Learning Sum-Product Networks

Renato Lui Geh

Institute of Mathematics and Statistics — University of São Paulo

Introduction

Definition 1 (Generalized sum-product network).

A sum-product network (SPN) is a DAG where each node n is either:

1. A tractable univariate probability distribution;
2. A product of SPNs: $v_n = \prod_{j \in \mathrm{Ch}(n)} v_j$; or
3. A weighted sum of SPNs: $v_n = \sum_{j \in \mathrm{Ch}(n)} w_{n,j} v_j$.

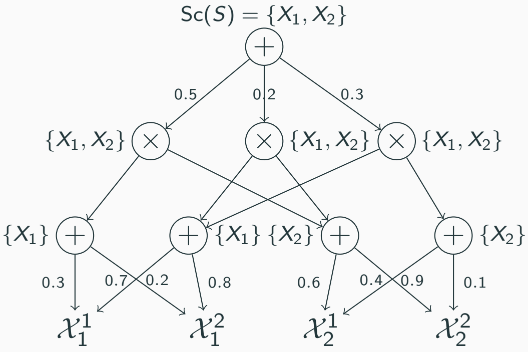
Where v_n is the value of node n, $\mathrm{Ch}(n)$ its set of children and $w_{n,j}$ the weight of edge $n \to j$.

Scope

The scope $\mathrm{Sc}(n)$ of node n is the union of the scope of its children.

The scope of a leaf is the set of all variables in the distribution.

Let S be the root of the SPN below:



Validity

Definition 2 (Validity).

Let S be an SPN. If S correctly computes and marginalizes an unnormalized probability $\phi(\mathbf{X})$, then it is said to be *valid*.

If for every sum node n

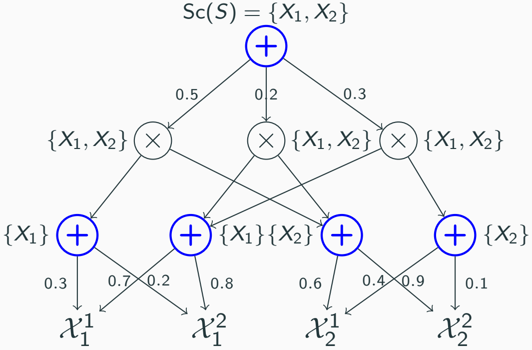
$$\forall j \in \mathrm{Ch}(n), w_{n,j} \geq 0 \text{ and } \sum_{j \in \mathrm{Ch}(n)} w_{n,j} = 1$$

then S represents the probability distribution itself.

A **sufficient**, yet not necessary, condition for validity is *completeness* and *consistency* (Poon and Domingos 2011).

Definition 3 (Completeness).

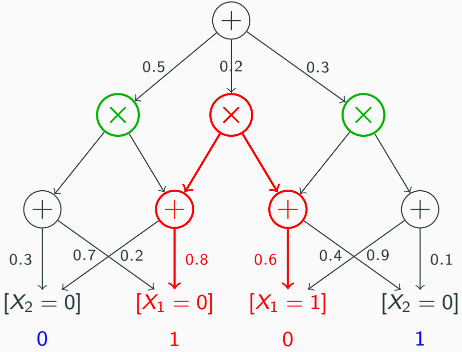
An SPN S is said to be complete, iff for each sum node $s \in S$, all children of s have same scope.



Consistency

Definition 4 (Consistency).

An SPN S is said to be consistent, iff no variable appears with a value v in one child of a product node, and valued u, with $u \neq v$, in another.

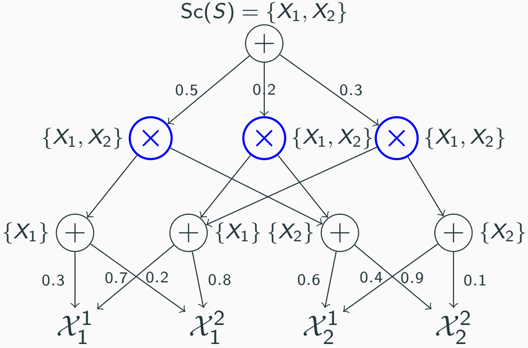


$$X = \{X_1 = 0, X_2 = 1\}$$

Decomposability

Definition 5 (Decomposability).

