5 (**Template Network**)**.** A template network is a directed acyclic graph with k roots and $k + d$ leaf nodes (where $d \geq n$). The d leaves represent univariate distributions of each data component in one time slice, and the remaining k leaves (interface nodes) are shared with the previous time slice (input interface nodes). The k roots are interface nodes shared with the next time slice (output interface nodes).

Definition 2.6 (**Bottom Network**)**.** A bottom network is a directed acyclic graph with k roots and d leaf nodes. The d leaf nodes are univariate distributions of each data component in one time slice. The k root nodes are output interface nodes connected to the second time slice.

Definition 2.7 (**Top Network**)**.** A top network is a directed acyclic graph with one root and k leaf nodes. The k leaves are input interface nodes, which also appear in the top of the template network at the last time slice.

Figure 2 is a template network based on the SPN in Figure 1, and Figure 3 is an RSPN using Figure 2 as the template network.

Melibari et al. (2016) proved that any RSPN is also a valid SPN under certain invariance constraints, which allows us to check validity of an RSPN without

having to unroll the network and evaluate the scope of each sum and product node.

Definition 2.8 (**Invariance (Melibari et al., 2016)**). Define T to be a template network over data in one step $< X_1, ..., X_n >^t$ and I be the set of input interface nodes of T. Let f be a bijective mapping between input interface nodes and output interface nodes. Let the scope of each input interface node i of the template network be the same as the scope of the corresponding output interface node j in the bottom network (i.e., $scope(i) \leftarrow scope(j)$ where $j = f(i)$). Then T is invariant if the following properties hold:

1. $\forall i \in I, scope(i) \cap \{X_1^t, ..., X_n^t\} = \emptyset$
2. $\forall i, j \in I, scope(i) \cap scope(j) = \emptyset \vee scope(i) = scope(j)$
3. The bijective mapping f from input interface nodes to output interface nodes satisfies the following properties: $\forall i, j \in I, scope(i) \cap scope(j) = \emptyset \iff scope(f(i)) \cap scope(f(j)) = \emptyset$ and $\forall i, j \in I, scope(i) = scope(j) \iff scope(f(i)) = scope(f(j))$
4. All interior and output sum nodes are complete
5. All interior and output product nodes are decomposable

Theorem 2.2 ((**Melibari et al., 2016**)). An RSPN is a valid SPN if the following property holds:

1. The bottom network is complete and decomposable
2. The scopes of all pairs of output interface nodes of the bottom network are either identical or disjoint
3. The template network is invariant
4. The top network is complete and decomposable

One can check that the template network in Figure 2 is invariant based on Definition 2.8 and the RSPN in Figure 3 is complete and decomposable using Theorem 2.2.

2.3. Extended Baum-Welch Algorithm

The Baum-Welch (BW) algorithm was introduced in 1970 to estimate the parameters for HMMs (Baum et al., 1970). BW provides an iterative scheme that monotonically increases the value of homogeneous polynomials with nonnegative coefficients over the probability simplex. The algorithm was then extended to

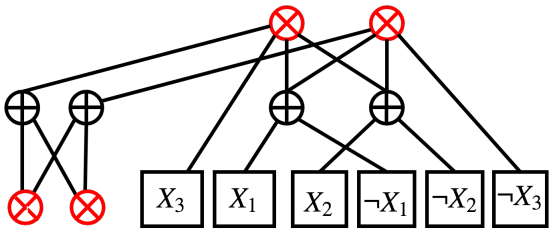


Figure 2: A template network for RSPNs based on Figure 1

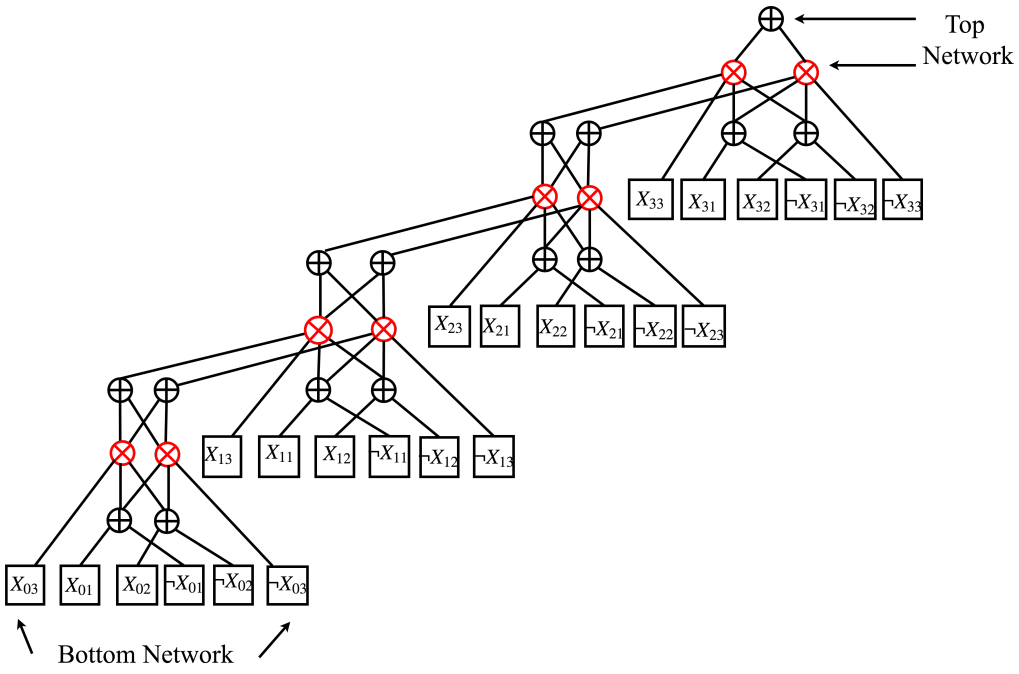


Figure 3: An RSPN which uses Figure 2 as the template network

maximize 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 (Pernkopf and Wohlmayr, 2010; 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 2.3. (Baum et al., 1967) Define $S(\theta)$ be a homogeneous degree d polynomial with non-negative coefficients. Let $\bar{\theta} := \{\bar{\theta}_{ij}\}$ be any point in the domain $\mathcal{D} : \sum_j \bar{\theta}_{ij} = 1, \forall i$. Then $\hat{\theta} = T(\bar{\theta}) = T(\{\bar{\theta}_{ij}\})$ is a transformation function:

$$\hat{\theta}_{ij} = \frac{\bar{\theta}_{ij} \frac{\partial S}{\partial \theta_{ij}}(\bar{\theta})}{\sum_j \bar{\theta}_{ij} \frac{\partial S}{\partial \theta_{ij}}(\bar{\theta})}, \tag{5}$$

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 2.3 is applied iteratively to optimize a polynomial $S(\theta)$. The transformation $T(\bar{\theta})$ is called a growth transformation 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 Equation 5 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.

