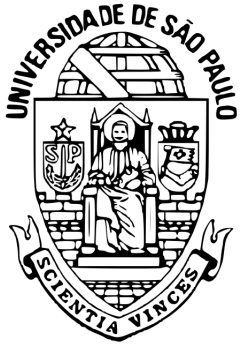

Mobile Robot Self-Driving Through Image Classification Using Discriminative Learning of Sum-Product Networks

Undergraduate Thesis

Student: Renato Lui Geh
Advisor: Prof. Denis Deratani Mauá

INSTITUTE OF MATHEMATICS AND STATISTICS
UNIVERSITY OF SÃO PAULO

São Paulo, Brazil

2018

Acknowledgements

I would like to greatly thank my advisor, Prof. Denis Deratani Mauá, for the support and attention, but most of all for the patience of having to read countless reports for both my undergraduate research and undergraduate thesis, many of which were not short.

To my parents, Chen and Luiz, for making sure I wanted for nothing, for giving me the best education possible, and for always assuring me of my abilities.

A special thank you to Maria Clara Cardoso, my best friend and confidant, whose support and company helped me immensely through the last few years. Our conversations are always filled with laughter and I will always hold them dear.

A warm thanks to my friends and colleagues Ricardo and Yan, whose camaraderie and friendship I cherish deeply.

I'd also like to express great gratitude to all professors I had during my undergraduate, whose trade is often overlooked and underappreciated, yet manage to teach us so much and inspire us to always do our best.

This work was partially supported by CNPq grant PIBIC 800585/2016.

Abstract

GEH, L. R. **Mobile robot self-driving through image classification using discriminative learning of sum-product networks**. Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil, 2018.

Driving has proven to be a very difficult task for machines to emulate, not only due to the inherent complexity of the problem but also because of the need for accurate real-time predictions. Nonetheless, recent advances in computer vision and machine learning have shown promising results in the real-world. Mobile robots are low-cost miniature computers with limited processing power and memory. The problem of self-driving can be similarly applied to the mobile robot domain as a down-scaled version of the same task, with an additional hardware constraint. Sum-product networks are probabilistic graphical models capable of representing tractable probability distributions containing a great number of variables. Exact inference is asymptotically linear to the number of edges in the network's graph, and its deep architecture is capable of representing a wide range of distributions. In this work, we attempt to model autonomous driving by using sum-product networks on a small mobile robot. We model this task as an imitation learning problem through image classification. We present accuracy results on an artificial self-driving dataset for different sum-product network learning algorithms, providing a comparative study not only for different network architectures, but also discriminative and generative models. Finally, we provide a real-world mobile robot implementation on a miniature computer.

Keywords: sum-product networks, probabilistic graphical models, machine learning, robotics

Abbreviations

DAG	Directed acyclic graph
EM	Expectation-maximization
GD	Gradient descent
IV	Indicator variable
MAP	Maximum a posteriori probability
MPE	Most probable explanation
MPN	Max-product network
PGM	Probabilistic graphical model
RV	Random variable
SGD	Stochastic gradient descent
SPN	Sum-product network
SPT	Sum-product tree

Symbols and notations

μ	Gaussian distribution mean
σ	Gaussian distribution standard deviation
$\mathcal{X}$	Sample space of random variables
Z	Partition function
ϕ	Factor (potential)
$[X = x]$	Indicator function for random variable valuation $X = x$

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

In this chapter we first describe the motivations and objectives of this thesis. Next, we describe the structure of this document.

1.1 Motivation and objectives

Self-driving is a challenging computer vision task, mainly due to its inherent complexity and the necessity for real-time decision making. Although there have been many promising results the past few years on autonomous driving, the task still relies on the underlying problem of following a pathway through visual cues (usually road markings). A possible approach to this task is through imitation learning by means of image classification. That is, the agent tasked with driving should be able to reliably mimic human behavior by correctly classifying whether to turn, stop or go straight given an image captured in front of the car.

Mobile robots are low cost machines capable of movement. These robots are usually small, and because of their size and cost often don't have the same performance capabilities as a desktop computer. However, these domain traits make mobile robot self-driving a very similar analogue to real-world autonomous cars. Processing power and memory constraints play a big role in this case, and translate well to embedded systems present in a self-driving car.

Sum-product networks (SPNs) are probabilistic graphical models that are able to represent a wide range of tractable probability distributions of many variables. SPNs have shown impressive results in several domains, and particularly that of image classification. Their deep architecture seems to capture features and contexts well, and since inference is computed in time linear to the network's edges, SPNs are promising models for fast inference in self-driving.

