An Empirical Study of Methods for SPN Learning and Inference

Cory J. Butz BUTZ@CS.UREGINA.CA
University of Regina, Canada
Jhonatan S. Oliveira OLIVEIRA@CS.UREGINA.CA
University of Regina, Canada
André E. dos Santos DOSSANTOS@CS.UREGINA.CA
University of Regina, Canada
André L. Teixeira TEIXEIRA@CS.UREGINA.CA
University of Regina, Canada
Pascal Poupart PPOUPART@UWATERLOO.CA
University of Waterloo, Canada
Agastya Kalra A6KALRA@UWATERLOO.CA
University of Waterloo, Canada

Abstract

In this study, we provide an empirical comparison of methods for *sum-product network* (SPN) learning and inference. LEARNSPN is a popular algorithm for learning SPNs that utilizes chop and slice operations. As *g-test* is a standard chopping method and *Gaussian mixture models* (GMM) using expectation-maximization is a common slicing method, it seems to have been assumed in the literature that this is the best pair in LEARNSPN. On the contrary, our results show that g-test for chopping and *k-means* for slicing yields SPNs that are just as accurate. Moreover, it has been shown that implementing SPN leaf nodes as *Chow-Liu Trees* (CLTs) yields more accurate SPNs for the former pair. Our experiments show the same for the latter pair, and that neither pair dominates the other. Lastly, we report an analysis of SPN topology for unstudied pairs. With respect to inference, we derive *partial propagation* (PP), which performs SPN exact inference without requiring a full propagation over all nodes in the SPN as currently done. Experimental results on SPN datasets demonstrate that PP has several advantages over full propagation in SPNs, including relative time savings, absolute time savings in large SPNs, and scalability.

Keywords: sum-product networks, learning, inference

1. Introduction

Deep learning (Goodfellow et al., 2016) is a powerful and robust framework which represents the real-world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. Even though it has been conjectured that deeper models are more expressive than shallow ones, *sum-product networks* (SPNs) (Poon and Domingos, 2011) are the only deep learning model where this is provably the case (Delalleau and Bengio, 2011; Martens and Medabalimi, 2014). Delalleau and Bengio (2011) explicitly write that these results contribute to the motivation of learning deep SPNs. Gens and Domingos (2013) introduced LEARNSPN, which has become the standard unsupervised learning algorithm for SPNs. LEARNSPN applies two general steps, namely, a "chop" operation for splitting variables (columns) in a dataset and a "slice" operation for clustering instances (rows) in a dataset. *G-test* (Woolf, 1957) is an established chopping method, while *Gaussian mixture models* (GMM)

using expectation-maximization (Duda et al., 2012) is a standard slicing method. As such, it seems to have been implicitly assumed in the literature (Rooshenas and Lowd, 2014; Vergari et al., 2015, 2018) that g-test and GMM is the best pair of chop and slice to use in LEARNSPN.

In this paper, we present an empirical study of three other pairs of chop and slice in LEARNSPN. We include *mutual information* (Cover and Thomas, 2012) for chopping, and *k-means* (Hastie et al., 2009) for slicing. Interestingly, our results show that g-test for chopping and k-means for slicing tend to produce as accurate SPNs as the pair g-test and GMM. Secondly, we extend the investigation of (Vergari et al., 2015), which showed that implementing SPN leaf nodes as *Chow-Liu trees* (CLTs) rather than as univariate distributions tended to yield more accurate SPNs, but which focused on the g-test and GMM pair. Our results show the same improvement for g-test and k-means, and that neither of these two pairs dominates the other. Lastly, we report an analysis of SPN topology for three unstudied pairs both utilizing and not utilizing CLTs. One key finding is that using CLTs tend to yield shallower and sparser SPNs, for all pairs of chop and slice.

A second main contribution of this paper is a new algorithm, called *partial propagation* (PP), for exact inference in SPNs. SPN inference currently requires full propagation over all nodes in the SPN (Poon and Domingos, 2011; Vergari et al., 2018). Instead, PP only propagates upwards from the query leaf nodes. Experiments suggest PP has several advantages, including relative time savings, absolute time savings in large SPNs, and scalability. In particular, time savings are in orders of magnitude from 23 times faster up to over 63,000 times faster than full propagation.

The rest of this paper is organized as follows. Section 2 contains background knowledge. Implementing CLTs as leaf nodes is reviewed in Section 3. In Section 4, an empirical study of LEARNSPN methods and detailed analysis are provided. Section 5 suggests a faster SPN inference algorithm. Conclusions are drawn in Section 6.