Gopalakrishnan et al. (1991) extended Theorem 2.3 to rational functions. In what follows, we will give an overview of how Gopalakrishnan et al. (1991) reduced the problem of finding growth transformations for rational functions to finding growth transformations for homogeneous polynomials with non-negative coefficients.

Let $R_d(\theta) = \frac{S_1(\theta)}{S_2(\theta)}$, where $S_1(\theta), S_2(\theta)$ are two polynomials defined in domain $\mathcal{D} : \sum_j \theta_{ij} = 1$. Then finding a growth transformation for $R_d(\theta)$ can be reduced to the problem of homogeneous polynomials with non-negative coefficients (which allows us to apply Theorem 2.3) in the following steps:

1. For any $x \in \mathcal{D}$, construct a polynomial $P_x(\theta) := S_1(\theta) - R_d(x)S_2(\theta)$. It can be shown that if $P_x(y) > P_x(x), \forall y \in \mathcal{D}$, then $R_d(y) > R_d(x)$. In this way, the problem of finding growth transformations for rational functions ($R_d(\theta)$) is reduced to polynomials ($P_x(\theta)$).

2. In this step, Gopalakrishnan et al. (1991) found a polynomial with nonnegative coefficients such that any growth transformation of that polynomial is also a growth transformation of $P_x(\theta)$. Let d be the degree of $P_x(\theta)$ and a be the minimal negative coefficient. Define $C(\theta) := -a(\sum_{i,j} \theta_{ij} + 1)^d$ and $P'_x(\theta) := P_x(\theta) + C(\theta)$. Then it can be shown easily that the coefficients of $P'_x(\theta)$ are all nonnegative and $C(\theta)$ is a constant on $\mathcal{D}$ (thus any growth transformation of $P'_x(\theta)$ is also a growth transformation of $P_x(\theta)$).
3. Gopalakrishnan et al. (1991) showed that $P'_x(\theta)$ has an equivalence representation as a homogeneous polynomial. Let d be the degree of $P_x(\theta)$ and $\beta = 1$. Then $\beta^d P'_x(\frac{\theta}{\beta})$ is a homogeneous polynomial and $\forall \theta \in \mathcal{D}, P'_x(\theta) = \beta^d P'_x(\frac{\theta}{\beta})$.

Based on the above three steps, Gopalakrishnan et al. (1991) proposed an iterative algorithm to monotonically increase rational functions on the probability simplex.

Theorem 2.4. (Gopalakrishnan et al., 1991) Define $R_d(\theta)$ to be a rational function. There exists a sufficiently large constant D such that T^D is a growth transformation for $R_d(\theta)$:

$$(T^C(\theta))_{ij} = \frac{\theta_{ij}\left(\frac{\partial P_x}{\partial \theta_{ij}}(\theta) + D\right)}{\sum_j \theta_{ij}\left(\frac{\partial P_x}{\partial \theta_{ij}}(\theta) + D\right)} \tag{6}$$

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. Gopalakrishnan et al. (1991) point out that the D required in Theorem 2.4 is too large to be useful. In practice, we will choose D to be slightly larger than the minimum required for all weights of sum nodes and variance of Gaussian leaves to be positive.

In discriminative learning, the conditional probability can typically be expressed as a rational function $R_d(\theta)$ (i.e., the data likelihood divided by the marginal of the inputs).

In the next two sections, we will show how to construct a polynomial $S(\theta)$ from the rational function corresponding to the conditional probability of SPNs and RSPNs and then apply the growth function to improve the conditional probability monotonically.

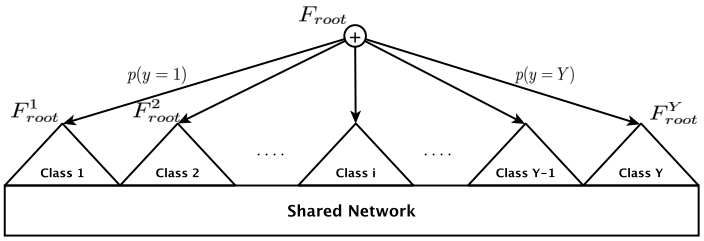


Figure 4: Class conditional SPN architecture.

3. Discriminative learning for 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 training of SPNs, then extend the discriminative gradient descent technique proposed by 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 SPN architecture shown in Figure 4 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 probability 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:

$$P(y|\mathbf{x}) = \frac{p(y)p(\mathbf{x}|y)}{p(\mathbf{x})} = \frac{p(y)F^y_{root}(\mathbf{x})}{F_{root}(\mathbf{x})} \tag{7}$$

The label associated with the sub-SPN that maximizes the conditional probability distribution is selected as follows:

$$\arg max_{y=1,...,Y} p(y) F^y_{root}(\mathbf{x}) \tag{8}$$

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

$$P(\mathbf{Y}|\mathbf{X}) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) = \prod_{n=1}^{N} \frac{P(y_n)F_{root}^{y_n}(\mathbf{x}_n)}{F_{root}(\mathbf{x}_n)} \tag{9}$$

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

$$log(P(\mathbf{Y}|\mathbf{X})) = \sum_{n=1}^{N} log(p(y_n)F_{root}^{y_n}(\mathbf{x}_n)) - log\big(F_{root}(\mathbf{x}_n)\big) \tag{10}$$

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 $F_{root}^{y}(\mathbf{x})$, is harder and usually does not have a closed form solution. Iterative methods are used in this case.

