Learning the Structure of Sum-Product Networks

Robert Gens RCG@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

Abstract

Sum-product networks (SPNs) are a new class of deep probabilistic models. SPNs can have unbounded treewidth but inference in them is always tractable. An SPN is either a univariate distribution, a product of SPNs over disjoint variables, or a weighted sum of SPNs over the same variables. We propose the first algorithm for learning the structure of SPNs that takes full advantage of their expressiveness. At each step, the algorithm attempts to divide the current variables into approximately independent subsets. If successful, it returns the product of recursive calls on the subsets; otherwise it returns the sum of recursive calls on subsets of similar instances from the current training set. A comprehensive empirical study shows that the learned SPNs are typically comparable to graphical models in likelihood but superior in inference speed and accuracy.

1. Introduction

Graphical models can compactly represent many complex distributions, but inference in them is generally intractable (Roth, 1996). Deep architectures, with many layers of hidden variables, are particularly expressive, but inference in them is correspondingly more difficult (Bengio, 2009). Recently, Poon and Domingos (2011) turned this around by introducing sum-product networks (SPNs), a class of models where adding layers increases expressiveness without losing tractability. SPNs are defined recursively, as weighted sums and products of smaller SPNs, with univariate distributions as the base case. SPNs have many interesting and important classes of probabilistic models as special cases, including mixture models, thin junction trees, non-recursive probabilistic context-free grammars, and others. They are also significantly more general than each of these. SPNs' combination of expressiveness and tractability makes them potentially a very attractive representation for many applications. In particular, they have achieved impressive results in several vision problems (Poon & Domingos, 2011; Amer & Todorovic, 2012; Gens & Domingos, 2012).

Given the structure of an SPN, its weights can be learned generatively or discriminatively. For the generative case, Poon and Domingos (2011) proposed an online hard EM algorithm, where each example is presented in turn to the SPN and the weight of each sum node's MAP child is incremented. They showed that this algorithm can successfully learn very deep SPNs, but they used a pre-defined SPN structure that was both expensive to learn (because of the large number of nodes) and insufficiently flexible (because the best nodes were often not in the predefined structure). For the discriminative case, Gens and Domingos (2012) proposed a backpropagation-style gradient descent algorithm which obtained state-of-the-art results on image classification problems, but again faced a trade-off between flexibility and cost of learning.

This trade-off can be avoided, or at least ameliorated, by learning the structure of the SPN from data. However, the only algorithm for SPN structure learning to date, proposed by Dennis and Ventura (2012), is quite limited. It essentially forms a set of hierarchical clusterings of the variables and builds the SPN based on them, and as a result is not able (except at the root) to take advantage of the context-specific independences that are crucial to SPNs' expressiveness. It clusters variables that have similar values in similar instances, and is therefore prone to splitting highly dependent variables, causing a large loss of likelihood. (For example, if X is the negation of Y, they will never be clustered.) The cost of learning and size of the SPN are worst-case exponential (order of the number of sum nodes raised to the number of sub-regions in a region of the SPN). The number of sum nodes in a region is

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

a fixed input parameter, and not learnable. Weights are learned as a post-processing step, and cannot be optimized during structure learning. The algorithm is quite complex and *ad hoc*, with no guarantee of finding even a local optimum of the likelihood, and no sense of how good the output SPN is. It has only been tested on an image completion task.

SPNs are related to multilinear formulas (Raz, 2004), arithmetic circuits (Darwiche, 2003), AND-OR graphs (Dechter & Mateescu, 2007), and other compact representations. Lowd and Domingos (2008) and Gogate et al. (2010) proposed algorithms for learning tractable models that are related to SPNs but more restricted. Lowd and Domingos' algorithm learns a Bayesian network with context-specific independence using the network's inference cost as the regularization penalty. Gogate et al. learn tractable high-treewidth Markov networks by recursively searching for features that split the remaining variables into approximately independent subsets.

In this paper, we propose the first algorithm for learning the structure of SPNs that does not sacrifice any of their expressiveness. We start by introducing a simplified definition of SPN that incorporates Poon and Domingos' conditions for tractability, avoids the use of indicator functions and network polynomials, and generalizes more easily to continuous variables. Our algorithm takes advantage of it by having a recursive structure that parallels the recursive structure of the definition. It can be viewed as an intimate combination of mixture model EM (for learning sum nodes) and graphical model structure learning (for learning product nodes). We test our algorithm on a large number of datasets from a wide variety of domains. Surprisingly, the learned SPNs typically have comparable likelihoods to unrestricted graphical models learned on the same data. At inference time they dominate in both speed and accuracy.

