Learning the Structure of Sum-Product Networks via an SVD-based Algorithm

Tameem Adel
Radboud University

David Balduzzi
Victoria University of Wellington

Ali Ghodsi
University of Waterloo

Abstract

Sum-product networks (SPNs) are a recently developed class of deep probabilistic models where inference is tractable. We present two new structure learning algorithms for sum-product networks, in the generative and discriminative settings, that are based on recursively extracting rank-one submatrices from data. The proposed algorithms find the subSPNs that are the most coherent *jointly* in the instances and variables – that is, whose instances are most strongly correlated over the given variables.

Experimental results show that SPNs learned using the proposed generative algorithm have better likelihood and inference results – and also much faster – than previous approaches. Finally, we apply the discriminative SPN structure learning algorithm to handwritten digit recognition tasks, where it achieves state-of-the-art performance for an SPN.

1 INTRODUCTION

Sum-product networks (SPNs), introduced in Poon and Domingos [2011], provide compact, tractable representations of probability distributions. Layers of hidden variables are added to the model so long as they maintain compactness whilst keeping inference tractable.

In this work, we present a new SPN structure learning algorithm that constructs an SPN by identifying coherent subSPNs that are sought concurrently across both the instance and variable dimensions. The subSPN search procedure aims at compactly and tractably representing the data and is capable of splitting the data matrix across both dimensions at once, if this leads to a better data representation.

Contribution. We make two main contributions. The first is `SPN-SVD`, a new SPN structure learning algorithm based on rank-one (rank-1) submatrix extraction. The problem of finding subSPNs is reformulated as a problem of finding approximate rank-1 submatrices of the data matrix. An important feature of the approach is that it splits the data along two dimensions (variables and instances) simultaneously when doing so optimises the objective. In contrast, previously developed approaches to learning the structure of SPNs split the data across one dimension only, at a time, without taking into consideration that such local improvement might drift the overall resulting SPN away from the optimal representation.

The second main contribution is an extension of our structure learning algorithm to the setting of *discriminative* learning [Gens and Domingos, 2012]. The discriminative structure learning algorithm, `DSPN-SVD`, first extracts the features that are the most dependent on the labels, where dependence is measured via the Hilbert-Schmidt Independence Criterion (HSIC), and then recursively applies `SPN-SVD`. To the best of our knowledge, it is the first structure learning algorithm designed for discriminatively training SPNs.

The performance of both algorithms is extensively evaluated. When evaluated on the Caltech-101 and Olivetti image datasets, `SPN-SVD` outperforms other SPN algorithms, with higher log-likelihood (LL) values and much faster performance. The discriminative structure learning algorithm achieves state-of-the-art performance on handwritten digit classification when compared with other SPN algorithms.

2 SUM-PRODUCT NETWORKS

SPNs are built by composing tractable distributions. A tractable distribution is a distribution whose partition function and mode can be computed in time $O(1)$ [Gens and Domingos, 2013]. A tractable univariate distribution, D_X, is an SPN. SPNs provide a representation in which single tractable distributions of the form D_X are combined into a richer and more complex distribution, provided that the resulting distribution remains tractable.

An SPN is a rooted directed acyclic graph (DAG), Gr, whose leaves are univariate distributions, and whose internal nodes are sum and product nodes. Edges from a sum node to its children are assigned positive weights, W. Let $Br(S)$ denote the branches (children) of S. The scope, sc, of a sum-product network, $S(Gr, W)$, is the set of variables that appear in the leaf nodes of the SPN [Poon and Domingos, 2011]. Sum nodes are denoted by $Sum_i(Br_1 : w_1, Br_2 : w_2, \ldots)$, where $Br_1, Br_2, \ldots$ are the branches and w_1 and w_2 are their respective weights. Similarly, product nodes are denoted by $Prd_i(Br_1, Br_2, \ldots)$.

The two composition rules used in SPNs are defined as follows. Firstly, a product, Prd, of SPNs, $sub_1, \ldots, sub_k$, over disjoint scopes, is an SPN, rooted by Prd:

$$\forall sub_i, sub_j \in Br(Prd) : sc(sub_i) \cap sc(sub_j) = \phi \quad (1)$$

A decomposable SPN is one in which each product node satisfies Eq. (1).

Secondly, a positive weighted sum, Sum, of SPNs over the same scope is an SPN rooted by Sum [Gens and Domingos, 2013]:

$$\forall sub_i, sub_j \in Br(Sum) : sc(sub_i) = sc(sub_j) \quad (2)$$

A complete SPN is one in which each sum node satisfies Eq. (2). A sum node, Sum, can be thought of as the result of summing out a hidden variable. The sum of weights of all the branches of a sum node is always equal to 1 ($\sum_{sub \in Br(Sum)} w_{sub} = 1$).

2.1 Related Work on Structure Learning

The emphasis in the SPN literature has recently shifted from learning parameters to learning the structure of models. Parameter learning algorithms assume a fixed structure and learn weights using either generative [Poon and Domingos, 2011] or discriminative [Gens and Domingos, 2012] training. A hard EM algorithm is used by Poon and Domingos [2011] to perform generative parameter learning on SPNs. Deep SPNs are learned successfully using hard EM with a *pre-defined* network structure. A discriminative parameter learning algorithm based on gradient descent was introduced in Gens and Domingos [2012].

The first algorithm to learn the structure of an SPN from data was proposed by Dennis and Ventura [2012]. The algorithm first clusters data instances and creates a corresponding sum node. Then, it clusters variables in a top-down approach, creating product nodes. There are three potential problems with the algorithm. The first is that any context-specific independences that appear after the first clustering are not taken into consideration because instances are not clustered after the first step. Secondly, the algorithm is based on a clustering method that ignores correlation between variables. It will therefore tend to place dissimilar, but strongly correlated, variables in different clusters. Finally, the structure and weights are learned using two distinct methods.