We will use $F_j^y(\mathbf{x})$ 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 $P(\mathbf{x}|y) = F_{root}^{y}(\mathbf{x})$. 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_{ij} in the SPN, where subscript i indicates that the parent node of the parameter is node i and the child node is j. The derivative of the log likelihood can be obtained as follows:

$$\begin{aligned}\frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial \theta_{ij}} &= \sum_{n=1}^{N} \frac{p(y_n)\frac{\partial F_{root}^{y_n}}{\partial \theta_{ij}}(\mathbf{x}_n)}{p(y_n)F_{root}^{y_n}(\mathbf{x}_n)} - \frac{\frac{\partial F_{root}}{\partial \theta_{ij}}(\mathbf{x}_n)}{F_{root}(\mathbf{x}_n)} \\ &= \sum_{n=1}^{N} \frac{1}{F_{root}^{y_n}(\mathbf{x}_n)}\frac{\partial F_{root}^{y_n}}{\partial \theta_{ij}}(\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)}\frac{\partial F_{root}}{\partial \theta_{ij}}(\mathbf{x}_n)\end{aligned} \tag{11}$$

Here, $\frac{\partial F_{root}^{y_n}}{\partial \theta_{ij}}(\mathbf{x}_n)$ and $\frac{\partial F_{root}}{\partial \theta_{ij}}(\mathbf{x}_n)$ can be calculated using Equation 2 in linear time. If θ_{ij} only appears in sub-SPN F^y , then Equation 11 can be further simplified. In this case, we use θ^y to denote the parameter appearing only in sub-SPN F^y.

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

$$\frac{\partial F_{root}^{y_n}}{\partial \theta^y} = \mathbb{1}_{[y \in SPN_{y_n}]}\frac{\partial F_{root}^{y}}{\partial \theta^y} \tag{12}$$

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:

$$\frac{\partial F_{root}}{\partial \theta^y}(\mathbf{x}_n) = p(y) \frac{\partial F^y_{root}}{\partial \theta^y}(\mathbf{x}_n) \tag{13}$$

Finally, based on Equations 12 and 13 we can rewrite Equation 11 as follows:

$$\frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial \theta^y} = \sum_{n=1}^{N} \frac{\partial F^y_{root}}{\partial \theta^y}(\mathbf{x}_n) \left[\frac{\mathbb{1}_{[y=y_n]}}{F^{y_n}_{root}(\mathbf{x}_n)} - \frac{p(y)}{F_{root}(\mathbf{x}_n)} \right] = \sum_{n=1}^{N} \frac{\partial F^y_{root}}{\partial \theta^y}(\mathbf{x}_n) \gamma_n \tag{14}$$

where $\gamma_n \geq 0$.

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 Equation 11, the following formula can be used to update each edge weight θ_{ij} by taking a small step η in the direction of the gradient.

$$\theta_{ij} \leftarrow \theta_{ij} + \eta \frac{\partial log(P(\mathbf{Y}|\mathbf{X}))}{\partial \theta_{ij}} \tag{15}$$

For univariate Gaussian nodes, we use μ_{uv} and σ^2_{uv} to denote the mean and variance of leaf node u for variable v. μ_{uv} and $(\sigma^2)_{uv}$ can be updated by taking a small step η in the direction of the gradient.

$$\mu_{uv} \leftarrow \mu_{uv} + \eta \sum_n \frac{x_{nv} - \mu_{uv}}{(\sigma^2)_{uv}} \mathcal{N}_{uv}(x_{nv}) \left(\frac{1}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial F_{uv}}(\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial F_{uv}}(\mathbf{x}_n) \right) \tag{16}$$

$$(\sigma^2)_{uv} \leftarrow (\sigma^2)_{uv} + \eta \sum_n \frac{\mathcal{N}_{uv}(x_{nv})}{2(\sigma^2)_{uv}} \left[\frac{(x_{nv} - \mu_{uv})^2}{(\sigma^2)_{uv}} - 1 \right] \left(\frac{1}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial F_{uv}}(\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial F_{uv}}(\mathbf{x}_n) \right) \tag{17}$$

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 2.3 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 $\theta = \{\theta_{ij}\}$ of the polynomial satisfy the condition $\sum_j \theta_{ij} = 1$ since the sum of the weights for each sum node is one. To apply Theorem 2.3, 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

$$\frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial \theta_{ij}} = \frac{1}{P(\mathbf{X}|\mathbf{Y})} \frac{\partial P(\mathbf{X}|\mathbf{Y})}{\partial \theta_{ij}} \tag{18}$$

Hence, we can rewrite Equation 5 as follows.

$$\hat{\theta}_{ij} = \frac{\theta_{ij} P(\mathbf{X}|\mathbf{Y}) \frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial \theta_{ij}}}{\sum_j \theta_{ij} P(\mathbf{X}|\mathbf{Y}) \frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial \theta_{ij}}} = \frac{\theta_{ij} \frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial \theta_{ij}}}{\sum_j \theta_{ij} \frac{\partial log(P(\mathbf{X}|\mathbf{Y}))}{\partial \theta_{ij}}} \tag{19}$$

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

In discriminative training for SPNs, applying Expectation-Maximization or CCCP does not lead to a closed form update formula (Gens and Domingos, 2012). Jebara and Pentland (1999, 2001) 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 EBW on conditional likelihood, let's derive an equation about $log(R_d(\theta)$:

$$
\begin{aligned}
\frac{\partial log(R_d(\theta))}{\partial \theta_{ij}} &= \frac{\partial log(S_1(\theta))}{\partial \theta_{ij}} - \frac{\partial log(S_2(\theta))}{\partial \theta_{ij}} \\
&= \frac{1}{S_1(\theta)} \frac{\partial S_1(\theta)}{\partial \theta_{ij}} - \frac{1}{S_2(\theta)} \frac{\partial S_2(\theta)}{\partial \theta_{ij}} \\
&= \frac{1}{S_1(\theta)} (\frac{\partial S_1(\theta)}{\partial \theta_{ij}} - \frac{S_1(\theta)}{S_2(\theta)} \frac{\partial S_2(\theta)}{\partial \theta_{ij}}) \\
&= \frac{1}{S_1(\theta)} (\frac{\partial S_1(\theta)}{\partial \theta_{ij}} - R_d(\theta) \frac{\partial S_2(\theta)}{\partial \theta_{ij}}) \\
&= \frac{1}{S_1(\theta)} \frac{\partial P_x(\theta)}{\partial \theta_{ij}}
\end{aligned}
\tag{20}
$$

Based on equation 20 and Theorem 2.4, Gopalakrishnan et al. (1991) proposed the following update formula:

$$
\hat{\theta}_{ij} = \frac{\bar{\theta}_{ij} \left[\frac{\partial log R_d}{\partial \theta_{ij}} (\bar{\theta}) + D \right]}{\sum_j \left[\bar{\theta}_{ij} \frac{\partial log R_d}{\partial \theta_{ij}} (\bar{\theta}) \right] + D} \tag{21}
$$

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 θ_{ij}, the mean μ_{uv}, and the variance $(\sigma^2)_{uv}$.