2. Background Knowledge

We use uppercase letters X to denote variables (features) and lowercase letters x to denote their values. We denote sets of variables with boldface uppercase letters $\mathbf{X}$ and their instantiations with boldface lowercase letters $\mathbf{x}$. Also, we may use $P(x)$ instead of $P(X = x)$. We assume that the dataset instances are independent and identically distributed.

Sum-product networks (SPNs) (Poon and Domingos, 2011) are a class of deep learning models with tractable probabilistic inference. This is an attractive feature when compared to probabilistic graphical models in general, including *Bayesian networks* (Pearl, 1988), where inference is NP-hard (Cooper, 1990). The *scope* (Gens and Domingos, 2013) of an SPN is the set of variables that appear in it. A univariate distribution is tractable if its partition function and its mode can be computed in O(1) time (Gens and Domingos, 2013).

Definition 1 *(Gens and Domingos, 2013) An SPN is defined as follows:* (i) *a tractable univariate distribution is an SPN;* (ii) *a product of SPNs with disjoint scopes is an SPN;* (iii) *a weighted sum of SPNs with the same scope is an SPN, provided all weights are positive; and* (iv) *nothing else is an SPN.*

An SPN can be graphically understood as a rooted directed acyclic graph. Each internal node is either a sum or product operation. Each leaf node is a univariate distribution over its variable. Each edge from a sum node to a child has a positive weight.

The value of an SPN $\mathcal{S}$ is the value of its root. The value of a product node v_i is the product of the values of its children. The value of a sum node v_i is

$$\sum_{v_j \in Ch(v_i)} w_{ij} val(v_j), \tag{1}$$

where $Ch(v_i)$ are the children of v_i, $val(v_j)$ is the value of node v_j, and w_{ij} is the weight of the edge from v_i to v_j. The value of a leaf node is the value of its univariate distribution.

The *depth* (Goodfellow et al., 2016) of an SPN $\mathcal{S}$ is the number of nodes in the longest path in $\mathcal{S}$ from the root to a leaf. Gens and Domingos (2013) measure the quality of an SPN $\mathcal{S}$ using likelihood. The *likelihood* of an SPN $\mathcal{S}$ is defined as the product of the values of $\mathcal{S}$, for each instantiation of the dataset. The higher the product, the better the quality.

Gens and Domingos (2013) introduced LEARNSPN, which has become the standard unsupervised learning algorithm for SPNs. LEARNSPN applies two general steps, namely, a "chop" operation for splitting variables (columns) in the dataset and a "slice" operation for clustering instances (rows) in the dataset.

The chopping method of LEARNSPN splits a dataset vertically. Here, we consider two chopping methods, namely, *g-test* (Woolf, 1957) and *mutual information* (Cover and Thomas, 2012). Although the scoring metric is different, both methods adhere to the following scoring procedure.

One variable $X_i \in V$ is chosen at random. Then the score $s(X_i, X_j)$ is computed, for all other variables X_j. If $s(X_i, X_j)$ is greater than a threshold, then X_j is considered to be similar to X_i and they are grouped together. Next, $s(X_j, X_k)$ is computed, for all X_k not previously grouped with X_i. Again, if $s(X_j, X_k)$ is greater than a threshold, then X_k is considered similar to X_j, which means X_k is similar to X_i, and X_k is grouped with X_i and X_j. This process repeats until no more variables can be grouped with X_i.

G-test tests pairwise independence of variables as follows:

$$G(X_i, X_j) = 2 \sum_{x_i} \sum_{x_j} c(x_i, x_j) log \frac{c(x_i, x_j)|T|}{c(x_i)c(x_j)}, \tag{2}$$

where $c(\cdot)$ counts the occurrences of a setting of a variable pair or singleton (Woolf, 1957), $|T|$ is the number of dataset instances, and log is the natural logarithm.

The second method we consider for the chopping operation is mutual information. *Mutual information* (MI) (Cover and Thomas, 2012) tests pairwise independence of variables as follows:

$$MI(X_i, X_j) = \sum_{x_i} \sum_{x_j} c(x_i, x_j) log \frac{c(x_i, x_j)}{c(x_i)c(x_j)}. \tag{3}$$

The slicing method of LEARNSPN splits a dataset horizontally. Here, we consider two slicing methods, namely, *k-means* (Hastie et al., 2009) and *Gaussian mixture models* (GMM) using expectation-maximization (Duda et al., 2012). As in (Vergari et al., 2015), here we focus on two clusters.

