Inference with Sum-Product Networks

Jhonatan S. Oliveira oliveira@cs.uregina.ca
Department of Computer Science
University of Regina
Regina, Canada

Editor: CS807 - Advanced Architecture - Dr. Gerhard and Trevor Tomesh

Abstract

Sum-product networks (SPNs) are a layer-wise mathematical models for learning and inference with probabilistic graphical model. The main advantage of SPNs is that a learned model, under certain conditions, is guaranteed to have tractable inference in polynomial time, besides of inference being exact. Given a SPN, inference is done by propagating up on the network. In this paper, it is shown how the propagation can be done in parallel.

Keywords: sum-product networks, resource constrain, gpu

1. The Problem

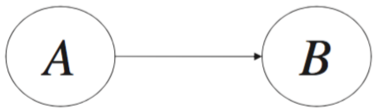
A *join probability distribution* (JPD) can be written as a polynomial. Indeed, Darwiche (2009) proposed a novel way of representing the JPD represented by a *Bayesian network* (BN) as a simple polynomial, called a *network polynomial*. Given a JPD, it is added to each configuration (row) of it variables called indicators. Each variable in a JPD has a correspondent indicator. This type of variables can only assume values zeros or ones. They are used to mark the correspondent JPD variable as being used or not. Next, the polynomial network can be obtained by summing each row of the JPD.

Example 1 *Consider the BN in Figure 1. By definition (Koller and Friedman, 2009), the JPD represented by the BN can be obtained by multiplying all conditional probability tables (CPTs). Figure 2 shows the JPD represented by the BN in Figure 1. Now, the indicator variables can be added, being one per variables in the JPD. Figure 3 illustrated the JPD with indicator variables λ_x for each JPD variable x. One can now represent the JPD by summing all terms for each row of the table. The polynomial network representing the JPD in Figure 3 is:*

$$f = \lambda_a \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{\overline{b}} \theta_a \theta_{\overline{b}|a} + \lambda_{\overline{a}} \lambda_b \theta_{\overline{a}} \theta_{b|\overline{a}} + \lambda_{\overline{a}} \lambda_{\overline{b}} \theta_{\overline{a}} \theta_{\overline{b}|\overline{a}}, \quad (1)$$

where each variable A and B can assume values in $\{a, \overline{a}\}$ and $\{b, \overline{b}\}$, respectively.

In 2011, Poon and Domingos (2011) extended the idea of the polynomial network for any unnormalized distribution. In practice, that means that any non-negative function can be represented as a polynomial network. This extension is now known as *sum-product networks* (SPNs).



A	Θ_A
true	$\theta_a = .3$
false	$\theta_{\bar{a}} = .7$

A	B	$\Theta_{B\|A}$
true	true	$\theta_{b\|a} = .1$
true	false	$\theta_{\bar{b}\|a} = .9$
false	true	$\theta_{b\|\bar{a}} = .8$
false	false	$\theta_{\bar{b}\|\bar{a}} = .2$

Figure 1: A BN defined by a DAG and a set of CPTs.

A	B	$\Pr(A, B)$
a	b	$\theta_a\theta_{b\|a}$
a	$\bar{b}$	$\theta_a\theta_{\bar{b}\|a}$
$\bar{a}$	b	$\theta_{\bar{a}}\theta_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$\theta_{\bar{a}}\theta_{\bar{b}\|\bar{a}}$

Figure 2: A JPD represented by the BN in Figure 1.

One can draw the network polynomial as a directed acyclic graph (DAG). Here, each multiplication is represented by a multiplication node, where the children are the factors involved in that particular multiplication. Sum nodes represent summing terms of the polynomial. The polynomial is written in a way that allows a pattern to form in the correspondent DAG. This pattern is a layer of sum and a layer of product. Thus, the name of the model SPNs.