A bottom-up approach, based on greedily merging small image regions into larger regions, was introduced in Peharz et al. [2013]. An online learning algorithm was proposed by Lee et al. [2013], where the problem was cast as an online clustering problem. They develop an incremental SPN structure learning algorithm based on dynamically modifying the number of clusters based on incoming data.

The most prominent general SPN structure learning algorithm was proposed by Gens and Domingos [2013]. It applies a recursive top-down approach which, at each step, checks whether variables can be split into approximately independent subsets – in which case a product node is constructed. Otherwise, the current instances are clustered, and a sum node is returned with weights proportional to the number of instances in each cluster. The algorithm greedily optimises the log-likelihood and overcomes several limitations in Dennis and Ventura [2012]. However, it only searches locally for the ideal splitting candidate at each step, see discussion of Table 1 below.

To the best of our knowledge, the most recent algorithm for general SPN structure learning was proposed in Rooshenas and Lowd [2014]. The authors adapt a method based on mixture modelling and arithmetic circuit learning. It does not only apply local modifications in the search of an optimal model as arithmetic circuit learning could lead to global changes. However, a global search over the data is neither systemic nor guaranteed.

Peharz et al. [2014] learn the structure of a restricted class of SPNs, where each sum node can have no more than one branch with a non-zero output for a certain input. Nath and Domingos [2014] develop an algorithm that learns the structure of relational SPNs.

Our focus is on learning the structure of general SPNs in both the generative and discriminative settings.

3 LEARNING THE STRUCTURE OF AN SPN

Previous SPN structure learning algorithms cluster instances without ensuring that the clustering respects context-specific independences (independences that hold only among instances of a specific context or cluster). In contrast, our proposed algorithm (`SPN-SVD`) concurrently checks the data matrix across both the instance and variable dimensions, looking for coherent subSPNs in the form of rank-1 submatrices.

Motivating Example. It is useful to consider a simple example in detail, so as to understand how `SPN-SVD` differs from `SPN-Gens`. Tables 1 & 2 contrast the steps

taken by SPN−SVD and SPN−Gens [Gens and Domingos, 2013]. Table 1(A) shows the data matrix, which consists of 6 instances with 4 variables each. SPN−Gens clusters the data into 2 clusters, as shown in Table 1(B) and in Figure 1. In the example, the cluster of elements with value 18 is split.

Table 1: SPN−Gens on an Example Data Matrix. Rows are instances and columns are variables.

40	40	40	40
40	40	40	40
40	18	18	40
20	18	18	20
20	20	20	20
20	20	20	20

A

40	40	40	40
40	40	40	40
40	18	18	40
20	18	18	20
20	20	20	20
20	20	20	20

B

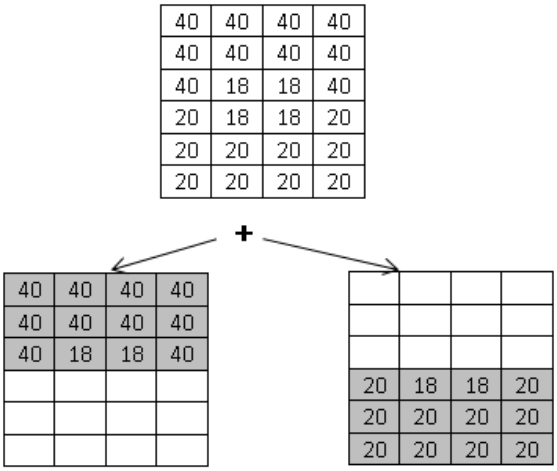

Figure 1: The SPN Structure Learned from Table 1 by SPN−Gens.

SPN−SVD deals with the same data quite differently, as shown in Table 2. The algorithm simultaneously searches over instances and variables and therefore immediately identifies the submatrix with all entries equal to 18, chooses it as a rank-1 submatrix. The algorithm then decomposes the original matrix into three components and acts recursively on each.

The crucial difference between the two algorithms is that SPN−SVD identifies the submatrix of entries with value 18 as an "atom" in the data, resulting in a graph structure that captures an important feature of the data, whereas SPN−Gens does not. Quantitatively, the final LL value for SPN−Gens is -2 whereas for SPN−SVD it is -1.39.

Table 2: SPN−SVD Applied to the Matrix from Table 1.

40	40	40	40
40	40	40	40
40	18	18	40
20	18	18	20
20	20	20	20
20	20	20	20

A

40	40	40	40
40	40	40	40
40	18	18	40
20	18	18	20
20	20	20	20
20	20	20	20

B

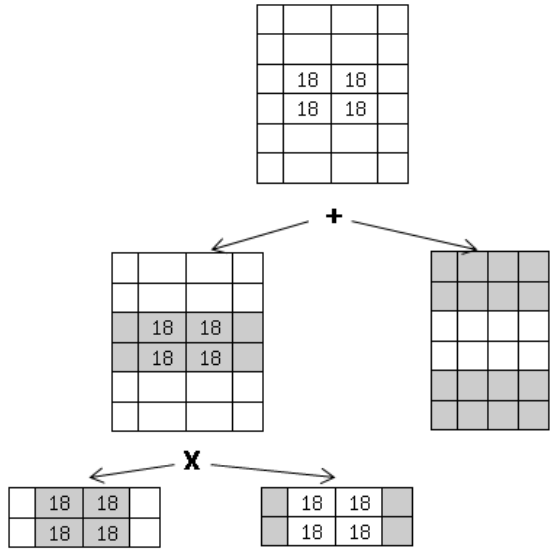

Figure 2: The SPN Structure Learned from Table 1 by SPN−SVD.