K-means is a method for partitioning the rows $I_0, \ldots, I_n$ of a dataset into k clusters. Each I_i is assigned to the cluster k_j that minimizes the Euclidean distance to the mean μ_j of that cluster: $||I_i - \mu_j||$. The mean μ_j is recalculated as the mean of those I_i assigned to cluster k_j. Then the above process is repeated using the recalculated μ_j. The process stops after 300 iterations or when

all μ_j do not change. For each cluster k_j, the initial mean μ_j can be randomly chosen without affecting the output of k-means.

The second method we consider for the slicing operation is GMM. The process is nearly identical to that for k-means, except that the representation of each cluster and the assignment of instances to cluster are more involved. Each cluster is a Gaussian distribution represented by its mean and covariance matrix. These can be randomly assigned initially without disturbing the end result. Next, the probability of each instantiation I_i being in each cluster is computed. I_i is assigned to the cluster with the higher probability. Do this for every instance. The mean and covariance matrix are recalculated for each cluster by following the expectation-maximization algorithm. This process repeats 100 times or until the clusters do not change. More formally, each cluster k_j is represented by a Gaussian distribution, denoted $\mathcal{N}(I|\mu_j, \Sigma_j)$, where μ_j is its vector mean and Σ_j is its covariance matrix. The scoring of each instance I_i being in each cluster is the probability $p(k_j|I_i)$, which can be computed as:

$$p(k_j|I_i) = \frac{w_j \, \mathcal{N}(I|\mu_j, \Sigma_j)}{z},$$

where w_j is a cluster weight and z is a normalization function.

A refinement of LEARNSPN was suggested by Vergari et al. (2015), and is given as Algorithm 1. There are four input parameters, with chop and slice methods understood. T is the collection of instances over variables V. The Laplace smoothing parameter is α. Finally, m denotes the minimum number of instances needed for a chop operation. Line 4 is the focus of the next section.

Algorithm 1 LEARNSPN(T, V, α, m)

Input: a set of instances T over variables V
Output: an SPN S
Main:

1: **if** $|V| == 1$ **then**
2: $\quad S \leftarrow$ UNIVARIATEDISTRIBUTION(T, V, α)
3: **else if** $|T| < m$ **then**
4: $\quad S \leftarrow$ CLTFACTORIZATION(T, V, α) ▷ CLT instead of a naive factorization
5: **else**
6: $\quad \{V_j\}_{j=1}^{C} \leftarrow$ SPLITVARIABLES(V, T) ▷ Using chop method
7: $\quad$ **if** $C > 1$ **then**
8: $\quad\quad S \leftarrow \prod_{j=1}^{C}$ LEARNSPN(T, V, α, m)
9: $\quad$ **else**
10: $\quad\quad \{T_i\}_{i=1}^{R} \leftarrow$ CLUSTERINSTANCES(T, V) ▷ Using slice method
11: $\quad\quad S \leftarrow \prod_{i=1}^{R} \frac{|T_i|}{|T|}$LEARNSPN$(T_i, V, \alpha, m)$
12: **return** S

3. CLTs and the Pair G-test and GMM

When focusing on g-test for chopping and GMM for slicing, (Vergari et al., 2015) has shown that LEARNSPN yields more accurate SPNs when leaf nodes are implemented as *Chow-Liu trees* (CLTs) rather than as univariate distributions.

Chow and Liu (1968) developed an elegant method to approximate an n-dimensional discrete probability distribution by a product of second-order distributions. The conditional independences learned by their algorithm are represented by a singly-connected DAG, called a *first-order dependence tree* in (Chow and Liu, 1968). Here, we use the term *Chow-Liu tree* (CLT). The important point is that their method is guaranteed to find an *optimal* approximation of the joint distribution under the given scoring metric (a measure of closeness) and the restriction to using second-order distributions.

A CLT is built as follows (Vergari et al., 2015). Compute the mutual information $MI(X_i, X_j)$ for every pair $X_i, X_j \in V, i \neq j$. Construct a complete graph $\mathcal{G}$ and assign the edge weight between X_i and X_j as $MI(X_i, X_j)$. Find a maximal spanning tree of $\mathcal{G}$. Randomly pick a root node and direct all edges away from the root. Construct the conditional probability table $P(X_i|Pa(X_i))$ using the given dataset, where $Pa(X_i)$ denotes the parent node of X_i in the constructed polytree.