In this work, we attempt to model self-driving of mobile robots through image classification. For the task of classification our objective is to use sum-product networks learned discriminatively, though we also give results for generative SPNs, comparing not only generative and discriminative learning, but also different SPN architectures.

1.2 Thesis structure

This thesis is structured as follows. In chapter 2, we first provide background on sum-product networks, where we formally define an SPN, present key properties on their structure, explain how to compute exact inference and find an approximation of the maximum a posteriori probability (MAP).

In chapter 3, we show how to compute the partial derivatives with respect to a sub-SPN and to its weights, leading on how to perform gradient descent and then on learning the weights of the network through gradient descent both generatively and discriminatively.

?? is dedicated to algorithms for learning the structure of an SPN. We explain the two structural learning algorithms that were used in the experiments.

For **??**, we first show how we model self-driving as an image classification problem. We then specify the architecture of the robot used in the experiments, giving specifications on the hardware and software used. Furthermore, we describe some concepts of control we use for navigation.

In **??**, we provide classification results on many image classification datasets from various domains with different learning algorithms. We then describe the self-driving dataset used for training, and give in-dataset accuracies as well as real-world empirical results on the mobile robot itself.

Finally, in **??** we give our conclusions and provide some discussion of the results.

There is an additional section of this thesis in which we give a brief subjective insight on the work done for this thesis. We also list subjects we deemed important for the work done in this thesis.

Furthermore, **??** contains all proofs done in this thesis.

Chapter 2

Sum-product networks

In this chapter we provide some background concepts needed for defining a sum-product network. Once this is covered, we formally define an SPN, list some interesting properties on their structure, and describe how to perform exact inference (i.e. extract the probability of evidence of some valuation) and how to find an approximation of the maximum a posteriori probability.

We leave all proofs in **??**.

2.1 Background

The objective of probabilistic modelling is to compactly represent a probability distribution, be able to find a good approximation to the real function and be able to efficiently compute both the marginals and modes. Probabilistic graphical models (PGMs) attempt to solve this through the use of graphs, representing distributions as a normalized product of factors (Pearl, 1988).

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

Where $x \in \mathcal{X}$ is a d-dimensional vector valuation of RVs $\mathbf{X}$ on sample space $\mathcal{X}$, and factor (also called a potential) ϕ_k is a function mapping instantiations of X to a non-negative number. Z is the partition function $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$ that sums out all variables and normalizes the term above it to the $[0, 1]$ range.

A downside of this representation is that inference is exponential on the worst case, which makes learning also exponential, as it uses inference as a subroutine. To get around this problem, Darwiche proposed in Darwiche, 2003 the notion of *network polynomial*.

A network polynomial is a function over the probabilities of each instantiation. Let $\Phi(x)$ be a probability distribution. The network polynomial of $\Phi(x)$ is the function $f = \sum_{x \in \mathcal{X}} \Phi(x)\Pi(x)$, where $\Pi(x)$ is the product of the IVs of each variable on instantiation x,

where each indicator variable $[Y = y]$ has a value of zero if $Y \neq y$ in x and a value of one otherwise (i.e. if $Y = y$ in x or $Y \notin x$).

As an example, take the bayesian network $\mathcal{N} = A \rightarrow B$ with binary variables. Let λ_a, $\lambda_{\overline{a}}$, λ_b and $\lambda_{\overline{b}}$ be the indicator variables for when $A = 1$, $A = 0$, $B = 1$ and $B = 0$ respectively. The network polynomial of $\mathcal{N}$ is the expression

$$f_{\mathcal{N}} = P(a)P(b|a)\lambda_a\lambda_b + P(a)P(\overline{b}|a)\lambda_a\lambda_{\overline{b}} + P(\overline{a})P(b|\overline{a})\lambda_a\lambda_{\overline{b}} + P(\overline{a})P(\overline{b}|\overline{a})\lambda_{\overline{a}}\lambda_{\overline{b}}.$$

The main advantage of this representation is to avoid recomputing terms. For instance, take an instantiation of $x = \{A = 0\}$. Then, the network polynomial will be as follows.

$$\begin{aligned} f_{\mathcal{N}}(x) &= P(a)P(b|a) \cdot 0 \cdot 1 + P(a)P(\overline{b}|a) \cdot 0 \cdot 1 + P(\overline{a})P(b|\overline{a}) \cdot 1 \cdot 1 + P(\overline{a})P(\overline{b}|\overline{a}) \cdot 1 \cdot 1 = \\ &= P(\overline{a})P(b|\overline{a}) + P(\overline{a})P(\overline{b}|\overline{a}) \end{aligned}$$

Which means we can avoid computing values from the two first terms. We can also compute the network polynomial of some unnormalized probability distribution as long as we divide by the partition function, defined as the network polynomial with all indicators set to one. Although the network polynomial has exponential size in terms of variables, computing the probability of evidence is linear in its size. By representing the network polynomial as an arithmetic circuit of sums and products, one can prove that the cost of inference is indeed polynomial.