2. Sum-Product Networks

The *scope* of an SPN is the set of variables that appear in it. A univariate distribution is tractable iff its partition function and its mode can be computed in $O(1)$ time.

Definition 1 *A* sum-product network (SPN) *is defined as follows.*

1. *A tractable univariate distribution is an SPN.*
2. *A product of SPNs with disjoint scopes is an SPN.*
3. *A weighted sum of SPNs with the same scope is an SPN, provided all weights are positive.*
4. *Nothing else is an SPN.*

It would be straightforward to relax Part 2 of this definition to allow for non-decomposable but consistent SPNs, as in Poon and Domingos (2011), but there is no advantage in doing it for this paper's purposes.

An SPN can be represented as a rooted directed acyclic graph with univariate distributions as leaves, sums and products as internal nodes, and the edges from a sum node to its children labeled with the corresponding weights. The sub-SPN S_i rooted at a node i represents a probability distribution over its scope. For simplicity, we focus on the case of SPNs over discrete variables, but the extension to continuous ones is straightforward.

Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathcal{X}$ be a state. The unnormalized probability $S(\mathbf{x})$ of $\mathbf{x}$ according to the SPN S is the value of S's root when each leaf is set to the probability of the corresponding variable's value in $\mathbf{x}$. The partition function of an SPN is $Z = \sum_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x})$. The normalized probability of $\mathbf{x}$ is $P(\mathbf{x}) = S(\mathbf{x})/Z$. It is easily seen that, if the weights at each sum node sum to one and the leaf distributions are normalized, then $Z = 1$ and $P(\mathbf{x}) = S(\mathbf{x})$.

Theorem 1 *The following quantities can be computed in time linear in the number of edges in an SPN.*

1. *The partition function of the SPN.*
2. *The probability of evidence in the SPN.*
3. *The MAP state of the SPN.*

Proof. The proof is recursive, starting from the leaves of the SPN. By definition, the partition function of a leaf distribution can be computed in $O(1)$ time. Let Z_i be the partition function of node i, and Z_{ij} the partition functions of its children. Let $\mathcal{X}_i$ be the set of possible states of i's scope, and similarly for $\mathcal{X}_{ij}$. If i is a product node, its partition function is $Z_i = \sum_{\mathbf{x}_i\in\mathcal{X}_i} S_i(\mathbf{x}_i) = \sum_{\mathbf{x}_{i,1}\in\mathcal{X}_{i,1}} \cdots \sum_{\mathbf{x}_{ij}\in\mathcal{X}_{ij}} \cdots \; S_{i,1}(\mathbf{x}_{i,1}) \ldots S_{ij}(\mathbf{x}_{ij}) \ldots = \prod_j \sum_{\mathbf{x}_{ij}\in\mathcal{X}_{ij}} S_{ij}(\mathbf{x}_{ij}) = \prod_j Z_{ij}$. If i is a sum node, its partition function is $Z_i = \sum_{\mathbf{x}_i\in\mathcal{X}_i} S_i(\mathbf{x}_i) = \sum_{\mathbf{x}_i\in\mathcal{X}_i} \sum_j w_{ij} S_{ij}(\mathbf{x}_i) = \sum_j \sum_{\mathbf{x}_i\in\mathcal{X}_i} w_{ij} S_{ij}(\mathbf{x}_i) = \sum_j w_{ij} \sum_{\mathbf{x}_{ij}\in\mathcal{X}_{ij}} S_{ij}(\mathbf{x}_{ij}) = \sum_j w_{ij} Z_{ij}$, where w_{ij} is the weight of the ith child. Therefore the partition function of a node can be computed in time linear in its number of children, and the partition function of the root can be computed in time linear in the number of edges in the SPN.