In essence, SPN−SVD and SPN−Gens are motivated by two different extreme cases.

SPN−Gens is inspired by the observation that if variables are independent, then they can be decomposed into separate (branches or) leaves of an SPN product node.

In contrast, SPN−SVD is inspired by a complementary observation: if a subset of variables is perfectly correlated over a subset of data, then it forms a rank-1 submatrix. These rank-1 submatrices are the "atoms" out of which SPN−SVD builds an SPN. Whereas SPN−Gens searches for independencies; SPN−SVD searches for correlated components.

Searching for rank-1 submatrices instead of independent variables has three potential advantages. Firstly, correlations are easier to estimate than independence. Secondly, the search for rank-1 submatrices occurs jointly over variables and instances, whereas clustering and identifying independencies are two unrelated procedures. Thirdly, extracting correlated submatrices reduces redundant compu-

tations, resulting in a faster algorithm, see Lemma 1 below.

3.1 Extracting a Rank-1 Submatrix

The main subroutine of SPN−SVD is a rank-1 extraction algorithm based on singular value decomposition (SVD). The approach derives from an algorithm for nonnegative matrix factorization (NMF) developed in Biggs et al. [2008a].

Let $X \in \mathbb{R}^{m \times n}$ denote a data matrix containing m instances and n variables. We denote by $X_{i\bullet}$ the i^{th} row of the matrix, corresponding to the i^{th} instance of the data, and by $X_{\bullet j}$ the j^{th} column of the matrix, corresponding to the j^{th} variable.

In the generative training case, assume the labels, if provided, are included in X. We introduce the notation $X_{(M,N)}$ to refer to the *submatrix* of X consisting of rows $M \subset \{1, \ldots, m\}$ and columns $N \subset \{1, \ldots, n\}$.

The algorithm extracts the submatrix of the data matrix X that is closest to having rank-1, denoted by B_1, by maximising

$$B_1 := \underset{X_{(M,N)}}{\operatorname{argmax}} \|X_{(M,N)}\|_F^2 - \gamma \|X_{(M,N)} - \sigma u v^\intercal\|_F^2, \quad (3)$$

where σ is the maximum singular value of the submatrix $X_{(M,N)}$ and $u \in \mathbb{R}^m, v \in \mathbb{R}^n$ are the dominant singular vectors of of $X_{(M,N)}$, and $\| \bullet \|_F$ is the Frobenius norm. Recall that the Frobenius norm is the root sum of the squared singular values.

The second term in Eq. (3) thus encourages the optimization to find a submatrix that is close to rank-1. The first term ensures the submatrix is biased towards having large singular values; γ controls the penalty incurred as $X_{(M,N)}$ deviates from being rank-1.

Algorithm 1 details the subroutine, extractR1, used to extract approximate rank-1 submatrices. For fixed M and N, extractR1 exactly coincides with the SVD power method. However, instead of fixing M and N, the inner loop searches for the submatrix with the closest rank-1 approximation. Lines 6 and 8 show a heuristic used to solve the NP-hard problem defined in Eq. (3) [Biggs et al., 2008a]. The criterion in lines 6 and 8 of Algorithm 1 decides whether or not to include one column or row separately. This makes the subroutine parallelisable and highly scalable in terms of memory and processing power. extractR1 computes the *dominant* singular vectors of a *submatrix*, which are less prone to perturbations by noise than the full matrix [Biggs et al., 2008b].

3.2 Generative SPN Structure Learning Algorithm (SPN−SVD)

SPN−SVD recursively extracts "atomic" submatrices from the input matrix. Each extraction breaks the input matrix into three pieces, which are glued together as sum and product nodes.

Algorithm 1 Function extractR1(X)

Input: $X \in \mathbb{R}^{m \times n}, \gamma > 1$
Output: [M, N, $Stop$]
1: Select $j_0 \in \{1, \ldots, n\}$ to maximise $\|X_{(:,j_0)}\|_F$
2: $M = \{1, 2, \ldots, m\}$, $N = \{j_0\}$
3: $u = X_{(:,j_0)}$
4: **repeat**
5: $\quad v = X_{(M,:)^\intercal} \cdot \frac{u(M)}{\|u(M)\|_F}$
6: $\quad N = \{j : \gamma v(j)^2 - \|X_{(M,j)}\|_F{}^2 > 0\}$
7: $\quad u = X_{(:,N)} \cdot \frac{v(N)}{\|v(N)\|_F}$
8: $\quad M = \{i : \gamma u(i)^2 - \|X_{(i,N)}\|_F{}^2 > 0\}$
9: **until** M, N, u, v do not change
10: **if** $X_{(M,N)} = X$ or ($|M| = 0$ and $|N| = 0$) **then**
11: $\quad Stop = true$
12: **else**
13: $\quad Stop = false$
14: **end if**

Algorithm 2 details the main steps of SPN−SVD. Rows are instances and columns are variables. Each variable is divided by its standard deviation before extracting rank-1 submatrices. The normalised values are used in the subroutine extractR1 only, and are not returned or updated in the matrix. Normalisation helps discover correlated sets of variables.

Given an input matrix, subroutine extractR1 recursively extracts an approximate rank-1 submatrix B_1. The matrix is then split into three components: the submatrix B_1, submatrix B_2 consisting of other variables on the same instances as B_1, and finally submatrix B_3 consisting of the remaining instances.

The optimization in (3) ensures that variables in B_1 are maximally correlated and the remaining variables, captured by B_2, are largely uncorrelated with B_1. The algorithm therefore combines B_1 and B_2 via a product node

$$Prd(B_1, B_2).$$

Finally, the remaining *instances*, captured by B_3, are added via a sum node

$$Sum\big(B_3 : w_1, Prd(B_1, B_2) : w_2\big),$$

where $w_2 = \frac{|M|}{m}$ and $w_1 = 1 - w_2$.