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

$$
\hat{\theta}_{ij} = \frac{\theta_{ij} \left[\sum_n \frac{1}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial \theta_{ij}} (\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial \theta_{ij}} (\mathbf{x}_n) \right] + D\theta_{ij}}{\left[\sum_{j \in Children(i)} \sum_n \frac{1}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial \theta_{ij}} (\mathbf{x}_n) - \frac{1}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial \theta_{ij}} (\mathbf{x}_n) \right] + D} \tag{22}
$$

Normandin and Morgera (1991) proposed a discrete approximation for univariate Gaussian distributions that allows us to update μ_{uv} and $(\sigma^2)_{uv}$:

$$
\hat{\mu}_{uv} = \frac{\left[\sum_n \left(\frac{\mathcal{N}_{uv}(x_{nv})}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial F^{uv}} (\mathbf{x}_n) - \frac{\mathcal{N}_{uv}(x_{nv})}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial F^{uv}} (\mathbf{x}_n) \right) x_{nv} \right] + D\mu_{uv}}{\left[\sum_n \left(\frac{\mathcal{N}_{uv}(x_{nv})}{F^{y_n}_{root}(\mathbf{x}_n)} \frac{\partial F^{y_n}_{root}}{\partial F^{uv}} (\mathbf{x}_n) - \frac{\mathcal{N}_{uv}(x_{nv})}{F_{root}(\mathbf{x}_n)} \frac{\partial F_{root}}{\partial F^{uv}} (\mathbf{x}_n) \right) \right] + D} \tag{23}
$$

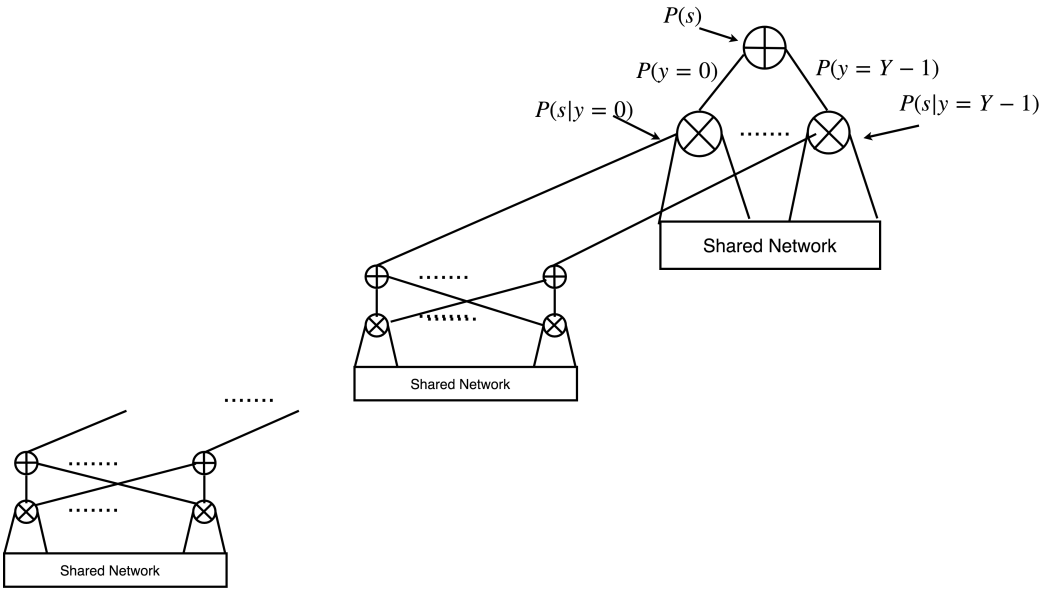


Figure 5: Discriminative RSPN structure

$$(\hat{\sigma^2})_{uv} = \frac{\left[\sum_n \left(\frac{\mathcal{N}_{uv}(x_{nv})}{F^{y_n}_{root}(\mathbf{x}_n)}\frac{\partial F^{y_n}_{root}}{\partial F^{uv}}(\mathbf{x}_n) - \frac{\mathcal{N}_{uv}(x_{nv})}{F_{root}(\mathbf{x}_n)}\frac{\partial F_{root}}{\partial F^{uv}}(\mathbf{x}_n)\right) x^2_{nv}\right] + D\left[(\sigma^2)_{uv} + (\mu_{uv})^2\right]}{\left[\sum_n \left(\frac{\mathcal{N}_{uv}(x_{nv})}{F^{y_n}_{root}(\mathbf{x}_n)}\frac{\partial F^{y_n}_{root}}{\partial F^{uv}}(\mathbf{x}_n) - \frac{\mathcal{N}_{uv}(x_{nv})}{F_{root}(\mathbf{x}_n)}\frac{\partial F_{root}}{\partial F^{uv}}(\mathbf{x}_n)\right)\right] + D} - (\hat{\mu}_{uv})^2 \tag{24}$$

Kanevsky (2004) proved that the discrete probability approximation of Gaussian densities in Equations 23 and 24 along with Equation 21 are growth transformations for some sufficiently large D and when $R_d(\theta)$ is a rational function that satisfies some smoothness constraints.

4. Discriminative learning for Recurrent Sum-Product Networks

In this section, our goal is to classify multivariate sequential data. Let the training set be $\mathbf{S} = \{\mathbf{s}_1, ..., \mathbf{s}_N\}$ and the corresponding labels associated with each sequence $\mathbf{Y} = \{y_1, ..., y_N\}$, where $\mathbf{s}_i = ((s_{11}, ..., s_{1M}), ..., (s_{l_i 1}, ..., s_{l_i M}))$, $y_i \in \{0, ..., Y\}$, N is the number of training examples, M is the feature size, l_i is the length of the sequence s_i, and Y is the number of classes. We will derive

the formulas to learn RSPNs discriminatively using gradient descent and extended Baum-Welch.

Figure 5 shows an example of RSPNs' architecture used in discriminative learning, which is a generalization of the architecture in Figure 2. In the top network, each input interface node represents the conditional probability given one class. The template network is modelling only the explanatory variables during each time slice.