The probability of evidence in an SPN S is just the ratio of the partition functions of S' and S, where S' is an SPN obtained from S by replacing the univariate distributions over the evidence variables by delta

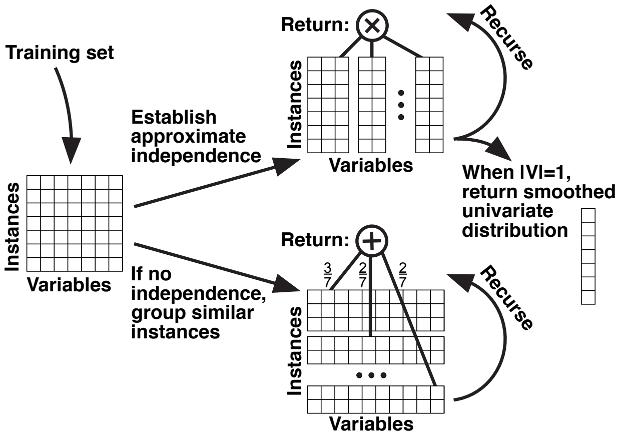
Training set
Instances
Variables
Establish approximate independence
Return:
Instances
Variables
Recurse
When |V|=1, return smoothed univariate distribution
If no independence, group similar instances
Return:
3/7
2/7
2/7
Instances
Variables
Recurse

Figure 1. A recursive algorithm for learning SPNs.

functions centered on the evidence values. Therefore it can also be computed in linear time.

The (or an) MAP state of an SPN can be computed as follows: (1) replace sum nodes with max nodes; (2) evaluate the SPN from the leaves to the root in a manner identical to computing the partition function; (3) starting from the root and following all children of each product node, for each sum node S_i choose the (or a) child with highest value of $w_{ij}M_{ij}$, where M_{ij} is the child's value computed in the previous step; (4) at each leaf node, choose the (or a) mode of the corresponding distribution. The total number of operations is thus also linear in the size of the SPN. □

3. Structure Learning

The method we propose for learning SPN structure is summarized in Algorithm 1 and illustrated in Figure 1. LearnSPN inputs an i.i.d. sample of a vector-valued variable, in the form of a matrix of instances by variables. If the vector is of unit length, LearnSPN returns the corresponding univariate distribution, with MAP estimates of the parameters. For example, for discrete variables the distribution may be a multinomial with Dirichlet prior, and for continuous ones it may be a normal with normal-Wishart prior. If the vector is of length greater than one, LearnSPN recurses on submatrices with either fewer rows or fewer columns. If it is able to split the variables into mutually independent subsets, it recurses on those, and returns the product of the resulting SPNs. Otherwise, it clusters the instances into similar subsets, recurses on those, and returns the weighted sum of the resulting SPNs. The weight of an SPN is the fraction of instances in the corresponding subset; it can also be smoothed using a Dirichlet prior.

Algorithm 1 LearnSPN(T, V)

```
input: set of instances $T$ and set of variables $V$
output: an SPN representing a distribution
    over $V$ learned from $T$
if $|V| = 1$ then
  return univariate distribution estimated
      from the variable's values in $T$
else
  partition $V$ into approximately independent
      subsets $V_j$
  if success then
    return $\prod_j$ LearnSPN($T, V_j$)
  else
    partition $T$ into subsets of similar instances $T_i$
    return $\sum_i \frac{|T_i|}{|T|} \cdot$ LearnSPN($T_i, V$)
  end if
end if
```

If there are no detectable dependencies among the variables, LearnSPN returns a fully factorized distribution. At the other extreme, if LearnSPN always fails to find independent subsets of variables until $|T| = 1$ (at which point all variables are independent), it returns a kernel density estimate of the distribution (Parzen, 1962). More typically, if $|T| \gg |V|$ LearnSPN will likely split on subsets of instances, and if $|V| \gg |T|$ it will likely split on subsets of variables. Crucially, LearnSPN can choose different variable splits for different sets of instances, resulting in tractable models with few or no conditional independences.

LearnSPN is an algorithm schema rather than a single algorithm. It can incorporate a variety of methods for splitting variables and instances into subsets. In particular, instances can be clustered using the EM algorithm (Dempster et al., 1977), and this is the method we will use in the rest of this paper. At each splitting step, we assume a naive Bayes mixture model, where all variables are independent conditioned on the cluster: $P(V) = \sum_i P(C_i) \prod_j P(X_j|C_i)$, where C_i is the ith cluster and X_j is the jth variable. For soft EM, where instances can be fractionally assigned to clusters, T needs to be extended with a weight for each instance, and each instance is passed to each cluster it has nonzero weight in. However, this is considerably less efficient than hard EM, where each instance is wholly assigned to its most probable cluster, and we will use the latter method. In either case, the learned SPN weights are now the mixing proportions $P(C_i)$. We use online EM with restarts, which automatically determines the number of clusters by assigning each new instance to its most likely cluster, possibly a new one. Overfitting is avoided via an exponential prior on the number of clusters.