The algorithm proceeds recursively by feeding B_1, B_2 and B_3 back into the algorithm as input matrices until one of three base cases is reached:

1. *The input matrix contains a single variable:*

   The remaining vector represents a univariate distribution and a leaf node is created (line 2).

Algorithm 2 Function SPN−SVD(A)

Input: $A \in \mathbb{R}^{m \times n}, \gamma > 1$
Output: Sum-product network S representing A
1: **if** #columns(A) = 1 **then**
2: **return** univariate distribution on variable.
3: **else if** #rows(A) = 1 **then**
4: **return** return Prd(variables in A).
5: **end if**
6: [M, N, $Stop$] = extractR1(A, γ)
7: Set $B_1 = A_{(M,N)}$, $B_2 = A_{(M,N^c)}$ and $B_3 = A_{(M^c,\{1\ldots n\})}$
8: **if** $Stop$ = true **then**
9: **return** multivariate distribution MVLN(A)
10: **else**
11: Construct $Prd(B_1, B_2)$
12: Call SPN−SVD(B_1) and SPN−SVD(B_2)
13: Construct

$$Sum\left(B_3 : \frac{m - |M|}{m}, Prd(B_1, B_2) : \frac{|M|}{m}\right)$$

14: Call SPN−SVD(B_3)
15: **end if**

2. *The input matrix contains a single instance:*

   All variables are independent and a product node is created (line 4). Its leaves are the relevant variables.

3. *The entire input matrix is extracted*:

   The variables in A are highly correlated since A has rank-1. The algorithm therefore constructs a sum node with a branch per instance in A, followed by product nodes over the variables. We refer to the node as a *multivariate leaf node* or $MVLN$.

   More precisely, if $A \in \mathbb{R}^{M \times N}$ and $B_1 \equiv A$ then for each instance j form product node

$$r_j := Prd_j(A_{j1}, \ldots, A_{jN}).$$

   Combine the product nodes by summing over instances:

$$\text{MVLN}(A) := Sum(r_1 : w, \ldots, r_M : w), \quad (4)$$

   where $w = \frac{1}{|M|}$.

Variables in a rank-1 submatrix cannot be independent. SPN−SVD achieves significant speedups by avoiding redundant searches for independencies and returning MVLNs. In contrast, when SPN−Gens encounters a rank-1 submatrix, it recursively searches for independences and clusters across its subsets, which leads to a slower implementation and also an SPN with a more complicated structure. The speedup from using MVLNs is reported in the experimental results below.

A commonly used assumption is that data is clustered in strongly correlated groups. For example, algorithms such as the group Lasso seeks solutions where groups of variables are zero together [Bach, 2008]. The following simple Lemma illustrates how large MVLNs arise in the more general setting where groups of variables receive the same, or even approximately the same, values.

Lemma 1. *Let $X \in \mathbb{R}^{m \times n}$ be a data matrix consisting of m instances. Suppose that X contains a group of instances $G \subset \{1, \ldots, m\}$ with similarity pattern $\mathbf{K}_G = \{k | X_{ik} = X_{jk}$ for all $i, j \in G\}$. Then SPN−SVD will find a multivariate leaf node satisfying*

$$|MVLN| \geq |\mathbf{K}_G| \cdot |G|.$$

Proof. The result follows immediately since the algorithm will either find the MVLN corresponding to the group, or a larger MVLN. □

An interesting question, deferred to future work, is to characterise the *collections* of groups with similarity patterns that are best suited to the SPN−SVD algorithm. It is also worth investigating how robustly MVLNs are extracted in the presence of noise.

3.3 Discriminative SPN Structure Learning Algorithm (DSPN−SVD)

Finally, we consider the setting of discriminative learning, where the algorithm is provided with labeled data. Discriminative learning models the conditional distribution $P(Y|X)$, rather than the joint distribution $P(X, Y)$. Discriminative SPNs combine the flexibility of selecting/extracting relevant features, with the tractability and representational prowess of SPNs. They can achieve high classification or regression accuracy by selecting variables that are dependent on Y. They were first introduced by Gens and Domingos [2012], where parameters were learned on a pre-defined structure.

We propose a new discriminative SPN structure learning algorithm, referred to as DSPN−SVD. We assume the labels are discrete, and belong to the set $\mathcal{C} = \{1, \ldots, l\}$. The algorithm extracts features Z from the input matrix, X, that are maximally correlated (in a suitable sense) with the labels Y. The algorithm then applies SPN−SVD to the learned features to construct a collection of generative SPNs, one per conditional distribution $P(Z|Y = j)$ for $j \in \mathcal{C}$, that are combined by a single sum-node.

Extracting Z requires a measure of dependence between variables. We use the Hilbert-Schmidt independence criterion (HSIC), which we briefly recall [Gretton et al., 2005].

Let $k(x, x')$ and $l(y, y')$ be kernels on the input space $\mathcal{X}$ and the label space $\mathcal{Y}$, with corresponding feature maps ϕ :

$\mathcal{X} \to \mathcal{F}$ and $\psi : \mathcal{Y} \to \mathcal{G}$ respectively. The Hilbert-Schmidt Independence Criterion is

$$HSIC(k, l, P_{XY}) := \|C_{xy}\|_F, \text{ where}$$
$$C_{xy} := \mathop{\mathbf{E}}_{(x,y)\sim P}[(\phi(x) - \mu_x) \otimes (\psi(y) - \mu_y)]$$

is the cross-covariance operator [Fukumizu et al., 2004]. We apply the HSIC for supervised feature selection following Song et al. [2007].

