An Introduction to Sum-Product Networks

A collection of studies on properties, structure, inference and learning on Sum-Product Networks

Student: Renato Lui Geh

Supervisor: Denis Deratani Mauá (DCC IME-USP)

University of São Paulo / Universidade de São Paulo (USP)

Institute of Mathematics and Statistics / Instituto de Matemática e Estatística (IME)

Abstract

This work is a collection of ongoing studies I am working on for my undergraduate research project on automatic learning of Sum-Product Networks. The main objective of this work is logging my study notes on this subject in an instructive and uncomplicated way. Most scientific papers are cluttered with intricate names and require extensive background on the subject in order for the reader to understand what is going on. In this paper we seek to provide an easy reference and introductory reading material to those who intend to work with Sum-Product Networks.

This study is divided into five main sections. We start with an introductory section regarding probabilistic graphical models and why Sum-Product Networks are so interesting. Next we talk about the structure of the model. Then we analyse some properties and theorems. After that, we look on how to perform exact tractable inference. And finally we take a look at how to perform learning.

At the end of this work I include a list of subjects I intend to study. They may include both new things as well as already included subjects that I plan on studying further. Since this is a work in progress (and in fact will be for the entirety of my undergraduate studies), some sections (or subsections) will be incomplete (or otherwise just not as thoroughly detailed). This means new sections and subsections may be added in the (near) future.

Contents

1 Introduction

In this section we show what the usual problems with probabilistic graphical models are and what led to the creation of Sum-Product Networks. Additionally, we show some results from experiments Poon and Domingos performed on the inaugural Sum-Product Network article *Sum-Product Networks: A New Deep Architecture* [PD11].

1.1 Motivation

Probabilistic Graphical Models (PGMs) perform inference through posterior probabilities on the query and evidence. Thus, inference would look like this:

$$P(X|\mathbf{e} = e_1, \ldots, e_q)$$

Where X is called the variable query and $\mathbf{e}$ the evidence, that is, the observed instances of the variables.

Using the definition of conditional probability,

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

We get the following equation:

$$P(X|\mathbf{e}) = \frac{P(X, \mathbf{e})}{P(\mathbf{e})} = \alpha P(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} P(X, \mathbf{e}, \mathbf{y}) \tag{1}$$

Where $\mathbf{y}$ is a hidden variable. That is, let $\mathbf{X}$ be the complete set of variables. Then $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$, where X is the query, $\mathbf{E}$ is the set of evidence variables and $\mathbf{Y}$ is a set of non-query non-evidence variables. Then $\mathbf{y}$ is an instance of $\mathbf{Y}$.

We can see that $P(X, \mathbf{e}, \mathbf{y})$ is actually a subset of the full joint distribution. Since we are summing out the hidden variables, we are actually discarding all the possible values of $\mathbf{y}$ and taking into account all the possibilities where the query given the evidence occur.

Now consider a Bayesian network as the PGM of our choice. We know that Bayesian networks have the property of representing the full joint distribution as a product of conditional probabilities:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|Par(X_i)) \tag{2}$$

Where $Par(X_i)$ are the values of the parents of X_i. From this property we know that we can now compute inference by applying Equation (2) on Equation (1). By doing that we get inference by computing the sum of products of conditional probabilities from the network. This is fundamental to Adnan Darwiche's *network polynomial* [Dar03; Dar09], a concept that is the core of Sum-Product Networks.

We know that we can compute inference by summing out the hidden variables and then multiplying the remaining factors, but this process relies on adding and then multiplying an exponential number of probabilities. In fact, if we don't take the order of the terms in the summation into account, the complexity reaches $O(np^n)$, where p is the number of possible values

a variable may take. If we move the independent terms from the summation the complexity is then $O(p^n)$. This is obviously intractable, and a reason why approximate inference is often the best solution.

Bayesian networks are not the only model that have intractable exact inference. Most PGMs suffer from intractability of inference, and hence intractability of learning. However Domingos and Poon argument that "classes of graphical models where inference is tractable exist [...], but are quite limited in the distributions they can represent compactly." [PD11].

Sum-Product Networks provide a graphical model where inference is both tractable and exact whilst still being more general than existing tractable models.

1.2 Background

In Adnan Darwiche's *A Differential Approach to Inference in Bayesian Networks* [Dar03] and *Modeling and Reasoning with Bayesian Networks* [Dar09], Darwiche presents a new way of representing full joint distributions through a *network polynomial*. In this subsection we will show what a network polynomial is.

Consider the following Bayesian network:

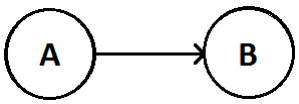

Figure 1: A Bayesian network $A \to B$, that is, the variable B depends on A.

A	Θ_A
a	$\theta_a = 0.3$
$\overline{a}$	$\theta_{\overline{a}} = 0.7$

A	B	$\Theta_{B\|A}$
a	b	$\theta_{b\|a} = 0.1$
a	$\overline{b}$	$\theta_{\overline{b}\|a} = 0.9$
$\overline{a}$	b	$\theta_{b\|\overline{a}} = 0.8$
$\overline{a}$	$\overline{b}$	$\theta_{\overline{b}\|\overline{a}} = 0.2$

Tables 1 and 2

The two tables above describe the Bayesian network in Figure 1. From these tables we can construct the full joint distribution table by applying the definition of conditional probability we saw in the previous subsection.