Under this scheme, we can see that at each instance splitting step LearnSPN(T, V) locally maximizes the posterior probability of the sub-SPN over (T, V). The mixture model's posterior is also a lower bound on the posterior of the SPN that LearnSPN will ultimately return for (T, V), since the posterior can only increase when the recursive calls attempt to model dependencies between variables within each cluster.

An alternative to clustering instances is to split them according to the value of a specific variable or subset of variables. The best variables to split on can be chosen using a mutual information criterion. This results in an algorithm similar to Gogate et al.'s (2010) and with some of the flavor of learning graphical models with context-specific independence.

Variable splits can also be found in a number of ways. Since mutual information is a submodular function, Queyranne's algorithm can be used to find in cubic time a split of the variables into two subsets with minimum empirical mutual information (Queyranne, 1998; Chechetka & Guestrin, 2008). However, we have found this to be too slow in practice. Alternatively, we can consider only pairwise dependencies. In this case, we can apply an independence test to each pair of variables, form a graph with an edge between each pair of variables found to be dependent, and recurse on each connected component. If the graph has only one connected component, the variable split fails, and LearnSPN proceeds to form an instance split.

Let an *independence oracle* be an independence test that declares two variables X_1 and X_2 to be independent iff all subsets of variables $V_1 \ni X_1$ and $V_2 \ni X_2$ are independent. Using such an oracle, factorizing the sub-SPN into the connected components found causes no loss of likelihood. Let a *granularity sequence* be a choice of the number of clusters at each step of LearnSPN, and assume LearnSPN uses soft EM for instance clustering and maximum likelihood estimates for univariate distributions. Then LearnSPN returns a locally optimal SPN, in the sense that no higher-likelihood SPN can be reached from it by a local repartition of variables or instances. This can be summarized in the following proposition.

Proposition 1 *Given a granularity sequence, LearnSPN with an independence oracle for variable splitting and EM for instance clustering returns a locally maximum likelihood SPN.*

4. Experiments

We evaluated LearnSPN on twenty real-world datasets and compared with popular graphical model structure learning algorithms. This is a much greater number of datasets than is typical in empirical evaluations of graphical model structure learning. The diverse set of domains includes click-through logs, plant habitats, nucleic acid sequences, collaborative filtering, and many others. The number of variables in a dataset ranges from 16 to 1556, and the number of instances varies from 2k to 388k.

We used the WinMine toolkit (Chickering, 2002) to learn Bayesian network structure. WinMine allows context-specific independence in the form of a decision tree at each node (Chickering et al., 1997), and is the most sophisticated graphical model structure learning package available. For Markov network structure learning, we ran algorithms by Della Pietra et al. (1997) and Ravikumar et al. (2010). The Della Pietra et al. algorithm is the canonical Markov network structure learner. Ravikumar et al. learns structure by inducing sparsity in a large set of weights with an L1 penalty.

The dimensions of the datasets are detailed in Table 1. We used discrete datasets because the three comparison systems do not support continuous variables. Thirteen of the datasets were processed by Lowd and Davis (2010); seven were assembled by Van Haaren and Davis (2012). We used the authors' train-validation-test splits, where most datasets reserve 10% of instances for validation and 15% for testing.

4.1. Learning

To cluster instances, we used hard incremental EM (Neal & Hinton, 1998) over a naive Bayes mixture model with the exponential prior $P(S) \propto e^{-\lambda C|V|}$, where C is the number of clusters and λ is the cluster penalty. We ran ten restarts through the subset of instances T four times in random order. We estimated $P(X_j|C_i)$ with Laplace smoothing, adding 0.1 to each count.

For variable splits, we used a G-test of pairwise independence: $G(x_1, x_2) = 2\sum_{x_1}\sum_{x_2} c(x_1, x_2)\cdot\log\frac{c(x_1,x_2)\cdot|T|}{c(x_1)c(x_2)}$, where the summations range over the values of each variable and $c(\cdot)$ counts the occurrences of a setting of a variable pair or singleton (Woolf, 1957). For each dataset, the cluster penalty λ and G-test significance p were chosen based on validation set performance[1].

[1] Grid search: $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ and $p \in \{0.0015, 0.0001\}$.

Table 1. Dataset statistics.