Similar to discriminative learning for SPNs, when training discriminative RSPNs, we are also trying to maximize the posterior $P(\mathbf{Y}|\mathbf{S})$.

$$P(\mathbf{Y}|\mathbf{S}) = \prod_{n=1}^{N} P(y_n|\mathbf{s}_n) = \prod_{n=1}^{N} \frac{P(y_n)F_n^{y_n}(\mathbf{s}_n)}{F_n^{root}(\mathbf{s}_n)} \tag{25}$$

However, different from SPNs, the polynomial function represented by RSPNs also depends on the length of the input data. We use subscript n to denote the different polynomials associated with each data sequence since they may have different lengths. The conditional log likelihood is computed as follows:

$$log(P(\mathbf{Y}|\mathbf{S})) = \sum_{n=1}^{N} log(p(y_n)F_n^{y_n}(\mathbf{s}_n)) - log\big(F_n^{root}(\mathbf{s}_n)\big) \tag{26}$$

Similar to Equation 11, the partial derivative of $log(P(\mathbf{Y}|\mathbf{X}))$ with respect to an arbitrary parameter θ_{ij} is:

$$\frac{\partial log(P(\mathbf{Y}|\mathbf{S}))}{\partial \theta_{ij}} = \sum_{n=1}^{N} \frac{1}{F_n^{y_n}(\mathbf{s}_n)} \frac{\partial F_n^{y_n}}{\partial \theta_{ij}}(\mathbf{s}_n) - \frac{1}{F_n^{root}(\mathbf{s}_n)} \frac{\partial F_n^{root}}{\partial \theta_{ij}}(\mathbf{s}_n) \tag{27}$$

Using the fact that same parameters are shared among time slices and applying the chain rule of multivariate calculus, we can extend Equation 2 to RSPNs:

$$\begin{aligned} \frac{\partial F_n}{\partial \theta_{ij}} &= \sum_{k=1}^{l_n} \frac{\partial F_n}{\partial F_{ki}} \frac{\partial F_{n,ki}}{\theta_{ij}} \\ &= \sum_{k=1}^{l_n} F_{n,kj} \frac{\partial F_n}{\partial F_{ki}} \end{aligned} \tag{28}$$

Here, l_n denotes the length of the data sequence n and $F_{n,kj}$ denotes the value of node j in the k-th time slice for the data sequence n.

Thus, Equation 27 can be rewritten as:

$$\frac{\partial log(P(\mathbf{Y}|\mathbf{S}))}{\partial \theta_{ij}} = \sum_{n=1}^{N} \left(\frac{1}{F_n^{y_n}(\mathbf{s}_n)} \sum_{k=1}^{l_n} F_{n,kj} \frac{\partial F_n^{y_n}}{\partial F_{ki}} - \frac{1}{F_n^{root}(\mathbf{s}_n)} \sum_{k=1}^{l_n} F_{n,kj} \frac{\partial F_n^{root}}{\partial F_{ki}} \right) \tag{29}$$

Since each RSPN is also a valid SPN, Algorithm 1 can be used to calculate $\frac{\partial F_n^{y_n}}{\partial F_{ki}}$ and $\frac{\partial F_n^{root}}{\partial F_{ki}}$.

4.1. Discriminative Learning for RSPNs Using Gradient Descent

Using Equation 29, the formula to update the weight of a sum node using gradient descent, θ_{ij} is:

$$\theta_{ij} \leftarrow \theta_{ij} + \eta \frac{\partial log(P(\mathbf{Y}|\mathbf{S}))}{\partial \theta_{ij}} \tag{30}$$

Similarly, for leaf univariate Gaussian distributions, using Equation 29 and applying the chain rule on the Gaussian density function, the formula to update μ_{uv} and $(\sigma^2)_{uv}$ is:

$$\begin{aligned} \mu_{uv} \leftarrow \mu_{uv} + \eta \Bigg(\sum_{n=1}^{N} \Bigg(& \frac{1}{F_n^{y_n}(\mathbf{s}_n)} \sum_{k=1}^{l_n} \mathcal{N}_{uv}(s_{nvk}) \frac{s_{nvk} - \mu_{uv}}{(\sigma^2)_{uv}} \frac{\partial F_n^{y_n}}{\partial F^{uvk}} \\ & - \frac{1}{F_n^{root}(\mathbf{s}_n)} \sum_{k=1}^{l_n} \mathcal{N}_{uv}(s_{nvk}) \frac{\partial F_n^{root}}{\partial F^{uvk}} \frac{s_{nvk} - \mu_{uv}}{(\sigma^2)_{uv}} \Bigg) \Bigg) \end{aligned} \tag{31}$$

$$\begin{aligned} \sigma_{uv}^2 \leftarrow \sigma_{uv}^2 + \eta \Bigg(\sum_{n=1}^{N} \Bigg(& \frac{1}{F_n^{y_n}(\mathbf{s}_n)} \sum_{k=1}^{l_n} \frac{\mathcal{N}_{uv}(s_{nvk})}{2(\sigma^2)_{uv}} \left[\frac{(s_{nvk} - \mu_{uv})^2}{(\sigma^2)_{uv}} - 1 \right] \frac{\partial F_n^{y_n}}{\partial F^{uvk}} \frac{s_{nvk} - \mu_{uv}}{(\sigma^2)_{uv}} \\ & - \frac{1}{F_n^{root}(\mathbf{s}_n)} \sum_{k=1}^{l_n} \frac{\mathcal{N}_{uv}(s_{nvk})}{2(\sigma^2)_{uv}} \left[\frac{(s_{nvk} - \mu_{uv})^2}{(\sigma^2)_{uv}} - 1 \right] \frac{\partial F_n^{root}}{\partial F^{uvk}} \frac{s_{nvk} - \mu_{uv}}{(\sigma^2)_{uv}} \Bigg) \Bigg) \end{aligned} \tag{32}$$

Here, s_{nvk} denotes the value of variable v in time slice k of data sequence s_n, and F_{uvk} is the leaf node u for variable v in the k-th time slice.