A	B	$P(A, B)$
a	b	$\theta_a \theta_{b\|a}$
a	$\overline{b}$	$\theta_a \theta_{\overline{b}\|a}$
$\overline{a}$	b	$\theta_{\overline{a}} \theta_{b\|\overline{a}}$
$\overline{a}$	$\overline{b}$	$\theta_{\overline{a}} \theta_{\overline{b}\|\overline{a}}$

Table 3 The full joint distribution of the Bayesian network in Figure 1.

Before we proceed we need to understand the concept of variable indicators and their consistency with their respective variables. We will take Boolean variables for simplicity as example. An indicator of a variable, usually written as $[\cdot]$ has a value of 1 if the variable is true and value 0 otherwise. Since $[X_i]$ is quite cumbersome, we abbreviate $[X_i]$ as x_i and $[\overline{X}_i]$ as $\overline{x}_i$. See [Appendix A] with regards to conflicting notations. Let us now define the notation of indicators more formally.

Definition. *Let* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *be the set of all variables in Bayesian network* $\mathcal{N}$*, with each variable* X_i *having* p *possible values. Then the indicators of an arbitrary variable* X_i *are denoted*

by x_i^j, $1 \leq j \leq p$ where j is the j-th possible value of the variable. All indicators follow the consistency rule.

For $p = 2$, we can abbreviate x_i^1 and x_i^2 to x_i and $\overline{x}_i$ and consider Boolean values. Let us now define consistency. We will define consistency for Boolean values. The extension to multi-valued variables is not discussed here.

Definition. *Let $\mathbf{x} = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ be the set of indicators of all variables in the set $\mathbf{X} = \{X_1, \ldots, X_n\}$, where variable X_i may have values 1 or 0, with x_i representing 1 and $\overline{x}_i$ otherwise. Let $\mathbf{e} = \{e_p, \ldots, e_q\}$ be the set of an observed event as evidence where e_j represents an observed value for variable X_j in set $\mathbf{X}$. Then the set $\mathbf{X}$ is consistent with $\mathbf{e}$ iff:*

- *For each variable X_i in set $\mathbf{X}$:*
  - *If there exists an observed value e_i in $\mathbf{e}$, then:*
    - *If $e_i = 1$, then $x_i = 1$ and $\overline{x}_i = 0$.*
    - *If $e_i = 0$, then $x_i = 0$ and $\overline{x}_i = 1$.*
  - *If there is no observed value e_i for X_i, then:*
    - *$x_i = 1$ and $\overline{x}_i = 1$.*

We now introduce Darwiche's network polynomial. Darwiche, in his article and book [Dar03; Dar09], uses indicators as λ_i and $\lambda_{\overline{i}}$ instead of x_i and $\overline{x}_i$. However, they are exactly the same as we have defined here. Since we named our variables A and B earlier, we will use Darwiche's notation for just this example, since it's more readable. For other examples we will use our own notation.

Table 3 is the full joint distribution that represents the Bayesian network in Figure 1. Darwiche proposes a compact way to represent such distribution by taking each term in the joint distribution, multiplying the relevant indicators, and then summing all terms into a polynomial function. This function is named the network polynomial (Equation 3) of the Bayesian network.

$$\Phi = \lambda_a \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{\overline{b}} \theta_a \theta_{\overline{b}|a} + \lambda_{\overline{a}} \lambda_b \theta_{\overline{a}} \theta_{b|\overline{a}} + \lambda_{\overline{a}} \lambda_{\overline{b}} \theta_{\overline{a}} \theta_{\overline{b}|\overline{a}} \tag{3}$$

To compute the probability of any evidence $\mathbf{e}$, we compute $\Phi(\mathbf{e})$ such that all indicators are consistent with $\mathbf{e}$. If we assume that the indicators will always be consistent with the evidence, then $\Phi(\mathbf{e})$ becomes the partition function when $\mathbf{e} = \emptyset$. This is true because, if $\mathbf{e} = \emptyset$, then all indicators must be set to 1. Therefore, the value of $\Phi(\mathbf{e})$ must be the highest possible value the function may take. The partition function normalizes an unnormalized distribution.

Now that we know what a network polynomial is, we may start our study on Sum-Product Networks. The next subsection focuses on some experiments and achievements Poon and Domingos performed on [PD11]. The next section introduces Sum-Product Networks.

1.3 Experiments

In this subsection we take a brief look at some of the results Poon and Domingos worked on on their article *Sum-Product Networks: A New Deep Architecure* [PD11].

Conducting experiments on two sets of image, Caltech-101 and the Olivetti face dataset, Poon and Domingos achieved astounding results. With the lowest mean squared error compared to Deep Boltzmann (DBMs), Deep Belief Networks (DBN), Principal Component Analysis (PCA) and Nearest Neighbour (NN), Sum-Product Networks (SPNs) outperformed all the other models.