Data set	$\|V\|$	Train	Valid.	Test	Density
NLTCS	16	16181	2157	3236	0.332
MSNBC	17	291326	38843	58265	0.166
KDDCup 2k	65	180092	19907	34955	0.008
Plants	69	17412	2321	3482	0.180
Audio	100	15000	2000	3000	0.199
Jester	100	9000	1000	4116	0.608
Netflix	100	15000	2000	3000	0.541
Accidents	111	12758	1700	2551	0.291
Retail	135	22041	2938	4408	0.024
Pumsb-star	163	12262	1635	2452	0.270
DNA	180	1600	400	1186	0.253
Kosarak	190	33375	4450	6675	0.020
MSWeb	294	29441	3270	5000	0.010
Book	500	8700	1159	1739	0.016
EachMovie	500	4524	1002	591	0.059
WebKB	839	2803	558	838	0.064
Reuters-52	889	6532	1028	1540	0.036
20 Newsgrp.	910	11293	3764	3764	0.049
BBC	1058	1670	225	330	0.078
Ad	1556	2461	327	491	0.008

Table 2. Structure learning summary.

	SPN	WinMine	SPN	DP	L1
Data set	LL	LL	PLL	PLL	PLL
NLTCS	-6.110	-6.025	-5.235	-4.941	-4.950
MSNBC	-6.113	**-6.041**	**-4.378**	-5.133	-6.060
KDDCup 2k	-2.182	-2.189	-2.084	-2.065	-2.067
Plants	-12.977	-12.647	-9.696	-9.636	**-9.405**
Audio	-40.503	-40.501	-38.604	-38.560	**-36.210**
Jester	-75.989	**-51.072**	-74.176	-50.258	**-46.610**
Netflix	-57.328	**-57.025**	-55.484	-56.419	**-51.067**
Accidents	-30.038	**-26.320**	-19.379	-26.766	**-12.446**
Retail	-11.043	-10.874	-10.590	-10.342	-10.326
Pumsb-star	-24.781	**-21.721**	-14.567	-23.665	**-9.653**
DNA	-82.523	**-80.646**	-63.431	-96.696	**-58.558**
Kosarak	-10.989	-10.834	-9.920	-10.579	-9.926
MSWeb	-10.252	**-9.697**	-8.917	-8.947	-8.728
Book	-35.886	-36.411	-34.153	-38.444	-36.346
EachMovie	-52.485	-54.368	-49.686	-64.808	-50.496
WebKB	-158.204	-157.433	-151.678	-174.448	-146.957
Reuters-52	-85.067	-87.555	-79.772	-104.173	-79.834
20 Newsgrp.	-155.925	-158.948	-151.887	-171.056	-148.316
BBC	-250.687	-257.861	**-245.340**	-272.916	-260.311
Ad	-19.733	-18.349	-11.326	-50.008	**-6.628**

We compared with Bayesian networks learned by the WinMine toolkit on test set log-likelihood (LL). We chose WinMine's per-parameter penalty κ according to validation set likelihood.

Since computing likelihood is intractable for the learned Markov networks, we instead compared test set pseudo-log-likelihood (PLL). This comparison greatly favors the Markov networks since they are trained to optimize PLL, and PLL is known to be a poor surrogate for likelihood. The methods of Della Pietra et al. (1997) and Ravikumar et al. (2010) are denoted as DP and L1, respectively. For DP, structure was learned using the open source code of Davis and Domingos (2010). L1 structure was learned using the OWL-QN package (Andrew & Gao, 2007) for L1 logistic regression. Weights were learned by optimizing PLL with the limited-memory BFGS algorithm. Markov networks with learned structure and weights were provided by Van Haaren and Davis (2012).

Table 2 shows the log-likelihood and pseudo-log-likelihood of learned methods. In all tables, bold indicates p=0.05 significance on a paired t-test. For a majority of datasets, the SPN's likelihood is not significantly different from WinMine's likelihood. SPN PLL is also comparable to DP and L1 on more than half of the datasets, even though those systems have the advantage of directly optimizing it. Presumably, if the Markov networks' likelihoods could be measured, they would be systematically worse than the SPNs'.

LearnSPN's learning times ranged from 4m to 12h, WinMine's from 1s to 10m, DP's from 16m to 24h, and L1's from 9s to 6h. The current implementation of LearnSPN could be made faster in a number of ways, such as reusing counts, as is done in WinMine.

4.2. Inference

High likelihood is not very useful if approximate inference hurts accuracy at query time. In this section we test the speed and accuracy of the learned models at query time, where the goal is to infer the probability of a subset of the variables (the query) given the values of another (the evidence).