In (Vergari et al., 2015), the accuracy of SPNs was improved by implementing leaf nodes with CLTs rather than with naive factorizations of univariate distributions. CLTs can improve the network likelihood by capturing finer local dependencies when estimating leaf distributions (Vergari et al., 2015). A second advantage of using CLTs is that they admit linear inference (Chow and Liu, 1968). Hence, the complexity of inference in SPNs with CLT leaf nodes remains linear (Vergari et al., 2015). Thus, CLTs are simple and tractable distributions, yet are able to model more dependencies than a naive factorization (Vergari et al., 2018).

4. CLTs and LearnSPN

In this section, we investigate the role of CLT leaf nodes in LEARNSPN for three other combinations of chop and slice methods.

We empirically evaluate each LEARNSPN combination in 20 benchmark datasets (Lowd and Davis, 2010; Van Haaren and Davis, 2012). The dataset characteristics are described in Table 1. We conduct a grid search over the following hyperparameters taken from (Vergari et al., 2015; Gogate et al., 2010). The g-test threshold values are $\{5, 10, 15, 20\}$, the α values are $\{0.1, 0.2, 0.5, 1.0, 2.0\}$, and the minimum number of instances values are $\{10, 50, 100, 500\}$. The MI threshold values were selected based on the calculated minimum and maximum MI values per dataset with the goal of learning the most accurate SPN, since the MI threshold does not take into account the number of instances in the dataset.

Table 2 reports SPN accuracy results after a grid search. SPN leaf nodes are implemented as univariate distributions in columns 2-5, while columns 6-9 are when leaf nodes are implemented as CLTs. Here, (g,k) means (g-test,k-means), (MI,k) denotes (MI,k-means), and G stands for GMM in (g,G) and (MI,G). As in (Vergari et al., 2015), accuracy values are shown to four decimal places, winners per leaf node implementation are shown in boldface, and overall winners are underlined.

In terms of SPN accuracy, g-test and GMM and the pair g-test and k-means can be seen as being tied in both univariate and CLT cases. Thus, one key finding of our experiments is that g-test and k-means is a viable alternative to the common pair of g-test and GMM.

The pair of MI and k-means also warrants some discussion. This pair is the overall winner in two datasets (Audio and Jester) when using univariate distributions as leaf nodes. And it wins the univariate setting in a third dataset, Netflix. In general, CLTs improve SPN accuracy in all combinations of chop and slice in LEARNSPN. CLTs are used in 15 of 20 overall winners.

Table 1: 20 benchmark datasets used in unsupervised deep learning.

Dataset	# variables	# training	# validation	# testing
NLTCS	16	16181	2157	3236
MSNBC	17	291326	38843	58265
KDDCup2K	65	180092	19907	34955
Plants	69	17412	2321	3482
Audio	100	15000	2000	3000
Jester	100	9000	1000	4116
Netflix	100	15000	2000	3000
Accidents	111	12758	1700	2551
Retail	135	22041	2938	4408
Pumb-star	163	12262	1635	2452
DNA	180	1600	400	1186
Kosarek	190	33375	4450	6675
MSWeb	294	29441	3270	5000
Book	500	8700	1159	1739
EachMovie	500	4525	1002	591
WebKB	839	2803	558	838
Reuters-52	889	6532	1028	1540
20Newsgroup	910	11293	3764	1540
BBC	1058	1670	225	330
Ad	1556	2461	327	491

As done in (Vergari et al., 2015) for the pair g-test and GMM, we now turn our attention to the graphical characteristics of the learned SPN.

Table 3 reports the number of edges in the various settings, where the winning pair with the lowest number of edges is in boldface per leaf node implementation and overall winners are underlined. The use of CLTs never increases the number of edges in the pairs g-test and GMM and MI and GMM. In general, CLTs tend to lower the number of edges compared to using univariate distributions.

Table 4 shows the number of layers in the learned SPN, where the winning pair with the greatest number of layers is in boldface per leaf node implementation and overall winners are underlined. Our results suggest that g-test yields deeper SPNs whereas MI produces shallower SPNs. The use of CLTs never increases the number of layers for pair g-test and GMM. Moreover, for the other 3 pairs, the use of CLTs doesn't increase the number of layers in the majority of cases.

We introduce the edges-per-layer ratio, denoted r, as the number of edges divided by the number of layers. A higher value of r means the SPN is dense; a lower value means the SPN is sparse. Table 5 reports this ratio under various settings, where the winning pair with the lowest value of r is in boldface per leaf node implementation and overall winners are underlined. Although CLTs lower the number of layers, which would lead to dense SPNs, the lowering in the number of edges is more pronounced. This is why CLTs yield sparser SPNs. More generally, our results suggest that MI yields dense SPNs, while g-test produces sparse SPNs.