An SPN is decomposable iff no variable appears in more than one child of a product node (i.e. scopes are disjoint).



Decomposability vs Consistency

Decomposability implies **consistency**.

But **decomposability** is much easier for learning, and allows for an interpretation of product nodes as *independencies* between variables.

Robert Peharz et al. 2015 shows **decomposable** SPNs are as representable as solely **consistent** ones.

Learning

Learning types

Two main types of learning:

Structure learning:

Learn graph structure from data.

Parameter learning:

Learn weights from data given a fixed graph structure.

Both usually attempt to optimize log-likelihood.

Structure learning

- Structure learning:
  1. **Poon-Domingos dense architecture** (Poon and Domingos 2011);
  2. **Gens-Domingos LearnSPN** (Gens and Domingos 2013);
  3. **Dennis-Ventura clustering architecture** (Dennis and Ventura 2012);
  4. **Random Tensorized SPNs (RAT-SPNs)** (R. Peharz et al. 2018);
  5. Indirect-Direct SPNs (ID-SPNs) (Rooshenas and Lowd 2014);
  6. LearnSPN+Chow-Liu Trees (LearnSPN-BTB) (Vergari, Mauro, and Esposito 2015).

Parameter learning

- Parameter learning:
  1. **Generative Gradient Descent** (Poon and Domingos 2011; Darwiche 2003);
  2. **Discriminative Gradient Descent** (Gens and Domingos 2012);
  3. Expectation-Maximization (Poon and Domingos 2011);
  4. Extended Baum-Welch (Rashwan, Poupart, and Zhitang 2018);
  5. Collapsed Variational Inference (Zhao et al. 2016);
  6. Bayesian Moment Matching (Rashwan, Zhao, and Poupart 2016).

Parameter learning

Generative vs Discriminative Gradient Descent

Generative:

- Optimize log-likelihood of $P(X, Y)$
- Gradient: $\frac{\partial}{\partial W} \log P(X, Y)$
- E.g. completion

Discriminative:

- Optimize log-likelihood of $P(Y|X)$
- Gradient: $\frac{\partial}{\partial W} \log P(Y|X)$
- E.g. classification

Generative representation is able to extract its **discriminative**.

$$P(Y|X) = \frac{P(X, Y)}{P(Y)}$$

Derivatives I

Let S be an SPN, and W the set of weights of S. Denote by S_n the sub-SPN rooted at node n.

Objective: find gradient $\frac{\partial}{\partial W} \log S$.

That is, compute each component $\partial S / \partial w_{n,j}$, for each edge $n \to j$.

$$
\begin{aligned}
\frac{\partial S}{\partial w_{n,j}}(X) &= \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial w_{n,j}}(X) \\
&= \frac{\partial S}{\partial S_n} \frac{\partial}{\partial w_{n,j}} \left(\sum_{i \in \mathrm{Ch}(n)} w_{n,i} S_i(X) \right) \\
&= \frac{\partial S}{\partial S_n} S_j(X).
\end{aligned}
$$

We now need to find a form for derivative $\frac{\partial S}{\partial S_n}$.

Derivatives II

Let's find the sub-SPN derivative $\frac{\partial S}{\partial S_j}$. From chain rule:

$$\frac{\partial S}{\partial S_j}(X) = \sum_{n \in \mathrm{Pa}(j)} \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial S_j}(X)$$

$$= \underbrace{\sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: sum}}} \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial S_j}(X)}_{(*)} + \underbrace{\sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: product}}} \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial S_j}(X)}_{(**)}$$

Let's analyze two cases: when n is a sum node $(*)$, and when it's a product node $(**)$.