Learning Caltech faces with an SPN took 6 minutes with 20 CPUs, learning for DBMs/DBNs ranged from 30 hours to over a week.[PD11]

"For inference, SPNs took less than a second to find the MPE completion of an image, to compute the likelihood of such a completion, or to compute the marginal probability of a variable [...]." When it comes to the other models: "estimating likelihood is a very challenging problem; estimating marginals requires many Gibbs sampling steps that may take minutes of even hours, and the results are approximate without guarantee on the quality." [PD11]

Figure 2: Image completion output from SPNs. Source: `http://spn.cs.washington.edu/spn/`.

Poon and Domingos also conducted preliminary experiments on object recognition. They ran classification on three classes (one vs the other two) against convolutional DBNs (CDBNs). SPNs showed almost flawless results.

Architecture	Faces	Motorbikes	Cars
SPN	99%	99%	98%
CDBN	95%	81%	87%

Table 4 Comparison between CDBN and SPN on object recognition.

More information can be found on Poon and Domingos' *Sum-Product Networks: A New Deep Architecture* [PD11] and Gens and Domingos' *Learning the Structure of Sum-Product Networks* [GD13].

2 Structure of Sum-Product Networks

In this section we work on the ideas presented in the previous sections. We will base this section on the works of Poon and Domingos on their article *Sum-Product Networks: A New Deep Architecture* [PD11] and also on Gens and Domingos' *Learning the Structure of Sum-Product Networks* [GD13].

2.1 Partition function

Probabilistic graphical models represent distributions compactly through a normalized product of factors.

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}})$$

This equation states that the probability of X taking value x is the product of all factors, with each factor taking a subset of the variables, divided by a partition function. The partition function is the result of all the product of factors summed out.

$$Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$$

However, this form cannot represent all distributions. Additionally, inference is exponential in scope size in the worst-case and learning takes sample size and time exponential. This is because the partition function Z is the sum of an exponential number of terms, and since all marginals are sums of these terms, if Z is tractable, then marginals are also tractable.

In Domingos and Poon [PD11], they claim that, since "Z is computed using only two types of operations sums and products", then "it can be computed efficiently if $\sum_{x \in \mathcal{X}} \prod_k \phi_k(k_{\{k\}})$ can be reorganized using the distributive law into a computation involving only a polynomial number of sums and products." From this idea, they elaborate on Darwiche's network polynomial and introduce Sum-Product Networks, a representation that seeks to admit an efficient form for Z whilst still being general.

2.2 Definition

Now we define a sum-product network formally.

Definition. *A sum-product network (SPN) is an acyclic digraph. An SPN S over variables $X_1, \ldots, X_n$ has leaves as indicators $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ and internal nodes as sums and nodes in alternating layers. For each edge (i, j) that emanates from a sum node i, there exists a non-zero weight w_{ij}. The value of a sum node i is $\sum_{j \in Ch(i)} w_{ij} v_j$, where $Ch(i)$ is the set of children of i and v_j is the value of the node j. The value of a product node i is $\prod_{j \in Ch(i)} v_j$. The value of an SPN is the value of its root.*

An SPN S as a function of the indicator variables $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ will be denoted by $S(x_1, \ldots, x_n, \overline{x}_1, \ldots, x_n)$. A complete state x means that, for each variable X_i, there are two possible outcomes for the indicators: either $x_i = 1, \overline{x}_i = 0$ or $x_i = 0, \overline{x}_i = 1$. When an SPN S has a complete state x, instead of enumerating each indicator, we simply abbreviate to $S(x)$. $S(e)$ will be used to denote when the indicators are specified according to the evidence e.

A sum-product network is an elaboration on Darwiche's network polynomial. We can look at an SPN as a way to graphically represent the network polynomial of the distribution.

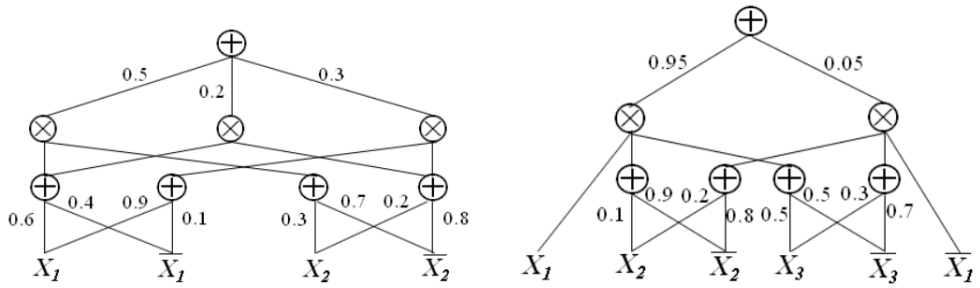

Figure 3: Two sum-product networks. The one on the left implements a naive Bayes mixture model and the other implements a junction tree. Note that the graph is shown as undirected, but it is in fact directed. All edges go from top to bottom. This is a choice of notation and may vary. Source: [PD11].

The partition function is the value of the network polynomial when all indicators are set to 1. That is, $Z = \Phi(\mathbf{e})$ when $\mathbf{e} = \emptyset$. If an SPN S has all indicators set to 1, we refer to it as $S(*)$. As we will see later, if an SPN S is valid, then $S(*) = Z_S$.

The probability of evidence, $P(e) = \Phi(e)/Z$, is linear in the size of the network polynomial, which in turn is exponential in the number of variables. However, it is possible, with a sum-product network, to represent and evaluate the probability of evidence in space and time polynomial.