Let $\Pi := \{W \in \mathbb{R}^{n\times d} : \langle W_{\bullet i}, W_{\bullet j}\rangle = \delta_{ij}\}$ denote the set of orthogonal projections from $\mathbb{R}^n = \mathcal{X}$ to $\mathbb{R}^d$. Given the standard dot product $\langle \bullet, \bullet\rangle$ on $\mathbb{R}^d$, each projection induces a kernel $K_W(x, y) := \langle Wx, Wy\rangle$ on $\mathcal{X}$.

Let $L(y, y) = \delta_{y,y'}$ be the Kronecker kernel, which is 1 if $y = y'$ and 0 otherwise. The Kronecker kernel is suitable for the categorical variable, $Y \in \mathcal{C}$, because it expresses precisely whether or not two labels are equal. It is easy to extend to real-valued or structured labels by employing more sophisticated kernels.

Let $X \in \mathbb{R}^{m\times n}$ be a data matrix with labels $Y \in \mathcal{Y}^m$, yielding empirical distribution $\hat{P}_{XY}$. Construct the centering matrix $H = (Id_m - m^{-1}11^\intercal)$ and empirical Kronecker kernel $L_{ij} = \delta_{y_i=y_j}$.

Lemma 2. *Let $V \in \mathbb{R}^{n\times d}$ be the top d eigenvectors of*

$$\chi := X^\intercal HLHX \tag{5}$$

Then V maximizes the Hilbert-Schmidt dependence:

$$V = \mathop{\text{argmax}}_{W\in\Pi} HSIC(K_W, L, \hat{P}_{XY}).$$

Proof. The vectors $V_{\bullet j}$, $j = 1, \ldots, d$, are the eigenvectors of χ. They therefore maximize the trace

$$\mathop{\text{argmax}}_{V}\ tr(V^\intercal X^\intercal HLHXV)$$

Since $tr(AB) = tr(BA)$, the objective can be rewritten as:

$$\mathop{\text{argmax}}_{V}\ tr(HXVV^\intercal X^\intercal HL) \tag{6}$$

Following Barshan et al. [2011], let $K = XVV^\intercal X^\intercal$. The objective in Eq. (6) is then

$$tr(HKHL),$$

which is the HSIC [Gretton et al., 2005]. □

The higher the HSIC value, the stronger the dependence between the projected representation of the data $Z = XV$ and Y, and thus the more useful the representation is for discriminative learning.

Algorithm 3 Function `DSPN-SVD`

Input: $X \in \mathbb{R}^{m\times n}, Y \in \mathcal{Y}^m, \gamma > 1, d > 1$
Output: Sum-product network S representing $Y|X$

1: Construct kernel matrix $L_{ij} = \left(\delta_{y_i=y_j}\right)_{i,j=1}^{m}$ and centering matrix $H = (Id_m - m^{-1}11^\intercal)$.
2: Compute the d eigenvectors $V \in \mathbb{R}^{n\times d}$ of $\chi = X^\intercal HLHX$ with the largest eigenvalues.
3: Set feature matrix $Z_{m\times d} \leftarrow X_{m\times n} \cdot V_{n\times d}$
4: Construct sum node $Sum_1(Br_j : w_j)$, where Br_j contains all instances with label j, and weight $w_j = \frac{\#\text{instances labeled } j}{m}$.
5: **for** label j in $\mathcal{Y}$ **do**
6: `SPN-SVD`(Br_j, γ)
7: **end for**

After extracting the features Z, there are two remaining steps. The first step constructs the base node for the discriminative SPN as a sum node that separates instances belonging to different labels. Nodes in the sum are weighted by the number of instances. The resulting network is thus automatically biased towards more common labels. The second step applies the generative `SPN-SVD` algorithm to each branch of the sum node using the extracted features in Z.

The main steps of `DSPN-SVD` are shown in Algorithm 3.

Figure 3 shows an example with 2 labels and 3 variables. As shown in the tables at the top of the figure, the X variables are replaced by a more suitable representation, Z. The base sum node then places instances of each label on separate branches, where each branch's subSPN is in turn learned by `SPN-SVD`. A simplified example of the feature extraction process is shown in Figure 4. Since the first two features convey no information about the labels, extracting only the third feature, $Z = X_{\bullet 3}$, maximises the HSIC.

4 EXPERIMENTS

4.1 Generatively Trained SPNs

Our main evaluation of `SPN-SVD` is based on comparing its accuracy and speed to other SPN structure learning algorithms. Following prior work on structure learning [Gens and Domingos, 2013, Rooshenas and Lowd, 2014], we report accuracy in terms of the test-set log-likelihood (LL) and query conditional log-likelihood (CLL). These values are obtained from experiments on the Caltech-101 dataset [Fei-Fei et al., 2004], the Olivetti face dataset [Salakhutdinov and Hinton, 2009], and 20 binary datasets. Caltech-101 is one of the most commonly used image datasets. It contains images divided into 101 categories, e.g. airplanes, cameras and faces. Each object category contains from 40 to 800 images. Images in Caltech-101 are 64×64 pixels. The Olivetti dataset contains 400 face images of 64×64

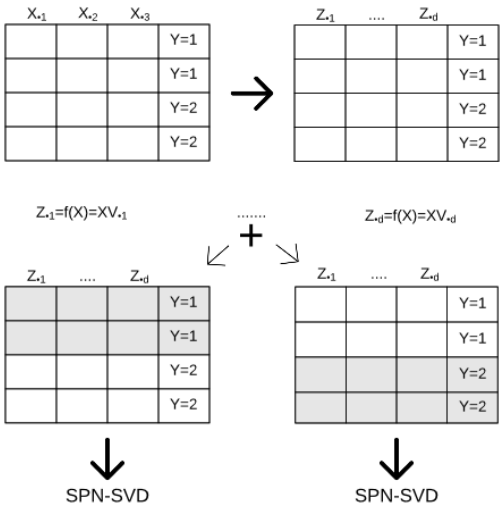

Figure 3: The Discriminative SPN Prior to Running SPN-SVD on Each Sum Node Branch.