Derivatives III

Case 1: when n is a sum node.

$$
\begin{aligned}
(*) &= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n:\ \mathrm{sum}}} \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial S_j}(X) \\
&= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n:\ \mathrm{sum}}} \frac{\partial S}{\partial S_n} \frac{\partial}{\partial S_j} \left(\sum_{i \in \mathrm{Ch}(n)} w_{n,i} S_i(X) \right) \\
&= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n:\ \mathrm{sum}}} \frac{\partial S}{\partial S_n} w_{n,j}
\end{aligned}
$$

Derivatives IV

Case 2: when n is a product node.

$$
\begin{aligned}
(**) &= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: product}}} \frac{\partial S}{\partial S_n} \frac{\partial S_n}{\partial S_j}(X) \\
&= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: product}}} \frac{\partial S}{\partial S_n} \frac{\partial}{\partial S_j} \left(\prod_{i \in \mathrm{Ch}(n)} S_i(X) \right) \\
&= \sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: product}}} \frac{\partial S}{\partial S_n} \prod_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k
\end{aligned}
$$

Derivatives V

Going back to our original formula and using the derived forms of $(*)$ and $(**)$:

$$\begin{aligned}\frac{\partial S}{\partial S_j}(X) &= \sum_{\substack{n \in \mathrm{Pa}(j)\\ n:\ \text{sum}}} \frac{\partial S}{\partial S_n}\frac{\partial S_n}{\partial S_j}(X) \quad + \sum_{\substack{n \in \mathrm{Pa}(j)\\ n:\ \text{product}}} \frac{\partial S}{\partial S_n}\frac{\partial S_n}{\partial S_j}(X) \\ &= \sum_{\substack{n \in \mathrm{Pa}(j)\\ n:\ \text{sum}}} \frac{\partial S}{\partial S_n} w_{n,j} \qquad\quad + \sum_{\substack{n \in \mathrm{Pa}(j)\\ n:\ \text{product}}} \frac{\partial S}{\partial S_n} \prod_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k\end{aligned}$$

Remark: when j is the root node: $\frac{\partial S}{\partial S_j} = \frac{\partial S}{\partial S} = 1$.

The above form lends itself nicely to an algorithmic format.

Derivatives VI

Algorithm 1 `Backprop`: Backpropagation derivation on SPNs

Input A valid SPN S with pre-computed probabilities $S_n(X)$

Output Partial derivatives of S with respect to every node and weight

Initialize $\frac{\partial S}{\partial S_n} = 0$ except $\frac{\partial S}{\partial S} = 1$
for each node $n \in S$ in top-down order **do**
    if n is sum node **then**
        for all $j \in \mathrm{Ch}(n)$ **do**
            $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + w_{n,j} \frac{\partial S}{\partial S_n}$
            $\frac{\partial S}{\partial w_{n,j}} \leftarrow \frac{\partial S}{\partial S_n} S_j$
    else
        for all $j \in \mathrm{Ch}(n)$ **do**
            $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + \frac{\partial S}{\partial S_n} \prod_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k$

Generative gradient descent

Let's go back to our original objective of finding the gradient. For the **generative** (joint distribution) case:

$$
\begin{aligned}
\frac{\partial}{\partial W} \log P(X, Y) &= \frac{\partial}{\partial W} \log S(X, Y) \\
&= \frac{1}{S(X, Y)} \frac{\partial S}{\partial W}(X, Y) \propto \frac{\partial S}{\partial W}(X, Y)
\end{aligned}
$$

So it is sufficient to find $\frac{\partial S}{\partial W}$, which we already have. Our weight update is then:

$$
\Delta w_{n,j} = \eta \frac{\partial S}{\partial w_{n,j}}(X, Y)
$$

Discriminative gradient descent

For the **discriminative** (conditional distribution) case:

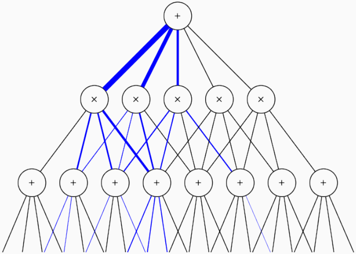
$$
\begin{aligned}
\frac{\partial}{\partial W}\log P(Y|X) &= \frac{\partial}{\partial W}\log\left(\frac{P(Y,X)}{P(X)}\right) \\
&= \frac{\partial}{\partial W}\log P(Y,X) \quad - \quad \frac{\partial}{\partial W}\log P(X) \\
&= \frac{1}{S(Y,X)}\frac{\partial}{\partial W}S(Y,X) \quad - \quad \frac{1}{S(X)}\frac{\partial}{\partial W}S(X)
\end{aligned}
$$