4.2. Discriminative Learning for RSPNs Using Extended Baum-Welch

Combining with Equation 21 and Equation 29, we can get the formula to update θ_{ij} using Extended Baum-Welch:

$$\begin{aligned}\hat{\theta}_{ij} &= \frac{\theta_{ij}(\frac{\partial log(P(\mathbf{Y}|\mathbf{S}))}{\partial \theta_{ij}} + D)}{\sum_j(\theta_{ij}\frac{\partial log(P(\mathbf{Y}|\mathbf{S}))}{\partial \theta_{ij}}) + D} \\ &= \frac{\theta_{ij}\left(\sum_n\left(\frac{1}{F_n^{y_n}(\mathbf{s}_n)}\sum_k F_{n,kj}\frac{\partial F_n^{y_n}}{\partial F^{ki}} - \frac{1}{F_n^{root}(\mathbf{s}_n)}\sum_k F_{n,kj}\frac{\partial F_n^{root}}{\partial F^{ki}}\right) + D\right)}{\sum_j\left(\theta_{ij}\left(\sum_n\left(\frac{1}{F_n^{y_n}(\mathbf{s}_n)}\sum_k F_{n,kj}\frac{\partial F_n^{y_n}}{\partial F^{ki}} - \frac{1}{F_n^{root}(\mathbf{s}_n)}\sum_k F_{n,kj}\frac{\partial F_n^{root}}{\partial F^{ki}}\right)\right)\right) + D}\end{aligned} \tag{33}$$

Using the discrete approximation for univariate Gaussian distributions suggested in Section 3.2, the formula to update μ_{uv} and σ_{uv}^2 is as follows:

$$\hat{\mu}_{uv} = \frac{\left[\sum_n\sum_k\left(\left(\frac{\mathcal{N}_{uv}(s_{njk})}{F_{root}^{y_n}(\mathbf{s}_n)}\frac{\partial F_{root}^{y_n}}{\partial F^{uvk}}(\mathbf{s}_n) - \frac{\mathcal{N}_{uv}(x_{njk})}{F_{root}(\mathbf{s}_n)}\frac{\partial F_n^{root}}{\partial F^{uvk}}(\mathbf{s}_n)\right)s_{nvk}\right)\right] + D\mu_{uv}}{\left[\sum_n\sum_k\left(\frac{\mathcal{N}_{uv}(s_{njk})}{F_{root}^{y_n}(\mathbf{s}_n)}\frac{\partial F_{root}^{y_n}}{\partial F^{uvk}}(\mathbf{s}_n) - \frac{\mathcal{N}_{uv}(x_{njk})}{F_n^{root}(\mathbf{s}_n)}\frac{\partial F_{root}}{\partial F^{uv}}(\mathbf{s}_n)\right)\right] + D} \tag{34}$$

$$\begin{aligned}(\hat{\sigma}^2)_{uv} = &\frac{\left[\sum_n\sum_k\left(\frac{\mathcal{N}_{uv}(s_{njk})}{F_{root}^{y_n}(\mathbf{s}_n)}\frac{\partial F_{root}^{y_n}}{\partial F^{uvk}}(\mathbf{s}_n) - \frac{\mathcal{N}_{uv}(x_{nvk})}{F_n^{root}(\mathbf{s}_n)}\frac{\partial F_n^{root}}{\partial F^{uvk}}(\mathbf{s}_n)\right)x_{nvk}^2\right] + D\left[(\sigma^2)_{uv} + (\mu_{uv})^2\right]}{\left[\sum_n\sum_k\left(\frac{\mathcal{N}_{uv}(x_{nvk})}{F_{root}^{y_n}(\mathbf{s}_n)}\frac{\partial F_{root}^{y_n}}{\partial F^{uvk}}(\mathbf{s}_n) - \frac{\mathcal{N}_{uv}(x_{nvk})}{F_n^{root}(\mathbf{s}_n)}\frac{\partial F_{root}}{\partial F^{uvk}}(\mathbf{s}_n)\right)\right] + D} \\ &- (\hat{\mu}_{uv})^2\end{aligned} \tag{35}$$

5. Experiments

To evaluate discriminative SPNs using EBW, we carried out four experiments. In the first and second experiment, we compared EBW to generative EM (discriminative EM requires second order derivatives (Salojärvi et al., 2005), which

Table 1: Description of eight datasets used to train feedforward SPNs

Dataset	Var#	Dataset Size	Classes#	Var Type
Banknote (Lohweg et al., 2013a)	4	1371	2	Binary
Voice (Becker, 2016)	20	3167	2	Cont
Credit Card (Dal Pozzolo et al., 2014)	29	284806	2	Cont
Breast Cancer (Michalski et al., 1986)	30	865	2	Cont
Sensorless Drive (Lohweg et al., 2013b)	48	58509	11	Cont
Fault Detection (Huawei)	70	14354	41	Cont
Activity Recognition (Anguita et al., 2012)	561	10299	6	Cont
MNIST (LeCun et al., 1998)	784	70000	10	Binary

are not tractable for SPNs of 1 thousand to 1 million parameters) and discriminative gradient descent on SPNs and RSPNs respectively. The third experiment aims to illustrate the advantage of SPNs over Support Vector Machines (SVMs) and Neural Networks in the case of missing features. In a fourth 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.

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.

5.1. EBW versus Other Parameter Learning Algorithms

In the first and second experiment, we used sixteen different datasets (8 for SPNs and 8 for RSPNs)[2] that span a wide spectrum of domains. The descriptions of the eight datasets used to train feedforward SPNs are in Table 1 [3], and the

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

[3]For banknote datset, we binarized all features by setting values smaller than the mean to 0 and 1 otherwise. This was done to increase the number of binary benchmarks.

Table 2: Test accuracy of EBW, generative EM, and discriminative GD on SPNs with the approximate # of nodes and parameters reported in the second and third column.

Dataset	# nodes	# params	EBW	genEM	discGD
Banknote	163	64	**86.13%**	83.94%	**86.13%**
Voice	2.1k	1.4k	**97.15%**	96.20%	96.20%
Credit Card	4.7k	3.5k	**99.92%**	99.38%	**99.92%**
Breast Cancer	6.1k	4.3k	**96.42%**	92.85%	91.07%
Sensorless Drive	67k	52k	**99.44%**	99.36%	55.41%
Fault Detection	288k	192k	**60.45%**	58.67%	58.12%
Activity Recognition	93k	64k	**90.53%**	88.66%	76.45%
MNIST	1.5M	1M	**95.07%**	93.35%	62.89%

Table 3: Description of eight datasets used to train RSPNs

	length	class #	dimensions #	train size	test size
Japan Vowel	7-29	9	12	270	370
Arabic Digit	4-93	10	13	6600	2200
AUSLAN	45-136	95	22	1140	1425
wafer	104-198	2	6	298	896
libras	45	15	2	180	180
CharacterTrajectories	109-205	20	3	300	2558
uWaveGestureLibrary	315	8	3	200	4278
PEMS	144	7	963	267	173

other eight multivariate and sequential datasets to train RSPNs can be found in Table 3. For the datasets that do not 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 does not 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 2 shows the test accuracy of EBW, discGD and genEM on SPNs, and Table 4 shows the corresponding results for RSPNs. In both tables, EBW always outperforms genEM. We also observed that discGD converges quickly to good