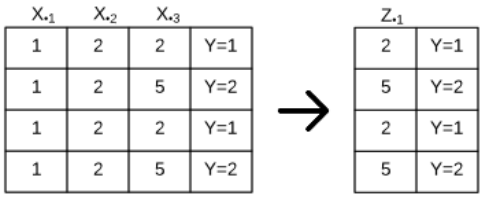

Figure 4: An Example of X, Y and Z where $m = 4, n = 3, d = 1$. $Z = XV = X[0\ 0\ 1]^\intercal$.

pixels. Importantly, these datasets are not binary, in contrast to previous datasets used for SPN structure learning. The datasets are discrete-valued. Extending to the continuous case is straightforward. 60% of the instances of each object category are used for training, 10% for validation (needed mainly for `WinMine`) and 30% for testing.

Accuracy. For Caltech-101, values of LL and inference are displayed as average values across all object categories, as well as averages for some of the individual object categories, while only the grand average is displayed for Olivetti. By "average", we mean that the LL values displayed represent their respective summation of LL values divided by the number of test instances. Univariate leaf distributions are multinomials with Laplace smoothing (add 0.1).

We compared `SPN-SVD` with four algorithms: (1) `SPN-Gens` [Gens and Domingos, 2013], with code available online; (2) `ID-SPN` [Rooshenas and Lowd, 2014] by the Libra toolkit; (3) `SPN-Dennis` [Dennis and Ventura, 2012], which was implemented via algorithms 1, 2 & 3 of Dennis and Ventura [2012]; (4) Bayesian network structure learning with the `WinMine` toolkit [Chickering, 2002], which was chosen because it can express context-specific independence.

Table 3 shows the test-set LL values obtained for 18 example object categories from Caltech-101, the whole Caltech-101 dataset and the Olivetti face dataset using the `SPN-SVD`, `SPN-Gens`, `ID-SPN`, `SPN-Dennis` algorithms and `WinMine`. The greater the LL, the better. The total number of instances (training + test + validation) in each category or dataset is shown in Table 3. Bold red signifies that an algorithm is significantly better than competitors on a category, whereas bold black indicates that an algorithm is better than competitors on a category. Significant results are identified using a paired t-test (performed in the log scale) with $p = 0.05$. Out of the 101 Caltech-101 categories, `SPN-Gens` is significantly better than its competitors in 5 categories, `WinMine` in 9 categories, `ID-SPN` in 12 categories while `SPN-SVD` is significantly better in 42 categories, 9 of which are shown in Table 3.

Training time. An important advantage of `SPN-SVD` is its rapid training time. `SPN-SVD` took 2.5 hours with 1 CPU to build the SPN and calculate the test-set LL values for Caltech-101 and Olivetti. In contrast, `SPN-Gens` took 13.5 hours, `ID-SPN` took 12 hours, and `SPN-Dennis` took 7.5 hours to perform the same task. `WinMine` took 2.5 hours to build the Bayesian network and calculate LL values.

Per the discussion of Lemma 1, we expect that larger $MVLN$s lead to larger reductions in run-time. Our experiments show a 3-fold speedup when comparing `SPN-SVD`'s performance with and without $MVLNs$. Returning MVLNs thus accounts for most of the 4.5-fold speedup of `SPN-SVD` compared to `SPN-Gens`.

Another major advantage of `SPN-SVD` is that is has few tuning parameters. The generative algorithm has one parameter, γ, which controls the penalty for deviating from rank-1, whereas `DSPN-SVD` has a 2^{nd} parameter: d, the number of extracted features. In comparison, `ID-SPN`, for example, has: L_1 prior parameters C^{ji}, split penalty SP^{ji}, maximum edges ME^{ji} for each AC node, cluster penalty, standard deviation of the Gaussian priors, and the number of main iterations [Rooshenas and Lowd, 2014].

Queries. Next, we investigate the accuracy and speed of queries. Queries are generated following Gens and Domingos [2013]. Experiments are performed with a range of query and evidence variables, see Table 4. A number of instances are selected randomly from the test-set of each object category or dataset, and then queries $P(Q = q|E = e)$ are created by randomly picking proportions of the variables. The average CLL $\log P(Q = q|E = e)$ is com-

Table 3: Test-set LL and Learning Time. Results are shown for 18 Caltech-101 Categories, Caltech-101 & Olivetti. Bold red signifies that an algorithm significantly outperforms the rest.