SPNs are defined recursively, since the subnetwork at an arbitrary node n in an SPN is an SPN. We will denote an SPN rooted at node n as $S_n(\cdot)$. Let $\mathbf{X}$ be the set of variables in an SPN S, then for all $x \in \mathbf{X}$ the values of $S(x)$ define an unnormalized probability distribution over $\mathbf{X}$. The unnormalized probability of evidence e of an SPN S is given by $\Phi_S(e) = \sum_{x \in e} S(x)$, where the sum is over states consistent with e. The Z partition function of a distribution defined by $S(x)$ is $Z_S = \sum_{x \in \mathbf{X}} S(x)$. The scope of an SPN S is the set of variables that appear in S. A variable X_i is negated in S if $\overline{x}_i$ is a leaf and non-negated if x_i is a leaf.

The normalized probability distribution of an SPN S_n is given by $P(x) = S_n(x)/Z_n$.

Taking the left SPN in Figure 1 as an example, we have the value of the SPN as:

$$\begin{aligned} S(x_1, x_2, \overline{x}_1, \overline{x}_2) = \quad & 0.5(0.6x_1 + 0.4\overline{x}_1)(0.3x_2 + 0.7\overline{x}_2) + \\ & + 0.2(0.6x_1 + 0.4\overline{x}_1)(0.2x_2 + 0.8\overline{x}_2) + \\ & + 0.3(0.9x_1 + 0.1\overline{x}_1)(0.2x_2 + 0.8\overline{x}_2) \end{aligned}$$

The network polynomial is then:

$$\Phi = (0.5 \times 0.6 \times 0.3 + 0.2 \times 0.6 \times 0.2 + 0.3 \times 0.9 \times 0.2)x_1x_2 + \ldots$$

If we have a complete state x as $X_1 = 1, X_2 = 0$ then $S(x) = S(x_1, x_2, \overline{x}_1, \overline{x}_2) = S(1, 0, 0, 1)$. If the evidence e is $X_1 = 1$, then, following the consistency rule, we have that $S(e) = S(1, 1, 0, 1)$. And as we have seen before, $S(*) = S(1, 1, 1, 1)$.

3 Properties of Sum-Product Networks

We will now discuss a few properties described in Poon and Domingos' *Sum-Product Networks: A New Deep Architecture* [PD11] and Gens and Domingos' *Learning the Structure of Sum-Product Networks* [GD13].

We first introduce two properties: completeness and consistency. Then we define validity, a property that guarantees exact inference in time linear to the size of the SPN's edges. After that we discuss tractability of the partition function, decomposability and a few other properties.

3.1 Completeness and consistency

Let us define the two properties:

Definition. *A sum-product network is complete iff all children of the same sum node have the same scope.*

What this definition says is that, the children of a sum node must all be from the same variable. The trivial case is when all the children of the sum node i are leaves. If S_i is complete, then all leaves are indicators of a same variable. This is easily extended to the general case, since if, given a sum node j, all $Ch(j)$ are complete then all $S_k \in Ch(j)$ are complete and thus S_j is complete. If there exists an $S_k \in Ch(j)$ that is incomplete, then S_j is incomplete.

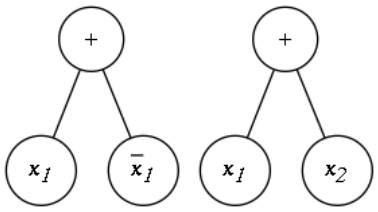

Figure 4: On the left a complete SPN. On the right an incomplete SPN.

Definition. *A sum-product network is consistent iff no variable appears negated in one child of a product node and non-negated in another.*

This means that product nodes take different variables and multiply them together. A variable X_i is negated if there exists a leaf $\overline{x}_i$ on the SPN. It is non-negated if there exists a leaf x_i. Therefore, if an SPN S_k has a variable X_i and is consistent, then for all children of S_k, the presence of x_i and $\overline{x}_i$ in more than one children of S_k is forbidden. That leads to the fact that if an SPN S_p has an inconsistent sub-SPN S_q, then S_p is inconsistent.

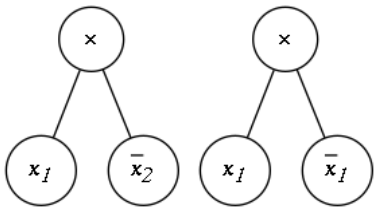

Figure 5: On the left a consistent SPN. On the right an inconsistent SPN.

If an SPN is consistent but incomplete, it does not include monomials that should be present in the network polynomial and $S(e) \leq \Phi_S(e)$. If the SPN is complete but inconsistent, the expansion will include more monomials than present in the network polynomial and thus $S(e) \geq \Phi_S(e)$. As we will see later, this means that if the SPN is inconsistent and/or incomplete inference will be approximate.

3.2 Validity

Validity is an important characteristic of SPNs. Not all SPNs are valid, but validity guarantees that inference will be exact and tractable.

Definition. *A sum-product network S is valid iff $S(e) = \Phi_S(e)$ for all evidence e.*

This means that an SPN computes the probability of evidence correctly only if it is valid. Another important aspect of validity is that, if an SPN S is valid then $S(*) = Z_S$. Validity guarantees that the probability of evidence will be computed in time linear to the SPN's size.