Weight updates will take the following form:

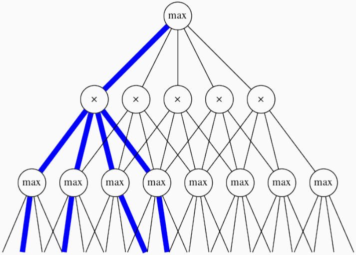
$$
\Delta w_{n,j} = \eta\left(\frac{1}{S(Y,X)}\frac{\partial S(Y,X)}{\partial w_{n,j}} - \frac{1}{S(X)}\frac{\partial S(X)}{\partial w_{n,j}}\right)
$$

Soft vs hard derivation

Results derived in the previous slides are called **soft**. Soft gradient means weight updates are derivatives of network evaluations. The **deeper** the network, the **fainter** the signal.



(a) Soft gradient



(b) Hard gradient

This is called **gradient diffusion**. A solution to this is **hard** gradient descent.

Hard derivatives I

Let S be an SPN. Instead of $\frac{\partial S}{\partial W}$, we'll derive $\frac{\partial M}{\partial W}$, where M is the Max-Product Network (MPN) of S. M can be extracted from S by replacing sums with max nodes.

Soft:

- $\partial S / \partial W$
- W is the set of weights of S
- Messages are derivatives

Hard:

- $\partial M / \partial W$
- W is the multiset of weights that a forward pass through M visits
- Messages are counts

Hard derivatives II

Objective: find gradient $\frac{\partial M}{\partial W}$

We know that $M(X) = \prod_{w_i \in W} w_i^{c_i}$, where c_i is the number of times w_i appears in W. Let's take the logarithm of M on each component:

$$
\begin{aligned}
\frac{\partial \log M}{\partial w_{n,j}}(X) &= \frac{\partial}{\partial w_{n,j}} \log \left(\prod_{w_i \in W} w_i^{c_i} \right) \\
&= \frac{1}{\prod_{w_i \in W} w_i^{c_i}} \cdot c_{n,j} w_{n,j}^{c_{n,j}-1} \cdot \prod_{w_i \in W \setminus \{w_{n,j}\}} w_i^{c_i} \\
&= c_{n,j} \frac{w_{n,j}^{c_{n,j}-1}}{w_{n,j}^{c_{n,j}}} = \frac{c_{n,j}}{w_{n,j}}
\end{aligned}
$$

Hard generative gradient descent

From our previous result, we know that:

$$\frac{\partial \log M}{\partial w_{n,j}}(X) = \frac{c_{n,j}}{w_{n,j}}$$

But that's exactly the log-likelihood for the generative case! This gives us the following weight update:

$$\Delta w_{n,j} = \eta \frac{c_{n,j}}{w_{n,j}}$$

Note how $c_{n,j}$ is an integer, and $w_{n,j} \in [0, 1]$, meaning the signal passed at learning does not depend on network size or depth, avoiding the problem of gradient diffusion.

Hard discriminative gradient descent I

For the discriminative case we want:

$$\frac{\partial}{\partial W} \log \tilde{P}(Y|X) = \frac{\partial}{\partial W} \log \left(\frac{\tilde{P}(Y,X)}{\tilde{P}(X)} \right) = \frac{\partial}{\partial W} \log \left(\frac{M(Y,X)}{M(X)} \right)$$

Where $\tilde{P}$ is the MAP. Apply chain rule:

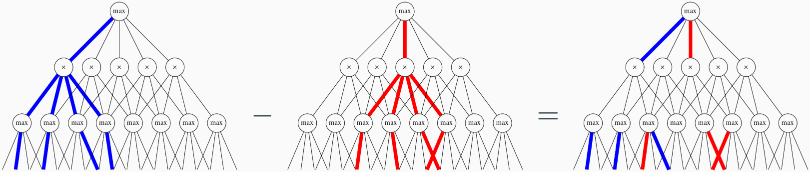
$$\frac{\partial}{\partial W} \log \left(\frac{M(Y,X)}{M(X)} \right) = \frac{\partial}{\partial W} \log M(Y,X) - \frac{\partial}{\partial W} \log M(X)$$