Dataset	# inst.	SPN-SVD	SPN-Gens	SPN-Dennis	ID-SPN	WinMine
Faces	435	**-1122.71**	-1520.03	-1607.8	-1440.84	-1309.37
Faces-Easy	435	**-1002.11**	-1298.59	-1490.21	-1314.09	-1320.87
Accordion	55	**-974.93**	-1114.05	-1507.79	-1300	-1240
Airplanes	800	**-587.4**	-920.69	-1000.3	-898.7	-914.81
Anchor	42	-1315.71	-1420.1	-1392.28	-1404.12	**-1239.8**
Ant	42	**-770.2**	-1535.82	-1980.3	-1264.1	-1271.94
Background-Google	467	**-1105.49**	-1316.8	-2020.88	-1291.16	-1220
Barrel	47	**-774.23**	-1330.4	-1289.4	-1259.7	-1300.86
Bass	54	**-1051.7**	-1293.11	-1712.84	-1321.49	-1212.37
Beaver	46	-1167.33	-1570.26	-1487.79	-1290.1	**-1012.03**
Binocular	33	**-907.48**	-1390.3	-1600.3	-1400.44	-1309.4
Bonsai	128	**-887.42**	-1551.09	-1979.26	-1302.37	-1336.28
Brain	98	-1270.1	**-1208.41**	-1498	-1307.12	-1286.2
Brontosaurus	43	**-837.02**	-1288.13	-1600.26	-1393.9	-1410.61
Buddha	85	-1291.15	-1374.12	-1230.8	**-1172.28**	-1219
Butterfly	91	**-1020.67**	-1397.19	-1535.91	-1230.11	-1207.44
Camera	50	-1201.8	-1470.25	-1488.85	**-1019.51**	-1200.49
Cannon	43	**-956.47**	-1303.1	-1404.71	-1307.8	-1288.1
Caltech-101 (All)	9144	**-892.93**	-1492.12	-1780.5	-1250.6	-1269.29
Olivetti	400	**-189.81**	-294.36	-302.55	-295.81	-293.9
Learning Time		2.5 hours	13.5 hours	7.5 hours	12 hours	2.25 hours

puted and normalised by the number of query variables following Gens and Domingos [2013]. Table 4 shows the results, for varied proportions of evidence and query variables, in the form of the average CLL for both SPN-SVD and SPN-Gens.

Both SPN-SVD and SPN-GENS achieve average dataset CLL values that are significantly higher than the results obtained by SPN-Dennis, ID-SPN and the conditional marginal likelihood (CMLL) values of WinMine. The latter three results are therefore not reported to save space. Similarly, we only show queries of 5 object categories, rather than 18, along with the average CLL of the whole Caltech-101 and Olivetti datasets.

Across all proportions of object categories, there are 84 categories in which SPN-SVD significantly outperforms SPN-Gens, and 18 where the converse occurs. As inference is linear in the number of edges of an SPN, there is not a major difference between average query time for SPN-SVD and SPN-Gens.

Image completion. To confirm that the LL values are visually meaningful, an image completion task was applied to a select few images from Caltech-101. Two images are shown in Figure 5, taken from one of the face categories of Caltech-101, referred to as Faces-easy. The left half of each test image is inferred after building an SPN using training images from the same faces category. In each case, the right half of the test image is given as evidence and the left half is regarded as query variables, and inference is performed by the SPN. The top part of Figure 5 displays the original images and the bottom shows the images inferred by SPN-SVD.

Binary datasets. In Table 5, we report the test-set LL values of SPN-SVD, SPN-Gens and ID-SPN (LL values of SPN-Dennis are significantly lower) on 20 binary datasets used in Gens and Domingos [2013], Rooshenas and Lowd [2014]. The number of instances in a binary dataset ranges from 2k to 388k, and the number of variables ranges from 16 to 1556 [Gens and Domingos, 2013]. Out of the 20 datasets, SPN-SVD outperforms the alternatives in 7 datasets, whereas ID-SPN outperforms the rest in 6 datasets. Significant results are identified using a paired t-test with $p = 0.05$.

The results on discrete and binary datasets indicate that SPN-SVD achieves, by far, state-of-the-art performance for an SPN on discrete datasets. This is where interpreting correlations makes a huge difference. SPN-SVD is also at par with SPN state-of-the-art on binary datasets.

4.2 Discriminatively Trained SPNs

We present results obtained by applying discriminative SPNs on two handwritten digit recognition datasets, USPS [Hull, 1994] and MNIST [LeCun et al., 1998]. USPS consists of 1100 images per digit for each of the 10 digits. Each image is 16×16. For each digit, 800 images are assigned to the training set and 300 to the test set. MNIST consists of 6000 training images per digit, each of size 28×28, and a test set of 1000 images per digit. Discriminative SPN structures were learned by DSPN-SVD in both cases. The number of extracted features d was chosen by cross-validation.

Table 4: Average CLL & Query Time. Results are normalised by number of query variables. Results are shown for 5 Caltech-101 categories, Caltech-101 & Olivetti. SVD refers to SPN−SVD, and Gens to SPN−Gens.

	30% Q., 50% Ev.		10% Q., 30% Ev.		30% Q., 30% Ev.		50% Q., 30% Ev.	
Dataset	SVD	Gens	SVD	Gens	SVD	Gens	SVD	Gens
Faces	**-0.301**	-0.318	**-0.81**	-0.96	**-0.221**	-0.319	**-0.4**	-0.53
Faces-Easy	**-0.118**	-0.16	**-0.86**	-0.908	**-0.238**	-0.318	**-0.511**	-0.543
Accordion	-0.314	**-0.312**	**-0.88**	-0.95	**-0.284**	-0.313	**-0.47**	-0.523
Airplanes	**-0.211**	-0.221	**-0.058**	-0.074	**-0.202**	-0.222	**-0.309**	-0.371
Anchor	**-0.301**	-0.419	**-0.761**	-0.944	**-0.256**	-0.331	**-0.501**	-0.569
Caltech-101 (All)	**-0.117**	-0.24	**-0.131**	-0.204	**-0.204**	-0.34	**-0.312**	0.423
Olivetti	**-0.27**	-0.289	**-0.205**	-0.234	**-0.439**	-0.472	**-0.466**	0.513
Avg. query time	31 ms	30 ms	29 ms	28 ms	30 ms	32 ms	26 ms	27 ms

Table 5: Test-set LL for 20 Binary Datasets. Bold red: an algorithm significantly outperforms the rest.