We will now cite the first theorem for SPNs as written in Poon and Domingos' *Sum-Product Networks: A New Deep Architecture* [PD11]. See the original article for the proof.

Theorem. *A sum-product network is valid if it is complete and consistent.*

Note that completeness and consistency are not necessary for the SPN to be valid. There are SPNs that satisfy $\Phi_S(e) = \sum_{x \in e} S(x)$ for all evidence e, but if an SPN S is both complete and consistent, then all sub-SPNs of S are valid.

As we have seen on the previous subsection invalid SPNs can be used to compute approximate inference.

3.3 Other properties

Definition. *An unnormalized probability distribution $\Phi(x)$ is representable by a sum-product network S iff $\Phi(x) = S(x)$ for all states x and S is valid.*

This states that in all possible states an SPN S will compute the probability correctly, including the partition function.

Theorem. *The partition function of a Markow network $\Phi(x)$, where x is an n-dimensional vector, can be computed in time polynomial in n if $\Phi(x)$ is representable by a sum-product network with a number of edges polynomial in n.*

Proof. Follows immediately from the definitions of SPN and representability. □

Definition. *A sum-product network is decomposable iff no variable appears in more than one child of a product node.*

What this definition means is that SPNs are more general than other representations since it does not require the distribution to be decomposable, unlike arithmetic circuits, mixture models, junction trees and others.

If for every sum node i in the SPN S, $\sum_{j \in Ch(i)} w_{ij} = 1$, then $Z_S = 1$ and $P(x) = S_n(x)$. In this case, for each sum node i, we can view i as summing out an implicit hidden variable Y_i whose values correspond to $Ch(i)$, since summing out a variable means setting all the indicators concerning the variable to 1, and children of product nodes with values 1 can be omitted. If i has no parents, then the children's weights are the prior distribution $w_{ij} = P(Y_i = j)$. If i has parents, then $w_{ij} = P(Y_i = j|\pi_i)$, where π_i is the condition that "on at least one path from Y_i to the root, all of Y_i's ancestors have the values that lead to Y_i (the ancestors being the hidden variables corresponding to the sum nodes on the path)." [PD11] If the SPN is also decomposable then the sub-SPN rooted at the j-th child represents the distribution of the variables in it conditioned on $Y_i = j$. See [PD11] for more information.

3.4 Multi-valued discrete variables

For multi-valued variables, we can simply replace the Boolean indicators $[X_i = 1]$,$[X_i = 0]$ with indicators for the possible values the variable may take x_i^j: $[X_i = x_i^1], \ldots, [X_i = x_i^m]$. As an abbreviation we may also indicate the indicators as $x_i^1, \ldots, x_i^m$. The multinomial distribution over X_i would then be written as $\sum_{j=1}^{m} p_i^j x_i^j$, with $p_i^j = P(X_i = x_i^j)$.

3.5 Continuous variables

We can construct SPNs with continuous variables by assuming variables have an infinite number of values. The sum of the indicators $\sum_{j=1}^{m} p_i^j x_i^j$ can be viewed as an integration of the densities $\int p(x)dx$ with $p(x)$ as the probability density function of X. We will not go into details on this subject in this paper[1]. For more information see [PD11].

[1] I plan on elaborating more on this later as my study on SPNs progresses.

4 Inference on Sum-Product Networks

As we have seen, inference on SPNs is tractable. In fact, we can compute marginals and the MAP in time linear to the SPN's edges. In this section we will first take a look on how to compute the probability of a complete state, then we see how to find the marginals of an SPN. And finally we show that computing the MPE is possible my maximizing the nodes. We assume that we only work.

4.1 Probability of a complete state

Computing the probability of a complete state $\mathbf{X}$ is simple. First we set all indicators to be consistent with the state. For instance, let $\mathbf{X} = \{X_1 = 1, X_2 = 0\}$ be the complete state. Then the indicators are $(x_1, x_2, \overline{x}_1, \overline{x}_2) = (1, 0, 0, 1)$. Computing the probability of $\mathbf{X}$ is then $P(\mathbf{X} = \{X_1 = 1, X_2 = 0\}) = S(\mathbf{X})/Z$. Assuming our SPN S is valid and therefore $S(*) = Z_S$, and since for each sum node i: $\sum_{j \in Ch(i)} w_{ij} = 1 = Z_S$, our probability is then $P(\mathbf{X}) = S(\mathbf{X})$ and thus computing the value of the SPN is sufficient.

To compute the value of the SPN we do a bottom-up pass. A bottom-up pass consists of computing all nodes starting from the bottom and then going up. The trivial case are the leaves. The values of the leaves are the values of the indicators. In our example, the values of the leaves are $(x_1, x_2, \overline{x}_1, \overline{x}_2) = (1, 0, 0, 1)$. Consider the SPN on Figure 6. The next nodes to be computed are the sum nodes. The values of the sum nodes are, for each node i: $\sum_{j \in Ch(i)} v_j w_{ij}$. After that we go up a layer again, which in our case means computing the product nodes: $\prod_{j \in Ch(i)} v_j$, where i is the product node. Finally, once we have the values for our product nodes we may compute the value of the root node, which is a sum node. The resulting root node is then the probability of the complete state $\mathbf{X}$ we wanted to find.