Table 2: SPN accuracy after grid search in LearnSPN with and without the use of CLTs.

	Univariate				Chow-Liu Tree			
Dataset	(g,k)	(MI,k)	(g,G)	(MI,G)	(g,k)	(MI,k)	(g,G)	(MI,G)
NLTCS	-6.0540	-6.0672	**-6.0401**	-6.0541	-6.0507	-6.0603	**-6.0380**	-6.0510
MSNBC	-6.0439	-6.0436	**-6.0398**	-6.0403	-6.0436	-6.0458	**-6.0398**	-6.0428
KDDCup2K	-2.1432	-2.1522	-2.1454	**-2.1398**	**-2.1441**	-2.1623	-2.1452	-2.1470
Plants	-12.8811	-13.0127	**-12.8801**	-12.9124	-12.8023	-12.7737	**-12.7194**	-12.7290
Audio	-40.1169	**-40.0530**	-40.7407	-40.4305	**-40.1104**	-40.1549	-40.4953	-40.4180
Jester	-52.9799	**-52.9483**	-53.9569	-53.5165	**-52.9879**	-53.0224	-53.7964	-53.6560
Netflix	-56.7965	**-56.6903**	-58.3797	-57.7690	**-56.6467**	-56.6544	-58.1006	-57.9649
Accidents	**-28.9153**	-35.4896	-28.9551	-35.6297	-28.8570	-30.2291	**-28.8562**	-30.1731
Retail	**-10.9444**	-11.1678	-10.9479	-11.0923	**-10.9343**	-11.1678	-10.9484	-11.1000
Pumb-star	-23.3351	-29.7128	**-23.3134**	-29.6281	-23.0648	-24.0734	**-22.9298**	-24.0073
DNA	**-81.5530**	-98.2353	-81.6295	-97.9827	**-81.4946**	-85.7810	-81.6295	-86.6187
Kosarek	-10.7942	-11.3216	**-10.7297**	-11.0351	-10.7644	-11.1537	**-10.7088**	-10.8278
MSWeb	-9.9942	-10.2431	**-9.8477**	-10.0384	-10.0016	-10.2464	**-9.8259**	-9.8443
Book	-34.7721	-35.4775	**-34.2895**	-35.1611	-34.6492	-35.7391	**-34.2895**	-34.9308
EachMovie	-52.2136	-52.5846	**-51.5116**	-52.2844	-52.6627	-53.0856	**-51.5688**	-52.4801
WebKB	**-154.6667**	-156.7409	-154.7319	-156.9452	**-154.2209**	-154.6686	-154.5529	-154.7833
Reuters-52	**-83.8645**	-85.6881	-84.0903	-85.6974	**-83.8598**	-85.1386	-84.0099	-84.5777
20Newsgroup	**-152.5908**	-154.1383	-153.3014	-153.6602	-153.4568	-156.6570	**-153.3028**	-155.7261
BBC	**-247.6164**	-250.1484	-248.5855	-251.1441	**-247.3655**	-255.5471	-247.5619	-253.3433
Ad	**-15.7866**	-27.5521	-15.7918	-28.3282	**-15.0341**	-16.4187	-15.4933	-15.8275

Finally, a few comments regarding the hyperparameter m in LEARNSPN are worthy. We considered the values 10, 50, 100, and 500 (Vergari et al., 2015). When SPN leaf nodes are implemented as univariate distributions, the grid search selected the lowest values 10 and 50 in the majority of datasets; the highest value 500 was never chosen. In contrast, when utilizing CLTs, the highest value 500 was selected by the grid search in over half the datasets. And the two highest values 100 and 500 were chosen in the strong majority of cases. These results are consistent with (Vergari et al., 2015), where the focus was on g-test and GMM.

5. SPN Exact Inference with Partial Propagation

SPN exact inference is conducted by a *full propagation* (FP) from all leaf nodes to the root (Poon and Domingos, 2011; Vergari et al., 2018). Here, we introduce a faster SPN exact inference algorithm, called *Partial Propagation* (PP).

First, we define a few pertinent SPN concepts. An SPN is *complete* if, for every sum node, its children have the same scope; otherwise, it is incomplete. An SPN is *decomposable* if, for every product node, the scopes of its children are pairwise disjoint. An SPN is *valid*, if it is complete and decomposable. A valid SPN defines a joint probability distribution and can answer queries in time linear in its size (number of edges) (Poon and Domingos, 2011). For each node N in an SPN $\mathcal{S}$, the *ancestors* of N, denoted $An(N)$, are those variables having a directed path to N. We define $An(\mathbf{N})$ for a set $\mathbf{N}$ of nodes in the obvious way. Furthermore, without loss of generality, we assume that the SPNs considered here are normalized (Peharz et al., 2015).