We generated queries from the test set of each dataset, varying the fraction of randomly selected query and evidence variables. For each proportion of query and evidence variables (e.g., 10% query, 30% evidence), a thousand instances were randomly selected from the test set. For each selected test instance, a query $P(\mathbf{Q}{=}\mathbf{q}|\mathbf{E}{=}\mathbf{e})$ was created by randomly choosing the appropriate fraction of variables and assigning their values. We report the average conditional log-likelihood (CLL) of the queries given the evidence, which approximates the KL-divergence between the inferred probabilities and the true ones, sampling from the test set. We normalize the CLL by the number of query variables to facilitate comparison among datasets and query proportions.

The SPN performs exact inference in time linear in the number of edges. Since exact inference is intractable for the learned graphical models on these domains,

Table 3. Average conditional log-likelihood normalized by number of query variables for two proportions of query and evidence. The last line shows the average time per query (ms/query).

	10% Query, 30% Evidence				50% Query, 30% Evidence			
Data	SPN	WinMine	DP	L1	SPN	WinMine	DP	L1
NLTCS	-0.232	-0.232	-0.231	-0.231	-0.371	-0.373	-0.374	-0.375
MSNBC	-0.236	**-0.234**	-0.237	-0.237	-0.337	-0.338	-0.341	-0.348
KDDCup 2k	-0.030	-0.030	-0.033	-0.089	**-0.032**	-0.036	-0.038	-0.077
Plants	-0.159	-0.158	-0.178	-0.174	**-0.161**	-0.226	-0.252	-0.246
Audio	-0.394	-0.401	-0.417	-0.397	**-0.390**	-0.780	-0.808	-0.773
Jester	-0.722	-0.638	-0.550	**-0.110**	-0.727	-0.910	-0.910	**-0.095**
Netflix	-0.573	-0.588	-0.612	-0.569	**-0.566**	-0.910	-0.910	-0.909
Accidents	**-0.300**	-0.526	-0.377	-0.564	**-0.245**	-0.790	-0.837	-0.805
Retail	-0.075	-0.076	-0.077	-0.638	**-0.078**	-0.130	-0.128	-0.496
Pumsb-star	**-0.151**	-0.364	-0.443	-0.480	**-0.125**	-0.553	-0.804	-0.648
DNA	**-0.486**	-0.675	-0.791	-0.702	**-0.444**	-0.820	-0.820	-0.820
Kosarak	**-0.055**	-0.057	-0.067	-0.424	**-0.053**	-0.111	-0.108	-0.297
MSWeb	**-0.036**	-0.038	-0.039	-0.040	**-0.033**	-0.079	-0.082	-0.080
Book	**-0.068**	-0.099	-0.105	-0.101	**-0.066**	-0.295	-0.344	-0.307
EachMovie	**-0.100**	-0.191	-0.228	-0.203	**-0.101**	-0.439	-0.704	-0.481
WebKB	**-0.182**	-0.504	-0.578	-0.518	**-0.184**	-0.708	-0.723	-0.723
Reuters-52	**-0.091**	-0.258	-0.307	-0.566	**-0.090**	-0.604	-0.722	-0.656
20 Newsgrp.	**-0.166**	-0.498	-0.532	-0.603	**-0.167**	-0.715	-0.722	-0.719
BBC	**-0.238**	-0.770	-0.789	-0.793	**-0.233**	-0.718	-0.718	-0.718
Ad	**-0.010**	-0.294	-0.126	-0.158	**-0.009**	-0.652	-0.518	-0.428
Time	24	1621	2397	5003	23	1629	2386	4858

we used Gibbs sampling as implemented by the open source Libra package (Lowd). We used the Libra default of 1000 samples with 100 iterations of burn-in.

As detailed in Table 3, SPNs are two orders of magnitude faster and significantly more accurate. We also ran Gibbs sampling with more chains and iterations; even when we allowed Gibbs sampling a factor of 10 or 100 more time, it did not close the gap in accuracy.

In Figure 5, for each of several representative datasets we plot graphs of normalized CLL where the fraction of query or evidence variables is varied. When query proportion is varied, evidence is fixed at 30%, and vice-versa. The learned SPNs have much higher CLL, a disparity that becomes more pronounced with longer queries and larger datasets.