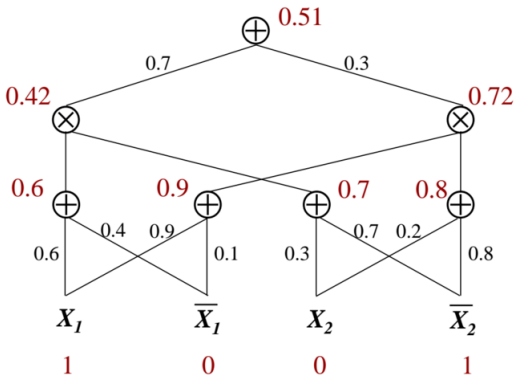

Figure 6: The value of the probability is $P(X_1 = 1, X_2 = 0) = S(1, 0, 0, 1) = 0.51$. Source: `http://spn.cs.washington.edu/talks/spn11.pdf`.

4.2 Marginals

Finding the marginals is similar to computing the probability of a complete state. However, for marginals we have unobserved variables. In this case we need to apply the consistency rule and, for each unobserved variable, have all its indicators set to 1.

Take the evidence $\mathbf{e} = \{X_1 = 1\}$ as example. Applying the consistency rule to the indicators on our SPN on Figure 7, we have the indicators as $(x_1, x_2, \overline{x}_1, \overline{x}_2) = (1, 1, 0, 1)$. Then, similar to computing the complete state, we do a bottom-up pass on our SPN, computing the value of each node. Since we assume the SPN is valid and $Z_S = 1$, the resulting marginal is $P(\mathbf{e} = \{X_1 = 1\}) = S(\mathbf{e})/Z = S(\mathbf{e}) = 0.69$.

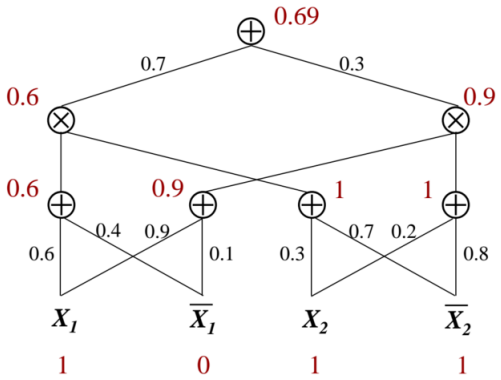

Figure 7: The value of the marginal is $P(X_1 = 1) = S(1, 1, 0, 1) = 0.69$. Source: `http://spn.cs.washington.edu/talks/spn11.pdf`.

4.3 Most probable explanation (MPE)

Computing the MPE of the SPN is similar to computing the MPE on Darwiche's Arithmetic Circuits [Dar03]: replace all sum nodes with max nodes, whose value is defined by $\max_{j \in Ch(i)} w_{ij} v_j$, where i is the max node; do a bottom-up pass to compute all values of each node and finally do a top-down pass to find the children with max values.

Let us take the evidence $\mathbf{e} = \{X_1 = 1\}$ as an example and compute the MPE state $\arg\max_X P(X|\mathbf{e})$ of the SPN on Figure 8. The indicators given the evidence $\mathbf{e}$ are $(x_1, x_2, \overline{x}_1, \overline{x}_2) = (1, 1, 0, 1)$. Again, we assume that the SPN is valid and that $Z_S = 1$.

First we replace all sum nodes with max nodes. From there we do a bottom-up pass to find all values of each node. Now, instead of computing the weighted sum on each sum node, we output the max value weighted on each edge amongst all children on each max node. Since the root of the SPN is now a max node, the value of the SPN will be the maximum value of all children.

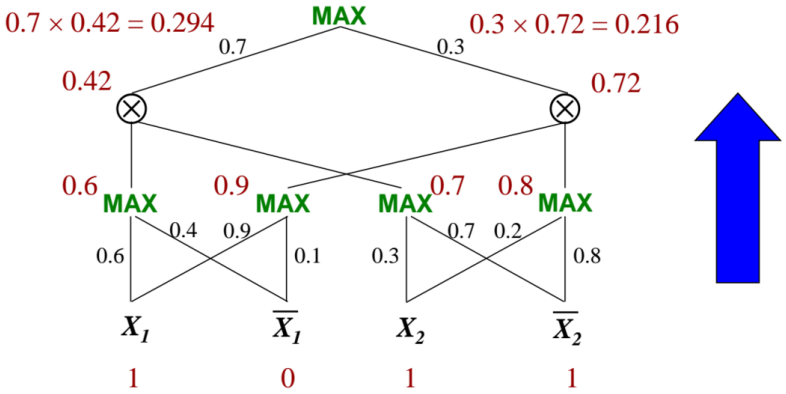

Figure 8: A bottom-up pass computes the maximum value a child of the root node may take according to the arguments, that is, this pass computes $\max P(X|\mathbf{e})$. Source: `http://spn.cs.washington.edu/talks/spn11.pdf`.

Now we need to get the max variable. We do this by doing a top-down pass and picking the child with the highest value for each node. Once we reach the leaves we have the $\arg\max_X P(X|\mathbf{e})$.