Dataset	**SPN-SVD**	**SPN-Gens**	**ID-SPN**
NLTCS	**-6**	-6.11	-6.02
MSNBC	-6.1	-6.11	**-6.04**
KDDCup 2k	-2.2	-2.18	**-2.13**
Plants	**-11.99**	-12.98	-12.54
Audio	-41.02	-40.5	**-39.79**
Jester	**-41.11**	-75.99	-52.86
Netflix	-58.02	-57.33	**-56.36**
Accidents	**-24.87**	-30.04	-26.98
Retail	**-10.6**	-11.04	-10.85
Pumsb-star	-23.7	-24.78	**-22.4**
DNA	**-80.07**	-82.52	-81.21
Kosarak	**-10.57**	-10.99	-10.6
MSWeb	**-9.22**	-10.25	-9.73
Book	**-30.18**	-35.89	-34.14
EachMovie	-52.47	-52.49	**-51.51**
WebKB	-153.5	-158.2	**-151.84**
Reuters-52	**-82.1**	-85.07	-83.35
20 Newsgrp.	-152.39	-155.93	**-151.47**
BBC	- 251	-250.69	**-248.93**
Ad	**-17.82**	-19.73	-19

Table 6 shows the results for DSPN−SVD, SPN−SVD, SPN−Gens and ID−SPN. Apart from boosting algorithms, DSPN−SVD achieves higher accuracy than other algorithms on USPS, including C4.5 as reported in Demiriz et al. [2002]. As per MNIST, DSPN−SVD also achieves the highest accuracy for an SPN, and 2.2% less than the current overall state-of-the-art accuracy on MNIST (reported as 99.79% by Wan et al. [2013] and 99.77% by Ciresan et al. [2012]). The flexibility of extracting features and building a discriminative SPN tailored for the respective dataset makes DSPN−SVD superior to SPN−SVD, as well as SPN−Gens and ID−SPN on both USPS and MNIST.

Table 6: Classification of Handwritten Digits.

Dataset	**DSPN-SVD**	**SPN-SVD**	**SPN-Gens**	**ID-SPN**
USPS	92.4%	90.2%	79%	77.1%
MNIST	97.6%	85%	81.8%	83.4%

Figure 5: Face Image Completions. The top row shows the original images; the bottom row shows images with the left half inferred using SPN−SVD.

5 CONCLUSION

State-of-the-art results when performing learning and inference on image datasets and digit classification indicate that the proposed SPN structure learning algorithms are effective.

Some important advantages of SPN−SVD over previously developed approaches are that it: (i) does not depend on local data splittings and instead globally splits the data based on rank-1 submatrix extraction; (ii) is based on correlations, which are easier to estimate than independences; and (iii) achieves considerable speedups by detecting large approximate rank-1 submatrices and avoiding redundant computations.

Interesting directions for future research include extending the discriminative setting to regression or structured-output learning by plugging more sophisticated kernels into the HSIC step, and enabling the SPN to model features optimised for different labels.

References

F R Bach. Consistency of the Group Lasso and Multiple Kernel Learning. *JMLR*, 2008.

E. Barshan, A. Ghodsi, Z. Azimifar, and M. Jahromi. Su-

pervised principal component analysis: visualization, classification and regression on subspaces and submanifolds. *In Pattern Recognition*, 44:1357–1371, 2011.

M. Biggs, A. Ghodsi, and S. Vavasis. Nonnegative matrix factorization via rank-one downdate. *In International Conference on Machine Learning (ICML)*, 25, 2008a.

M. Biggs, A. Ghodsi, and S. Vavasis. Nonnegative matrix factorization via rank-one downdate. *In http://www.arxiv.org/abs/0805.0120*, 2008b.

D. M. Chickering. The winmine toolkit. *Microsoft, Redmond, WA MSR-TR-2002-103*, 2002.

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.

A. Demiriz, K. P. Bennett, and J. Shawe-Taylor. Linear programming boosting via column generation. *In Machine Learning*, 46:225–254, 2002.

A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. *In NIPS*, 25, 2012.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples. *In proceedings of CVPR Workshop on Generative Model-Based Vision*, 2004.

K Fukumizu, F R Bach, and M I Jordan. Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces. *JMLR*, 5:73–99, 2004.

R. Gens and P. Domingos. Discriminative learning of sum-product networks. *In NIPS*, 25, 2012.

R. Gens and P. Domingos. Learning the structure of sum-product networks. *In International Conference on Machine Learning (ICML)*, 30, 2013.

A. Gretton, O. Bousquet, A. Smola, and B. Scholkopf. Statistical dependence with Hilbert-Schmidt norms. *In Algorithmic Learning Theory (ALT)*, 3734:63–77, 2005.

J. J. Hull. A database for handwritten text recognition research. *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *In Proceedings of the IEEE*, 86(11):2278–2324, 1998.

S.-W. Lee, H. Min-Oh, and Z. Byoung-Tak. Online incremental structure learning of sum–product networks. *In Neural Information Processing*, 2013.

A. Nath and P. Domingos. Learning tractable statistical relational models. *In Workshop on Learning Tractable Probabilistic Models (LTPM)*, 2014.

R. Peharz, B. C. Geiger, and F. Pernkopf. Greedy partwise learning of sum-product networks. *In Machine Learning and Knowledge Discovery in Databases*, 8189:612–627, 2013.

R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. *In Workshop on Learning Tractable Probabilistic Models (LTPM)*, 2014.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. *In UAI*, 27, 2011.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. *In International Conference on Machine Learning (ICML)*, 31, 2014.

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

Le Song, Alex J. Smola, Arthur Gretton, Karsten Borgwardt, and Justin Bedo. Supervised Feature Selection via Dependence Estimation. In *ICML*, 2007.

L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using DropConnect. *In International Conference on Machine Learning (ICML)*, 30:1058–1066, 2013.