To be sure that the difference between SPNs and the other methods was not just due to Gibbs sampling, we also tried using loopy belief propagation (BP). This only computes single-variable marginals, so we measured the conditional marginal log-likelihood (CMLL) instead of CLL: $CMLL(\mathbf{X}{=}\mathbf{x}|\mathbf{e}) = \sum_{x_i \in \mathbf{Q}} \log P(X_i{=}x_i|\mathbf{e})$, where $\mathbf{Q}$ is the set of query variables and $\mathbf{e}$ is the set of evidence variable values. We ran the same set of inference tasks as detailed above for the ten variable proportions presented in the graphs, running loopy BP[2] for the Bayesian and Markov networks and measuring CMLL for all models. Averaged across the ten proportions, SPNs achieved higher CMLL than WinMine on 11 datasets, 17 compared with DP, and 16 compared with L1. This is remarkable, considering that the PLL training of Markov networks naturally pairs with testing CMLL.

Overall, these experiments show that learning SPNs is a very attractive alternative to learning graphical models. In learning accuracy, SPNs were comparable to Bayesian networks on most data sets, and comparable to or better than Markov networks. For inference accuracy and speed, SPNs are clearly the representation of choice. Further, inference in SPNs does not involve tuning settings or diagnosing convergence as with MCMC or BP, and an SPN predictably takes the same amount of time to compute any query.

5. Conclusion

Sum-product networks (SPNs) compactly represent all marginals of a distribution, in contrast to graphical models, which compactly represent only the probabilities of complete states. As a result, inference in SPNs is always tractable. In this paper, we proposed a simple schema for learning SPN structure from data. Our algorithm recursively splits an SPN into a product of SPNs over independent sets of variables, if they can be found, or into a sum of SPNs learned from subsets of the instances, otherwise. In experiments on a large number of datasets, the SPNs obtained were

[2] We used Libra with its default of 50 maximum iterations and convergence threshold of 0.0001.

typically comparable in likelihood to graphical models, but inference in them was much faster, and also more accurate.

Directions for future work include large-scale applications of SPNs, further algorithms for SPN structure and weight learning, SPNs with multivariate leaf distributions, approximating intractable distributions with SPNs, relational SPNs, and parallelizing SPN learning and inference.

Code and supplemental results are available at http://spn.cs.washington.edu/learnspn/.

Acknowledgments

The authors are grateful to Xu Miao and Scott Aaronson for helpful discussions as well as Daniel Lowd and Jesse Davis for their data. This research was partly funded by ARO grant W911NF-08-1-0242, AFRL contracts FA8750-09-C-0181 and FA8750-13-2-0019, NSF grant IIS-0803481, and ONR grant N00014-12-1-0312. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, AFRL, NSF, ONR, or the United States Government.

References

Amer, M. R. and Todorovic, S. Sum-product networks for modeling activities with stochastic structure. In *CVPR*, 2012.

Andrew, G. and Gao, J. Scalable training of L1-regularized log-linear models. In *ICML 24*, 2007.

Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2:1–127, 2009.

Chechetka, A. and Guestrin, C. Efficient principled learning of thin junction trees. In *NIPS 20*, 2008.

Chickering, D. M. The WinMine Toolkit. *Microsoft, Redmond, WA MSR-TR-2002-103*, 2002.

Chickering, D. M., Heckerman, D., and Meek, C. A Bayesian approach to learning Bayesian networks with local structure. In *UAI 13*, 1997.

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

Davis, J. and Domingos, P. Bottom-up learning of Markov network structure. In *ICML 27*, 2010.

Dechter, R. and Mateescu, R. AND/OR search spaces for graphical models. *AIJ*, 171:73–106, 2007.

Della Pietra, S., Della Pietra, V., and Lafferty, J. Inducing features of random fields. *PAMI*, 19:380–392, 1997.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

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

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

Gogate, V., Webb, W., and Domingos, P. Learning efficient Markov networks. In *NIPS 23*, 2010.

Lowd, D. The Libra Toolkit. URL `http://libra.cs.uoregon.edu/`. Version 0.5.0, 2012.

Lowd, D. and Davis, J. Learning Markov network structure with decision trees. In *ICDM 10*, 2010.

Lowd, D. and Domingos, P. Learning arithmetic circuits. In *UAI 24*, 2008.

Neal, R.M. and Hinton, G.E. A view of the EM algorithm that justifies incremental, sparse, and other variants. *NATO ASI SERIES D: Behavioural and Social Sciences*, 89:355–370, 1998.

Parzen, E. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

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

Queyranne, M. Minimizing symmetric submodular functions. *Math. Programming*, 82(1):3–12, 1998.

Ravikumar, P., Wainwright, M. J., and Lafferty, J. D. High-dimensional ising model selection using L1-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319, 2010.

Raz, R. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC 36*, 2004.

Roth, D. On the hardness of approximate reasoning. *AIJ*, 82:273–302, 1996.