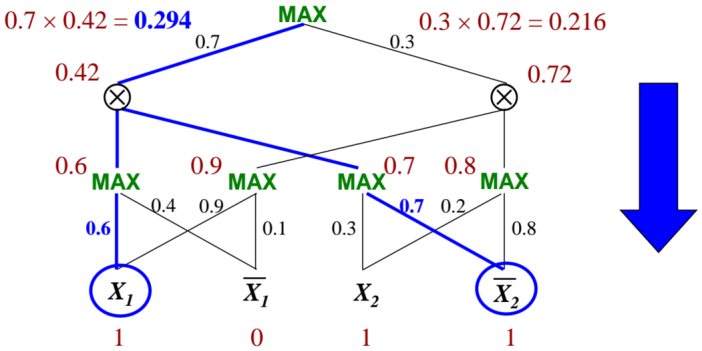

Figure 9: A top-down pass selects the child with the highest value for each node, resulting on $\arg\max_X P(X|\mathbf{e})$. Source: `http://spn.cs.washington.edu/talks/spn11.pdf`.

By extension of Darwiche's findings on [Dar03] this procedure will find the MPE state successfully if the SPN is decomposable or consistent.

5 Learning Sum-Product Networks

In this section we are going to show some algorithms on learning Sum-Product Networks.

Take note that this is an ongoing work, since this paper tries to show my progress throughout my undergraduate research project. That said, some subsections may not be complete or thoroughly detailed.

5.1 Learning the weights

This method for learning an SPN was proposed by Poon and Domingos on their article *Sum-Product Networks: A New Deep Architecture* [PD11].

The idea for this learning algorithm is simple: take a valid, dense, and generic SPN, run inference on it and update weights until convergence and then prune edges that have weight zero and remove non-root nodes that have no parents. Note that we do not learn the structure of the SPN, but learn the weights. Algorithm 1, from Poon and Domingos [PD11] describes the learning method.

Algorithm 1 LearnSPN

Input Set **D** of instances over variables **X**.
Output An SPN with learned structure and parameters.

```
1: S ←GenerateDenseSPN(X)
2: InitializeWeights(S)
3: repeat
4:     for all d ∈ D do
5:         UpdateWeights(S, Inference(S, d))
6:     end for
7: until convergence
8: S ←PruneZeroWeights(S)
9: return S
```

Although the idea illustrated in Algorithm 1 is simple, creating the starting SPN is not that simple. The SPN must be complete and consistent to be flexible and correct when representing the training set. Poon and Domingos suggest a general scheme for producing the initial SPN:

1. Select a set of subsects of the variables.
2. For each subset R:
   - 2.1. Create k sum nodes $S_1^R, \ldots, S_k^R$.
   - 2.2. Select a set of ways to decompose R into other selected subsets $R_1, \ldots, R_l$.
   - 2.3. For each of these decompositions:
     - 2.3.1. For all $1 \leq i_1, \ldots, i_l \leq k$:
       - 2.3.1.1. Create a product node with parents S_j^R and children $S_{i_1}^{R_1}, \ldots, S_{i_l}^{R_l}$.
     - 2.3.2. EndFor
   - 2.4. EndFor
3. EndFor

If we assume that we always select a polynomial number of subsets of the variables, and for each of these subsets we only choose a polynomial number of decompositions, then the SPN is guaranteed to have a polynomial size and therefore is efficient during the learning and inference passes after we run Algorithm 1.

After we create the initial SPN, we now need to initialize the weights. It is evident that we need not concern too much on the values of these initializations, since on the next steps we will be performing weight updating. It is a good idea to have all weights of a same sum node initialized uniformally to avoid bias.

For weight updating, we may either use Gradient Descent or Expectation-Maximization.

For gradient descent, Computing the partial derivative of the SPN S with respect to the weights can be done by computing the inference of the marginal. Let n_j be the child of the sum node n_i. Then:

$$\frac{\partial S(x)}{\partial w_{ij}} = \frac{\partial S(x)}{\partial S_i(x)} S_j(x)$$

This can be computed by using marginal inference. Weights are then updated by a gradient step. After each update, $S(*) = 1$ can be ensured by renormalizing each sum-node's edges at each step.

6 Future work and studies

This section enumerates things to be added to this paper, similar to a *to do* list. There is no order to the list.

- How to perform EM on SPNs by learning weights according to Poon and Domingos on *Sum-Product Networks: A New Deep Architecture* [PD11].
- Learning the structure of an SPN according to *Learning the Structure of Sum-Product Networks* [GD13] by Gens and Domingos.
- Learning the structure of an SPN through greedy search, basing on the work of Dennis and Ventura in *Greedy Structure Search for Sum-Product Networks* [DV15].
- Learning the SPN architecture through clustering variables, as Dennis and Ventura propose on their *Learning the Architecture of Sum-Product Networks Using Cluster on Variables* [DV12].
- Learning the structure of non-parametric Bayesian Sum-Product Networks, as seen on Lee, Watkins and Zhang's *Non-Parametric Bayesian Sum-Product Networks* [LWZ14].
- Study and add ID-SPN to this paper based on Rooshenas and Lowd's *Learning Sum-Product Networks with Direct and Indirect Variable Interactions* [RL14] and try to understand how their implementation on Libra [LR15] works.
- Write about discriminative and generative learning on SPNs.
- Study other publications on SPN available at [Dep].
- Do a comparison on each learning method.
- Implement a learning method.