For each component:

$$\begin{aligned} \frac{\partial}{\partial w_{n,j}} \log \tilde{P}(Y|X) &= \frac{\partial}{\partial w_{n,j}} \log M(Y,X) - \frac{\partial}{\partial w_{n,j}} \log M(X) \\ &= \frac{\partial}{\partial w_{n,j}} c_{n,j} - \frac{\partial}{\partial w_{n,j}} \hat{c}_{n,j} \\ &= \frac{\Delta c_{n,j}}{w_{n,j}} \end{aligned}$$

Hard discriminative gradient descent II

Visually, $\Delta c_{n,j}$ is the difference between the path of evaluation $M(Y,X)$ and $M(X)$.



$\frac{\partial}{\partial W} \log M(Y,X)$ $-$ $\frac{\partial}{\partial W} \log M(X)$ $=$ $\nabla \log \tilde{P}(Y|X)$

Edges in blue represent positive values, red are negative values and uncolored edges have zero value.

Summary

Generative Gradient Descent

Inference	Weight updates
Soft	$\Delta w_{n,j} = \eta \frac{\partial S}{\partial w_{n,j}}(X, Y)$
Hard	$\Delta w_{n,j} = \eta \frac{c_{n,j}}{w_{n,j}}$

Discriminative Gradient Descent

Inference	Weight updates
Soft	$\Delta w_{n,j} = \eta \left(\frac{1}{S(Y,X)} \frac{\partial S(Y,X)}{\partial w_{n,j}} - \frac{1}{S(X)} \frac{\partial S(X)}{\partial w_{n,j}} \right)$
Hard	$\Delta w_{n,j} = \eta \frac{\Delta c_{n,j}}{w_{n,j}}$

Structure learning

Definition 6 (Dataset).

A dataset is a matrix $D \in M_{m \times n}(\mathbb{Z})$, where **rows** are **instances** and **columns** represent **variables**.

Example 7.

Let D be a 3×2 black and white image dataset, where each pixel can take a value 0 (black) or 1 (white).

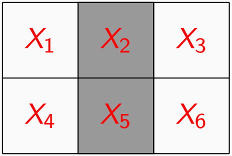
D	X_1	X_2	X_3	X_4	X_5	X_6
I_1	1	0	1	1	0	1
I_2	0	1	0	0	1	0

$\mathbf{X} = \{X_1, \ldots, X_n\}$ is the set of random variables (e.g. pixels) of D.

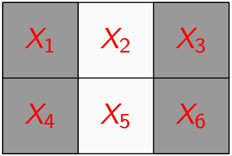
$\mathbf{I} = \{I_1, \ldots, I_m\}$ is the set of instances of D.

Preliminaries II

D	X_1	X_2	X_3	X_4	X_5	X_6
I_1	1	0	1	1	0	1
I_2	0	1	0	0	1	0



I_1



I_2

LearnSPN I

The Gens-Domingos LearnSPN schema exploits two SPN properties: **completeness** and **decomposability**.

Completeness:

Interpret sum children as clusters (similar instances).

Decomposability:

Interpret product children as independent sets (disjoint scopes).

Idea:

1. Partition by row through clustering.
2. Partition by column through variable independence tests.
3. Base case: univariate distribution.