Table 4: The test accuracies of EBW, generative EM, and discriminative GD on RSPNs

Dataset	EBW	genEM	discGD
Japan Vowel (Kudo et al., 1999)	**94.10%**	91.90%	89.19%
Arabic Digit (Hammami and Bedda, 2010)	**93.18%**	92.70 %	**93.18%**
AUSLAN (Kadous, 2002)	**84.21%**	78.95 %	77.19%
wafer (Olszewski, 2001)	**97.20%**	95.53 %	97.09%
libras (Dias et al., 2009)	**97.20%**	94.44 %	95.56%
CharacterTrajectories (Williams et al., 2006)	**62.54 %**	58.64 %	60.59%
uWaveGestureLibrary (Liu et al., 2009)	**72.46 %**	70.14 %	70.24%
PEMS (Cuturi, 2011)	**66.47 %**	64.27 %	64.921 %

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 6 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 did not converge to a solution in 20 epochs while EBW was able to converge after a few epochs.

5.2. 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 with mean imputation.

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. 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

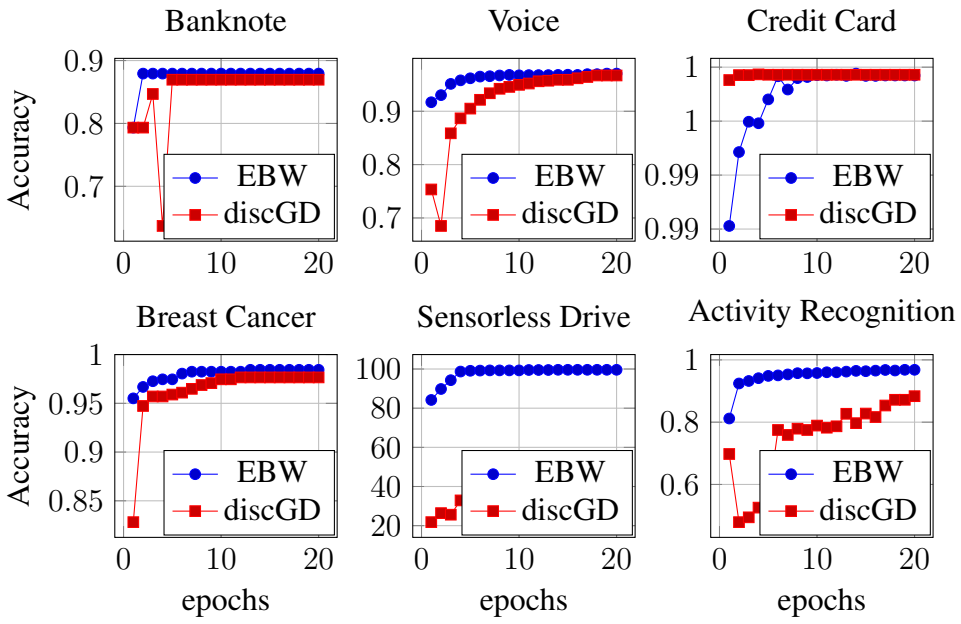


Figure 6: Accuracies on training data versus number of epochs for EBW and discriminative GD algorithms. The time per epoch is the same for both algorithms.

epochs to 20. Table 5 shows that EBW-trained SPNs are consistently more robust to missing features than both SVMs and NNs with mean imputation.

Note that several other techniques much more complicated than mean imputation have been proposed in the literature for neural networks and SVMs. For instance, 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 a more complex optimization problem (Smola et al., 2005). Since neural networks do not have a natural way of handling missing features, a preprocessing technique based on variational auto-encoders (Ivanov et al., 2019) has been proposed to generate imputed values based on observed features that can be fed to any classification technique that assumes complete data. Another possibility is to model the partial inputs with a probability density function based on which the expected value of each missing feature can be fed to any classifier. In the case of neural networks, the first layer must be modified to com-

Table 5: 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%	64.90%	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%

pute the expectation of missing features with respect to the fitted density (Śmieja et al., 2018). We caution the reader that our experiments do not imply that EBW for SPNs is the state of the art to handle missing features. The main contribution of the paper is a new discriminative learning technique for SPNs. We simply want to highlight that EBW for SPNs can handle missing features without any change and that it performs well in comparison to other techniques that do not require any change or complex preprocesssing such as mean imputation. SPNs could be combined with preprocessing techniques if desired.

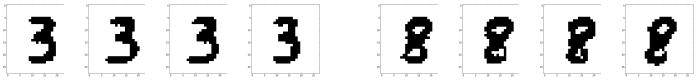
5.3. Visualizing the Parameters Learned by EBW

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

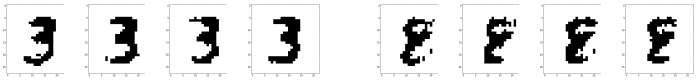
As shown in Figure 7, the sampled images from the generatively trained SPN resemble the appearance of digit '3' and digit '8'. On the other hand, the sampled images from the discriminatively trained SPN show the parts of the digits '3' and '8' that are discriminative. We know that the left part of digit '8' differentiates it from digit '3', which is what was learned by the SPN.

6. Conclusion and Future Work

We described a framework to train SPNs and RSPNs discriminatively using Extended Baum-Welch. We did so by formulating the conditional likelihood as



(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 7: 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'.

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.

In the future, it would be interesting to extend this parameter learning algorithm to classify sequential data where there is one label per time slice (e.g., part-of-speech tagging, behaviour recognition). Also, it would be interesting to derive a consistent structure learning algorithm for discriminative SPNs and RSPNs.

7. References

Adel, T., Balduzzi, D., Ghodsi, A., 2015. Learning the structure of sum-product networks via an SVD-based algorithm, in: Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence.

Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L., 2012. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine, in: International workshop on ambient assisted living, Springer. pp. 216–223.

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

Baum, L.E., Petrie, T., Soules, G., Weiss, N., 1970. A maximization technique

occurring in the statistical analysis of probabilistic functions of Markov chains. The annals of mathematical statistics .