Example 2 *The JPD in Figure 3 has a network polynomial described in Equation (1). This network polynomial can be draw as a DAG, as illustrated in Figure 4. Notice that multiplications in (1) are represented as a multiplication node in Figure 4 where the children are the involved factors. Similarly, sum in (1) are shown as sum nodes in Figure 4.t*

Inference in these models can be done by propagating up and down in the SPN DAG, as proposed by the *differential method* Darwiche (2003). It is out of the scope of this paper

A	B	$\Pr(A, B)$
a	b	$\lambda_a\lambda_b\theta_a\theta_{b\|a}$
a	$\bar{b}$	$\lambda_a\lambda_{\bar{b}}\theta_a\theta_{\bar{b}\|a}$
$\bar{a}$	b	$\lambda_{\bar{a}}\lambda_b\theta_{\bar{a}}\theta_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$\lambda_{\bar{a}}\lambda_{\bar{b}}\theta_{\bar{a}}\theta_{\bar{b}\|\bar{a}}$

Figure 3: The JPD from Figure 2 with indicator variables for each JPD variable.

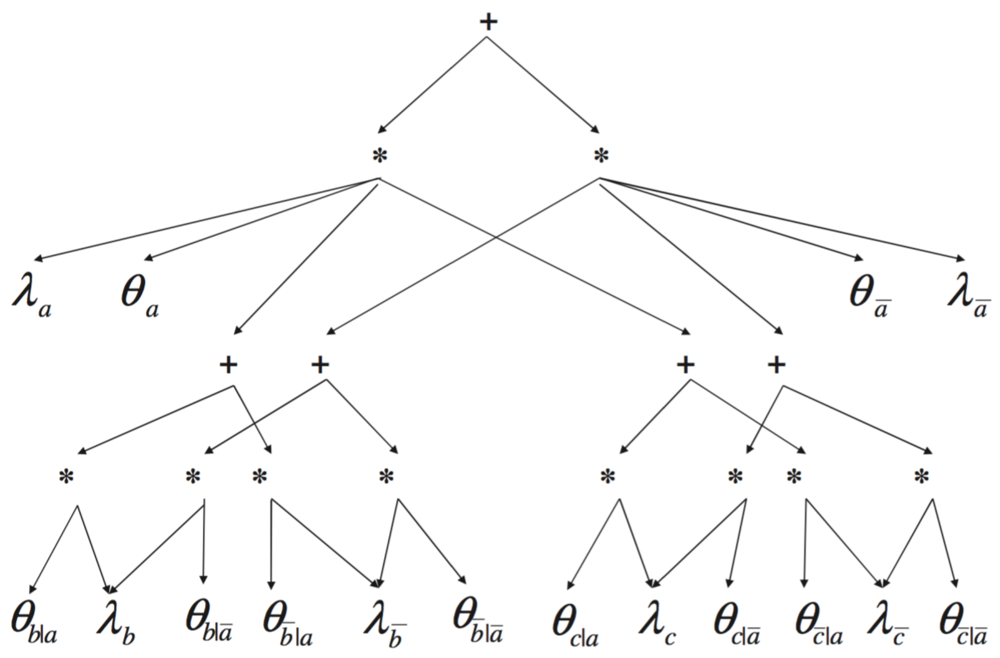

+
*
*
λ_a
θ_a
$\theta_{\overline{a}}$
$\lambda_{\overline{a}}$
$\theta_{b|a}$
λ_b
$\theta_{b|\overline{a}}$
$\theta_{\overline{b}|a}$
$\lambda_{\overline{b}}$
$\theta_{\overline{b}|\overline{a}}$
$\theta_{c|a}$
λ_c
$\theta_{c|\overline{a}}$
$\theta_{\overline{c}|a}$
$\lambda_{\overline{c}}$
$\theta_{\overline{c}|\overline{a}}$

Figure 4: A SPN representing the JPD in Figure 3.

to explain the differential approach when computing marginals for all variables of the JPD. But it is of our interest the process of propagation itself. Consider, for instance, propagating up in the DAG. Values for each node are computed by following the operators. Notice that nodes in the same layer can be computed simultaneously, since one does not require the value from the other.