LearnSPN II

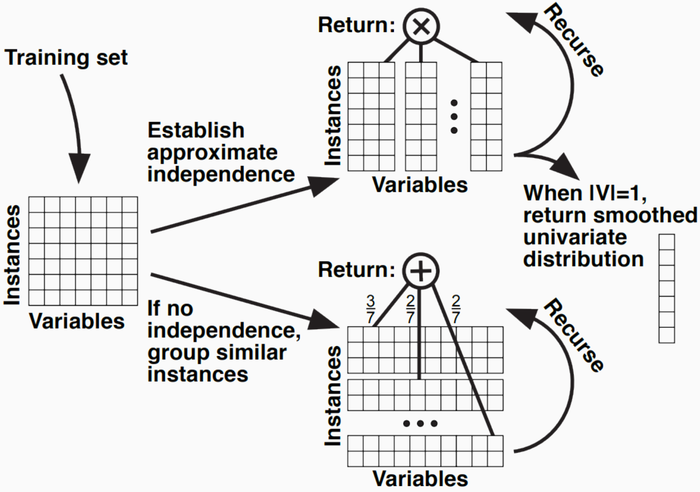
Algorithm 2 `LearnSPN`: Gens-Domingos structure learning schema

Input Set of instances I and scope X

Output SPN structure learned from I and X

1: **if** $|X| = 1$ **then**
2:     **return** univariate distribution over $I[X]$
3: Partition X into $P_1, P_2, \ldots, P_m$ st $\forall i, j,\ i \neq j,\ P_i \perp P_j$
4: **if** $m > 1$ **then**
5:     **return** $\prod_i \texttt{LearnSPN}(D, P_i)$
6: Cluster I such that $Q_1, Q_2, \ldots, Q_n$ are I's clusters
7: **return** $\sum_i \frac{|Q_i|}{|I|} \texttt{LearnSPN}(Q_i, X)$

LearnSPN III



Source: Gens and Domingos 2013

LearnSPN IV

LearnSPN is very flexible and modular.

Clustering:

k-means, *k*-mode, DBSCAN, ...

Variable independence:

G-test, χ^2 Pearson test, Mutual Information, ...

Univariate distribution:

Multivariate, gaussian, mixture of gaussians, ...

LearnSPN V

Pros:

- Flexible implementation;
- Very deep architecture even with small training size;
- Guarantees completeness and decomposability.

Cons:

- Generates only trees;
- Not as expressive as dense and deep networks;
- Tree-like structure and deepness impact inference speed.

Poon dense architecture

Idea:

1. Generate a dense architecture from data.
2. Learn weights with EM or Gradient Descent (GD).
3. Prune zero weights and redundant sub-SPNs.

Definition 8 (Region).

A region R is a

Thank you.

Questions?

References i

Darwiche, Adnan (May 2003). "A Differential Approach to Inference in Bayesian Networks". In: *J. ACM* 50.3, pp. 280–305. ISSN: 0004-5411. DOI: 10.1145/765568.765570. URL: http://doi.acm.org/10.1145/765568.765570.

Dennis, Aaron and Dan Ventura (2012). "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". In: *Advances in Neural Information Processing Systems* 25.

Gens, Robert and Pedro Domingos (2012). "Discriminative Learning of Sum-Product Networks". In: *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

- Gens, Robert and Pedro Domingos (2013). "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30.
- Peharz, R. et al. (2018). "Probabilistic Deep Learning using Random Sum-Product Networks". In: *ArXiv e-prints*.
- Peharz, Robert et al. (2015). "On Theoretical Properties of Sum-Product Networks". In: *International Conference on Artificial Intelligence and Statistics 18 (AISTATS 2015)*.
- Poon, Hoifung and Pedro Domingos (2011). "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27.

References iii

Rashwan, Abdullah, Pascal Poupart, and Chen Zhitang (2018). "Discriminative Training of Sum-Product Networks by Extended Baum-Welch". In: *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*. Vol. 72. Proceedings of Machine Learning Research, pp. 356–367.

Rashwan, Abdullah, Han Zhao, and Pascal Poupart (Sept. 2016). "Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, pp. 1469–1477. URL: http://proceedings.mlr.press/v51/rashwan16.html.

Rooshenas, Amirmohammad and Daniel Lowd (2014). "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *International Conference on Machine Learning 31 (ICML 2014)*.

Vergari, Antonio, Nicola di Mauro, and Floriana Esposito (2015). "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning". In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2015)*.

Zhao, Han et al. (20–22 Jun 2016). "Collapsed Variational Inference for Sum-Product Networks". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1310–1318. URL: http://proceedings.mlr.press/v48/zhaoa16.html.