Van Haaren, J. and Davis, J. Markov network structure learning: A randomized feature generation approach. In *AAAI 26*, 2012.

Woolf, B. The log likelihood ratio test (the G-test). *Annals of Human Genetics*, 21(4):397–409, 1957.

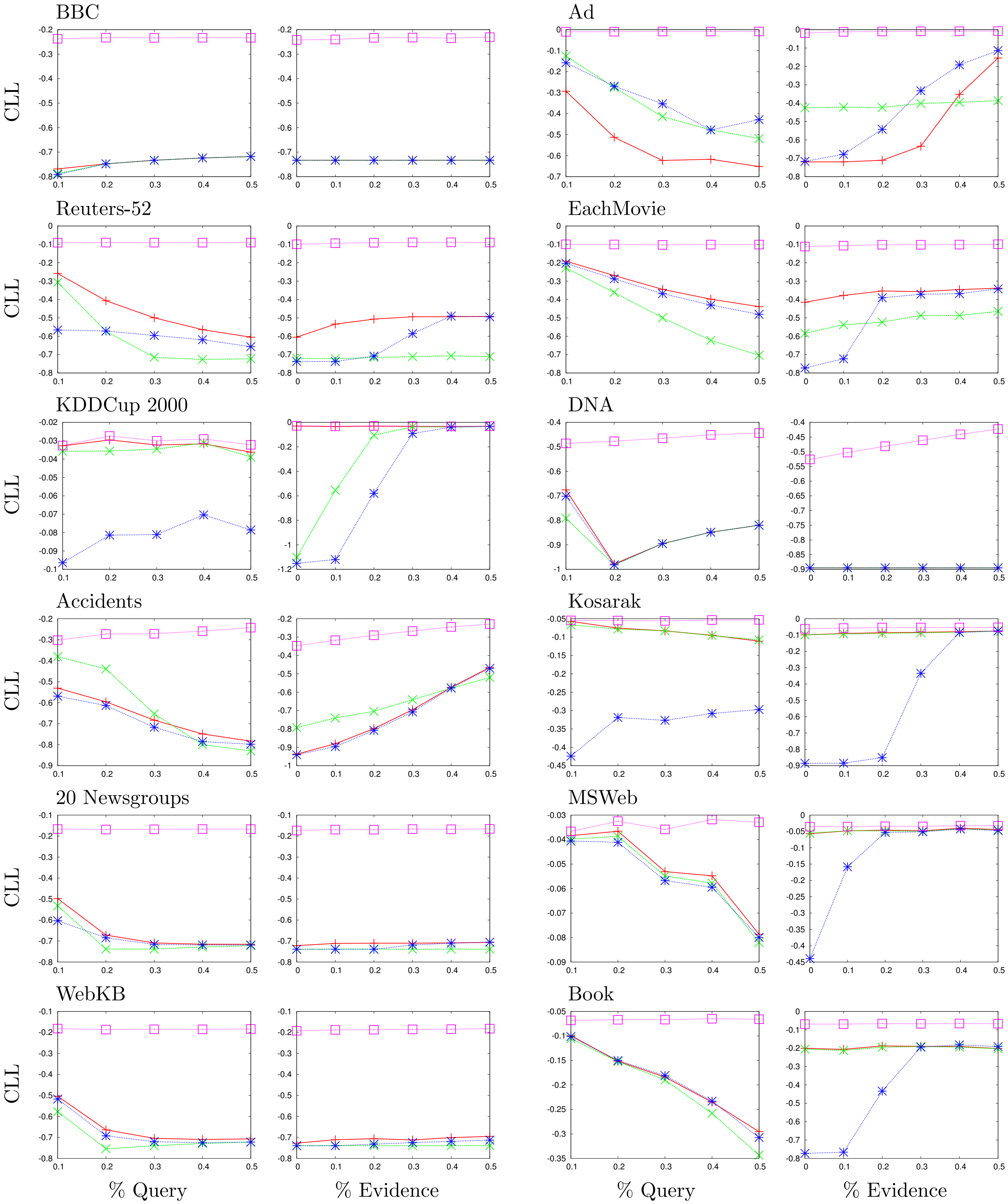

Figure 2. Average conditional log-likelihood normalized by number of query variables. Purple squares mark SPNs, blue asterisks L1, green Xs DP, and red crosses WinMine. For each dataset, the left chart fixes the fraction of evidence variables at 30% and varies the fraction of query variables; the right chart fixes query variables at 30% and varies evidence.