Example 3 *Consider the SPN in Figure 4. When propagating up in this DAG, the first product layer is computed. Here,* $\theta_{b|a}$ *is multiplied with* λ_b*, for instance. But notice that* λ_b *can also be multiplied with* $\theta_{b|\overline{a}}$*, simultaneously.*

This paper proposes a method for parallelizing the computation at each layer of the SPN, as described in the next section.

2. A Parallel Solution

An inference algorithm can be proposed to the propagation phase in a SPN. But since they are similar to the inference process using a polynomial network, as noticed in (Peharz et al., 2015), one can use the same propagation algorithm from (Darwiche, 2009). Thus, we show in Figure 5 one possible algorithm for propagating up and down in a SPN. This algorithm was taken from (Darwiche, 2009) Algorithm 34.

Parallelizing the Algorithm in Figure 5 can be simply done by parallelizing the outer for-loop, in line 5. The main idea here is that, in the SPN of Figure 4, one can compute

Algorithm 34 `CircP1`($\mathcal{AC}$, vr(), dr()). Assumes that the values of leaf circuit nodes v have been initialized in vr(v).

input:

$\mathcal{AC}$:	arithmetic circuit
vr():	array of value registers (one register for each circuit node)
dr():	array of derivative registers (one register for each circuit node)

output: computes the value of circuit output v in vr(v) and computes derivatives of leaf nodes v in dr(v)

main:

```
1:  for each circuit node $v$ (visiting children before parents) do
2:      compute the value of node $v$ and store it in vr($v$)
3:  end for
4:  $\mathrm{dr}(v) \leftarrow 0$ for all non-root nodes $v$; $\mathrm{dr}(v) \leftarrow 1$ for root node $v$
5:  for each circuit node $v$ (visiting parents before children) do
6:      for each parent $p$ of node $v$ do
7:          if $p$ is an addition node then
8:              $\mathrm{dr}(v) \leftarrow \mathrm{dr}(v) + \mathrm{dr}(p)$
9:          else
10:             $\mathrm{dr}(v) \leftarrow \mathrm{dr}(v) + \mathrm{dr}(p) \prod_{v' \neq v} \mathrm{vr}(v')$, where $v'$ is a child of parent $p$
11:         end if
12:     end for
13: end for
```

Figure 5: Algorithm (Darwiche, 2009) for performing inference in a SPN.

the values of all nodes at each layer. For instance, the first product layer can be computed all at once. Next, the sum layer can again be computed all at once, and so on. This is only possible since each layer does not require input values from its internal nodes.

There are few concerns when parallelizing the inference process. The first one is regarding layers that intersect the DAG later on and not from the top. For instance, consider the SPN in Figure 4. The layer involving nodes λ_a and θ_a has to wait for the other children of its parent. This can be safely done by imposing a limitation at each layer: only compute its values when the previously layer is done. The second possible issue is regarding shared variables within each layer. For instance, variable λ_b is involves in two multiplication at the bottom layer. Since we are only considering SPNs with real value nodes, this can be safely computed by creating a copy of that shared node value. In case where nodes are also functions, a more sophisticated solution is required. But this issue is out of the scope of this paper.

3. Conclusions

SPNs are layer-wise models used to represent JPDs in a compact and graphical way. Inference in a SPN can be done by propagating up and down on the SPN DAG. When propagating in the tree, operations in the same layer can be computed simultaneously, since they do not require input from within the same layer. In this way, one can perform inference in SPN hopefully faster. Future works will implement the proposed parallel method to verify improvements.

References

Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 1 edition, 2009.

Daphne Koller and Nir Friedman. Probabilistic Graphical Models - Principles and Techniques. *MIT Press 2009*, 2009.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro M Domingos. On Theoretical Properties of Sum-Product Networks. *AISTATS 2015*, pages 744–752, 2015.

Hoifung Poon and Pedro M Domingos. Sum-Product Networks: A New Deep Architecture. *UAI*, pages 337–346, 2011.