A Notation

In this section we show the notations we use throughout this paper.

A.1 Letters

We use an uppercase letter to denote a variable. A lowercase letter denotes an instance of a variable. A bold fonted letter is a set. A bold fonted uppercase letter is a set of variables. For instance:

$$\mathbf{X} = \{X_1 = x_1, \ldots, X_n = x_n\}$$

When dealing with indicator variables, the indicator function will be abbreviated as a lowercase letter. That is, $[X_i]$ will be abbreviated to x_i. Similarly, $[\overline{X}_i]$ will be written as $\overline{x}_i$. This clearly contradicts our first notation rule, however the two meanings will be clear from context and therefore conflicts will never occur.

A.2 Events and evidence

The letter 'e', regardless of case, is reserved for events and evidence. An uppercase 'E' is an evidence variable. An uppercase bold fonted '$\mathbf{E}$' is the set of evidence variables. A lowercase 'e' is a particular observed event variable. A bold fonted lowercase '$\mathbf{e}$' is the set of variables of a particular observed event.

A.3 Probabilities

All functions of the form $P(\cdot)$ are probability functions. Joint probability distributions have the variables separated by commas $P(X, Y)$ instead of $P(X \wedge Y)$. We call prior probabilities the probability functions of the form $P(X)$. Posterior probabilities are of the form $P(\mathbf{X}|\mathbf{Y})$.

When enumerating a set of instances we may omit commas, brackets or set name. For instance:

$P(\mathbf{X} = a\overline{b}c)$ is equivalent to
$P(\mathbf{X} = \{a, \overline{b}, c\})$ is equivalent to
$P(a\overline{b}c)$ is equivalent to
$P(a = true, b = false, c = true)$

A.4 Arrows

An arrow pointing to the right may have two possible meanings:

- Dependency
  - $A \rightarrow B$ is read as *B depends on A.*
- Directed edge connectivity
  - $A \rightarrow B$ is read as *there exists an edge from node A to node B.*

Meanings will be clear from context.

B Mathematical background

We shall now enumerate what the reader should already know before reading this paper. On certain topics we will also give some reference/bibliography on where to start. They may not be the best sources, but they are what I started with and I found them to be quite useful.

- **Basic probability**
- **Basic notion of algorithms applied to Artificial Intelligence**

  Description: gradient descent, expectation maximization, basic notion of algorithms in general, data structures

  Bibliographical notes:

  Gradient descent: Artificial Intelligence: A Modern Approach [RN10] — Section 18 Learning from Examples — 18.6 Regression and classification with linear models.

  Expectation maximization: Artificial Intelligence: A Modern Approach [RN10] — Section 20 Learning Probabilistic Models — 20.3 Learning with Hidden Variables: The EM Algorithm.

  – AIMA is not the most detailed and didactic book on EM. I recommend searching for other more detailed probabilistic oriented books.

  Algorithms, data structure, graph theory: Introduction to Algorithms [Cor+09].

- **Uncertainty and Probabilistic Reasoning**

  Description: inference, Baye's rule, Bayesian networks, inference on Bayesian networks, MEP, MAP

  Bibliographical notes:

  All: Artificial Intelligence: A Modern Approach [RN10] — Section 13: Quantifying Uncertainty and Section 14: Probabilistic Reasoning.

  All: Modeling and Reasoning with Bayesian Networks [Dar09].

- **Learning Probabilistic Models**

  Description: a basic notion on how to perform learning on probabilistic models

  Bibliographical notes:

  All: Artificial Intelligence: A Modern Approach [RN10] — Section 20 Learning Probabilistic Models.

References

[Cor+09] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd edition. The MIT Press, 2009.

[Dar03] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: (2003).

[Dar09] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. 1st Edition. Cambridge University Press, 2009.

[Dep] University of Washington Department of Computer Science. *Sum-Product Networks*. URL: `http://spn.cs.washington.edu/index.shtml`.

[DV12] Aaron Dennis and Dan Ventura. "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". In: *Advances in Neural Information Processing Systems* 25 (2012).

[DV15] Aaron Dennis and Dan Ventura. "Greedy Structure Search for Sum-Product Networks". In: *International Joint Conference on Artificial Intelligence* 24 (2015).

[GD13] Robert Gens and Pedro Domingos. "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30 (2013).

[LR15] Daniel Lowd and Amirmohammad Rooshenas. "The Libra Toolkit for Probabilistic Models". In: *Journal of Machine Learning Research* (2015). URL: `http://libra.cs.uoregon.edu/`.

[LWZ14] Sang-Woo Lee, Christopher Watkins, and Byoung-Tak Zhang. "Non-Parametric Bayesian Sum-Product Networks". In: *Workshop on Learning Tractable Probabilistic Models* (2014).

[PD11] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27 (2011).

[RL14] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *International Conference on Machine Learning 31 (ICML 2014)* (2014).

[RN10] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd edition. Prentice Hall, 2010.