Given a query $P(\mathbf{x})$ posed to an SPN, we call the nodes in $\mathbf{X} \cup An(\mathbf{X})$ *relevant*; the rest are *irrelevant*. We show that irrelevant nodes do not affect the value of a relevant node.

Table 3: The number of edges in the learned SPN.

	Univariate				Chow-Liu Tree			
Dataset	(g,k)	(MI,k)	(g,G)	(MI,G)	(g,k)	(MI,k)	(g,G)	(MI,G)
NLTCS	**1119**	24701	1129	17343	**661**	1139	1027	3129
MSNBC	5307	53379	**4073**	39230	5307	18180	**3833**	17148
KDDCup2K	5128	128830	**4446**	61555	5065	64805	**4446**	42770
Plants	**13789**	171118	15129	72652	9418	27720	**6302**	9012
Audio	**10968**	23634	17863	46359	10968	4747	**4095**	4949
Jester	**6614**	13635	9851	28078	6614	**2727**	3068	3333
Netflix	**14770**	22624	30054	46157	**4462**	4646	4607	5050
Accidents	**11746**	42446	11977	43456	11518	**4480**	10065	4592
Retail	3222	**272**	700	54128	3222	**272**	700	14144
Pumb-star	13336	39625	**12728**	35840	11053	6068	10220	**5576**
DNA	5199	**442**	3306	8869	5199	**724**	3306	905
Kosarek	4676	188028	**3481**	38060	2462	204752	**2232**	34380
MSWeb	13338	159005	**9065**	118258	9310	159005	**9065**	11722
Book	19097	64629	**3894**	121242	18763	63627	**3894**	22044
EachMovie	**22050**	63126	24458	41054	26690	6396	24458	**5674**
WebKB	**8902**	60480	9301	37800	**5608**	8400	8877	6720
Reuters-52	**70272**	100570	82084	78801	87414	20470	82084	**18690**
20Newsgroup	**17621**	35529	174703	34618	172565	38262	174703	**33707**
BBC	69310	33413	68117	**25531**	69310	**5295**	68117	**5295**
Ad	30097	271809	**20823**	191454	23748	39407	20823	**14015**

Lemma 2 *Consider a valid and normalized SPN $\mathcal{S}$ on variables $\mathbf{U}$ and a query $P(\mathbf{x})$, $\mathbf{X} \subset \mathbf{U}$. Then all irrelevant nodes have value 1.*

Proof Consider an irrelevant node v, where v is a leaf node with univariate distribution $P(Y)$ and $Y \cap \mathbf{X} = \emptyset$. By definition, $value(v) = 1$.

Without loss of generality, consider an irrelevant product node v with only leaf nodes as its n children. By definition, all n children are irrelevant. Hence,

$$\prod_{v' \in Ch(v)} value(v') = 1. \tag{4}$$

Next, again without loss of generality, consider an irrelevant sum node v with only leaf nodes as its n children. By definition, all n children are irrelevant. Thus,

$$value(v) = \sum_{v' \in Ch(v)} w \cdot value(v') = w_1 \cdot 1 + \ldots + w_n \cdot 1 = 1 \cdot (w_1 + \ldots + w_n) = 1, \tag{5}$$

since $\mathcal{S}$ is normalized. A similar argument holds for all remaining irrelevant nodes. ■

Theorem 3 shows that PP is sound.

Table 4: The number of layers in the learned SPN.

	Univariate				Chow-Liu Tree			
Dataset	(g,k)	(MI,k)	(g,G)	(MI,G)	(g,k)	(MI,k)	(g,G)	(MI,G)
NLTCS	15	3	**17**	5	**15**	2	**15**	3
MSNBC	**31**	**31**	25	5	**31**	2	25	21
KDDCup2K	**69**	3	31	3	**77**	2	31	2
Plants	**29**	7	27	21	**29**	2	22	18
Audio	**27**	3	**27**	3	**27**	2	17	2
Jester	11	3	**23**	3	11	2	**16**	2
Netflix	15	3	**31**	3	8	2	**18**	2
Accidents	25	15	**27**	3	25	2	**26**	2
Retail	**27**	3	13	3	**27**	2	13	2
Pumb-star	25	15	**27**	17	**23**	2	22	2
DNA	**15**	5	13	3	**15**	2	13	2
Kosarek	**29**	15	19	19	**17**	2	15	2
MSWeb	**33**	3	27	19	**35**	2	27	20
Book	**69**	3	11	3	**55**	2	11	2
EachMovie	25	3	**29**	9	**29**	4	**29**	6
WebKB	**17**	3	**17**	3	11	2	**15**	2
Reuters-52	**35**	3	27	7	**33**	2	27	2
20Newsgroup	21	3	**31**	3	**33**	2	31	2
BBC	**29**	9	25	9	**29**	2	25	2
Ad	**75**	31	33	9	**59**	20	33	12