Becker, K., 2016. Gender recognition by voice .

Bilmes, J., 2010. Dynamic graphical models. IEEE Signal Processing Magazine 27, 29–42.

Conaty, D., Maua, D., de Campos, C., 2017. Approximation complexity of maximum a posteriori inference in sum-product networks, in: Proceedings of The 33rd Conference on Uncertainty in Artificial Intelligence, AUAI.

Cuturi, M., 2011. Fast global alignment kernels, in: Proceedings of the 28th international conference on machine learning (ICML-11), pp. 929–936.

Dal Pozzolo, A., Caelen, O., Le Borgne, Y.A., Waterschoot, S., Bontempi, G., 2014. Learned lessons in credit card fraud detection from a practitioner perspective, in: Expert systems with applications.

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

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

Dias, D., Madeo, R., Rocha, T., Biscaro, H., Peres, S., 2009. Hand movement recognition for brazilian sign language: A study using distance-based neural networks, in: International Joint Conference on Neural Networks.

Gens, R., Domingos, P., 2012. Discriminative learning of sum-product networks, in: Advances in Neural Information Processing Systems, pp. 3239–3247.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. `http://www.deeplearningbook.org`.

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

Hammami, N., Bedda, M., 2010. Improved tree model for arabic speech recognition, in: International Conference on Computer Science and Information Technology.

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

Ivanov, O., Figurnov, M., Vetrov, D., 2019. Variational autoencoder with arbitrary conditioning, in: International Conference on Learning Representations.

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

Jebara, T., Pentland, A., 1999. Maximum conditional likelihood via bound maximization and the CEM algorithm, in: Advances in neural information processing systems, pp. 494–500.

Jebara, T., Pentland, A., 2001. On reversing jensen's inequality, in: Advances in Neural Information Processing Systems, pp. 231–237.

Kadous, M., 2002. Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series. Ph.D. thesis. The University of New South Wales.

Kalra, A., Rashwan, A., Hsu, W.S., Poupart, P., Doshi, P., Trimponias, G., 2018. Online structure learning for feed-forward and recurrent sum-product networks, in: Advances in Neural Information Processing Systems, pp. 6944–6954.

Kanevsky, D., 2004. Extended baum transformations for general functions, in: 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE. pp. I–821.

Kudo, M., Toyama, J., Shimbo, M., 1999. Multidimensional curve classification using passing-through regions. Pattern Recognition Letters 20, 1103–1111.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition, in: Proceedings of the IEEE.

Liu, J., Zhong, L., Wickramasuriya, J., Vasudevan, V., 2009. uwave: Accelerometer-based personalized gesture recognition and its applications. Pervasive and Mobile Computing 5, 657–675.

Lohweg, V., Hoffmann, J., D´orksen, H., Hildebrand, R., Gillich, E., Hofmann, J., Schaede, J., 2013a. Banknote authentication with mobile devices .

Lohweg, V., Paschke, F., Bayer, C., Bator, M., Mönks, U., Dicks, A., Enge-Rosenblatt, O., 2013b. Sensorlose zustandsüberwachung an synchronmotoren, in: Workshop Computational Intelligence.

Mei, J., Jiang, Y., Tu, K., 2018. Maximum a posteriori inference in sum-product networks, in: Thirty-Second AAAI Conference on Artificial Intelligence.

Melibari, M., Poupart, P., Doshi, P., Trimponias, G., 2016. Dynamic sum product networks for tractable inference on sequence data, in: Conference on Probabilistic Graphical Models, pp. 345–355.

Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N., 1986. The multi-purpose incremental learning system aq15 and its testing application to three medical domains., in: The Fifth National Conference on Artificial Intelligence.

Molina, A., Natarajan, S., Kersting, K., 2017. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions, in: Thirty-First AAAI Conference on Artificial Intelligence.

Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., Kersting, K., 2018. Mixed sum-product networks: A deep architecture for hybrid domains, in: The Thirty-Second AAAI Conference on Artificial Intelligence.

Murphy, K., 2002. Dynamic Bayesian Networks: Representation, Inference and Learning. Ph.D. thesis. University of California, Berkeley.

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

Normandin, Y., Morgera, S.D., 1991. An improved mmie training algorithm for speaker-independent, small vocabulary, continuous speech recognition, in: [Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing, IEEE. pp. 537–540.

Olszewski, R., 2001. Generalized Feature Extraction for Structural Pattern Recognition in Time-Series Data. Ph.D. thesis. Carnegie Mellon University.

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

Pelckmans, K., De Brabanter, J., Suykens, J.A., De Moor, B., 2005. Handling missing values in support vector machine classifiers. Neural Networks 18, 684–692.

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

Poon, H., Domingos, P., 2011. Sum-product networks: a new deep architecture, in: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, AUAI Press. pp. 337–346.

Rashwan, A., Zhao, H., Poupart, P., 2016. Online and distributed bayesian moment matching for parameter learning in sum-product networks, in: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, pp. 1469–1477.

Rooshenas, A., Lowd, D., 2014. Learning sum-product networks with direct and indirect variable interactions, in: International Conference on Machine Learning, pp. 710–718.

Salojärvi, J., Puolamäki, K., Kaski, S., 2005. Expectation maximization algorithms for conditional likelihoods, in: Proceedings of the 22nd international conference on Machine learning, ACM. pp. 752–759.

Śmieja, M., Struski, Ł., Tabor, J., Zieliński, B., Spurek, P., 2018. Processing of missing data by neural networks, in: Advances in Neural Information Processing Systems, pp. 2719–2729.

Smola, A.J., Vishwanathan, S., Hofmann, T., 2005. Kernel methods for missing variables., in: the Tenth Workshop on Artificial Intelligence and Statistics.

Williams, B.H., Toussaint, M., Storkey, A.J., 2006. Extracting motion primitives from natural handwriting data, in: International Conference on Artificial Neural Networks, Springer. pp. 634–643.

Zhao, H., Adel, T., Gordon, G., Amos, B., 2016a. Collapsed variational inference for sum-product networks, in: International Conference on Machine Learning, pp. 1310–1318.

Zhao, H., Melibari, M., Poupart, P., 2015. On the relationship between sum-product networks and Bayesian networks, in: International Conference on Machine Learning, pp. 116–124.

Zhao, H., Poupart, P., Gordon, G.J., 2016b. A unified approach for learning the parameters of sum-product networks, in: Advances in neural information processing systems, pp. 433–441.