Theorem 3 *(Partial Propagation). Consider an SPN $\mathcal{S}$ on $\mathbf{U}$ and $\mathbf{X} \subseteq \mathbf{U}$. The value $\mathcal{S}(\mathbf{x})$ can be correctly computed by using only nodes in $\mathbf{X} \cup An(\mathbf{X})$.*

Proof The value of all leaf nodes are given. Thus, irrelevant nodes play no role in determining the value of relevant leaf nodes. Consider a sum node that is relevant. Since $\mathcal{S}$ is complete, all its children have the same scope. Since the sum node is relevant, all of its children are relevant. Thus, irrelevant nodes do not determine the value of a relevant sum node. Lastly, consider a product node that is relevant. Since $\mathcal{S}$ is decomposable, the scopes of its children are pairwise disjoint. Hence, some children may be relevant while others are irrelevant. However, the value of each irrelevant node is 1. Therefore, the irrelevant nodes are immaterial to the value of a relevant product node. ■

It is straightforward to establish that PP has linear complexity.

A naive implementation of PP will not lead to any time savings in SPN inference. For instance, checking whether every node in the network is relevant or irrelevant is wasteful. We implement PP with a recursive procedure, which begins with a call for the value at the root node. The key is to check whether the scope of each child contains at least one query variable. A child node is relevant only if this is true; otherwise, it is irrelevant. Next, recursively call for the value of the relevant children. Algorithm 2 formalizes this discussion and takes a query $P(\mathbf{x})$ posed to an SPN $\mathcal{S}$.

The experiments are performed using a Python implementation running on a MacBook Pro 2015 with a 2.2 GHz Intel Core i7 Processor and 16 GB RAM. The 9 SPNs used are listed in column 1

Table 5: The edges-per-layers ratio r (denseness) of the learned SPN.

	Univariate				Chow-Liu Tree			
Dataset	(g,k)	(MI,k)	(g,G)	(MI,G)	(g,k)	(MI,k)	(g,G)	(MI,G)
NLTCS	75	8234	**66**	3469	**44**	570	68	1043
MSNBC	171	1722	**163**	7846	171	9090	**153**	817
KDDCup2K	**74**	42943	143	20518	**66**	32403	143	21385
Plants	**475**	24445	560	3460	325	13860	**286**	501
Audio	**406**	7878	662	15453	406	2374	**241**	2475
Jester	601	4545	**428**	9359	601	1364	**192**	1667
Netflix	985	7541	**969**	15386	558	2323	**256**	2525
Accidents	470	2830	**444**	14485	461	2240	**387**	2296
Retail	119	91	**54**	18043	119	136	**54**	7072
Pumb-star	533	2642	**471**	2108	481	3034	**465**	2788
DNA	347	**88**	254	2956	347	362	**254**	453
Kosarek	**161**	12535	183	2003	**145**	102376	149	17190
MSWeb	404	53002	**336**	6224	**266**	79503	336	586
Book	**277**	21543	354	40414	**341**	31814	354	11022
EachMovie	882	21042	**843**	4562	920	1599	**843**	946
WebKB	**524**	20160	547	12600	**510**	4200	592	3360
Reuters-52	**2008**	33523	3040	11257	**2649**	10235	3040	9345
20Newsgroup	**839**	11843	5636	11539	**5229**	19131	5636	16854
BBC	**2390**	3713	2725	2837	**2390**	2648	2725	2648
Ad	**401**	8768	631	21273	**403**	1970	631	1168

Algorithm 2 PARTIAL PROPAGATION$(N, \mathbf{x}, \mathcal{S})$

Input: the root node N, a query $P(\mathbf{x})$, and an SPN $\mathcal{S}$
Output: the value $\mathcal{S}(\mathbf{x})$
Main:

```
if scope(N) ∩ X ≠ ∅ then                    ▷ Compute only if node N is relevant
    if N is a sum then
        return Σ_{C∈Ch(N)} w_C · PARTIAL PROPAGATION(C, x, S)
    else if C is a product then
        return Π_{C∈Ch(N)} PARTIAL PROPAGATION(C, x, S)
    else
        return C ∩ x
```

of Table 6. Column 2 shows the number of nodes in each SPN. For each SPN, 1000 queries $P(\mathbf{x})$ were randomly generated and processed by FP and PP. Average times for inference are reported in seconds in columns 3 and 4.

Table 6 is encouraging in several ways. First, it can be seen that PP was always faster than FP. Second, the time savings in column 5 are in orders of magnitude from 23 times faster up to over 63,000 times faster. Third, the absolute time difference can be worthy. On the two largest SPNs, FP takes 12 to 37 seconds, while PP requires a small fraction of a second. This will be valuable in

Table 6: An empirical comparison of FP and PP in SPNs.

SPN	# Nodes	FP (s)	PP (s)	FP/PP
cancer	75	0.00602	0.00026	23
earthquake	87	0.00753	0.00026	29
survey	143	0.01235	0.00027	46
asia	172	0.01776	0.00030	59
sachs	933	0.07568	0.00047	161
child	1,953	0.24504	0.00046	533
alarm	4,782	0.64698	0.00058	1,115
hailfinder	99,916	12.69837	0.00154	8,246
insurance	286,328	37.58020	0.00059	63,695

applications such as recommendation systems, which have millions of online users, each of which requires an answer in real-time. In these applications, PP can alleviate the total time bottleneck. Lastly, Table 6 indicates that PP scales. Compare the smallest SPN with the largest SPN. Whereas FP went from a fraction of a second up to 37 seconds, PP roughly doubled. In fact, the larger the SPN, the greater the time savings of PP over FP.

6. Conclusions

This empirical study of SPNs has yielded several insights. With respect to learning, our results show that g-test and k-means is equally effective to the commonly used pair of g-test and GMM. Similar to the result found in (Vergari et al., 2015) for g-test and GMM, our results show that implementing SPN leaf nodes as CLTs yields more accurate SPNs for 3 other pairs of chop and slice in LEARNSPN. In fact, CLT utilization occurred in 15 out of 20 overall winners. With respect to SPN topology, g-test yields sparse SPNs, while MI produces dense SPNs. The use of CLT leaf nodes never increases the number of edges for the pair g-test and GMM and for the pair MI and GMM. More generally, CLTs tend to lower the number of edges compared to using univariate distribution leaf nodes. With respect to the number of layers, g-test yields deeper SPNs than MI does.

Another main contribution of this manuscript is a new exact inference algorithm for SPNs, called *partial propagation* (PP). SPN inference is currently conducted using a full propagation over all nodes in a SPN. In contrast, PP only propagates upwards from query leaf nodes. The experimental results in Table 6 are very encouraging. In particular, time savings are in orders of magnitude from 23 times faster up to over 63,000 times faster than full propagation.

References

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054142.

G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

T. Cover and J. Thomas. *Elements of information theory*. John Wiley & Sons, 2nd edition, 2012.

O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. In *Proceedings of the Twenty-Fourth Conference on Advances in Neural Information Processing Systems*, pages 666–674, 2011.

R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley & Sons, 2012.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *Proceedings of the Thirtieth International Conference on Machine Learning*, pages 873–880, 2013.

V. Gogate, W. Webb, and P. Domingos. Learning efficient markov networks. In *Proceedings of the Twenty-Third Advances in Neural Information Processing Systems*, pages 748–756, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

T. Hastie, R. Tibshirani, and J. Friedman. Overview of supervised learning. In *The elements of statistical learning*, pages 9–41. Springer, 2009.

D. Lowd and J. Davis. Learning markov network structure with decision trees. In *Proceedings of the Tenth IEEE International Conference on Data Mining*, pages 334–343, 2010.

J. Martens and V. Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 744–752, 2015.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346, 2011.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the Thirty-First International Conference on Machine Learning*, pages 710–718, 2014.

J. Van Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1148–1154, 2012.

A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358, 2015.

A. Vergari, R. Peharz, N. Di Mauro, A. Molina, K. Kersting, and F. Esposito. Sum-product autoencoding: Encoding and decoding representations using sum-product networks. In *Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence*, page (to appear), 2018.

B. Woolf. The log likelihood ratio test (the g-test). *Annals of human genetics*, 21(4):397–409, 1957.