2.2 Definitions and properties

Sum-product networks borrow many concepts from network polynomials and arithmetic circuits. There are many definitions of SPNs and in this thesis we present two. The first definition is given by the seminal article Poon and Domingos, 2011, and can be seen as a more low-level approach to defining the network. The second, based on Gens and Domingos, 2013, is a stronger definition, but one which we will use more throughout this thesis, as it lends itself better to continous data.

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be the set of all variables. We shall call this set the root scope. Let G be a graph G. The sets of vertices and edges of G will be denoted by $V(G)$ and $E(G)$. We will call $\mathrm{Ch}(n)$ and $\mathrm{Pa}(n)$ the sets of children and parents of node $n \in V(G)$.

Definition 2.1 (Sum-product network; Poon and Domingos, 2011)**.** *A sum-product network (SPN) over variables $X_1, X_2, \ldots, X_n$ is a DAG whose leaves are indicator variables $[X_1 = x_1^1], [X_2 = x_2^1], \ldots, [X_n = x_n^1], \ldots, [X_1 = x_1^d], [X_2 = x_2^d], \ldots, [X_n = x_n^d]$. Its internal nodes are weighted sums or products. Each edge coming out from a sum node n to another node j has a non-negative weight associated with it. We denote such weight by $w_{n,j}$. The value of a sum node n is $v_n = \sum_{j \in Ch(n)} w_{n,j} v_j$, where v_j is the value of node j. The value of a product node n is $v_n = \prod_{j \in Ch(n)} v_j$. The value of a leaf node is the value of the indicator variable. The value of the SPN is the value of its root node.*

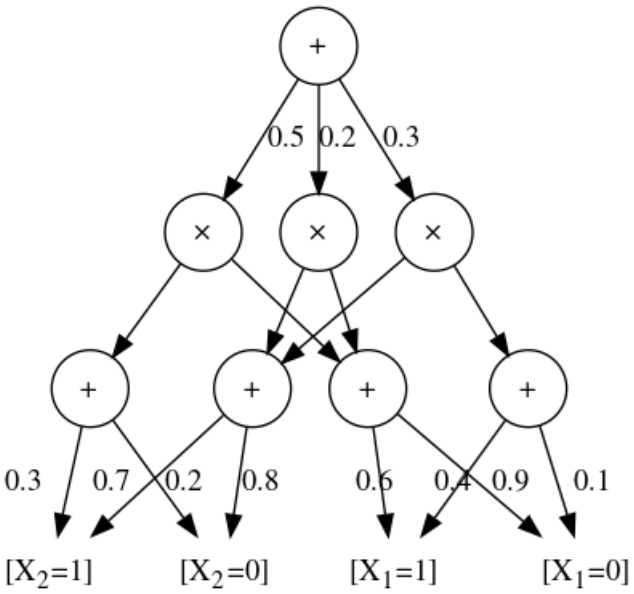

Figure 2.1: *An example of an SPN.*

Throughout this thesis, we denote by $S(X = x)$ the value of an SPN S given evidence x. A sub-SPN S_n of S is the subgraph of S rooted at n. A node in an SPN is itself an SPN. When all indicator variables are set to one, the value of S is denoted by $S(*)$. The scope of an SPN S, denoted by $\text{Sc}(S)$, is the union set of all scopes of its children. A leaf's scope is the scope of its IV.

Definition 2.2 (Validity)**.** *An SPN is valid iff, for all evidence $E = e$, $S(E = e) = \Phi_S(E = e)$, where Φ_S is an unnormalized probability distribution.*

Definition 2.3 (Completeness)**.** *An SPN is complete iff all children of the same sum node have the same scope.*

Definition 2.4 (Consistency)**.** *An SPN is 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.*

Validity in an SPN means that the network correctly and efficiently computes the probability of evidence of the distribution it represents. In this document we only work with valid SPNs, as we wish to always compute exact inference. However, non-valid SPNs are an interesting field of research for approximate inference in SPNs.

A sufficient condition for validity is completeness and consistency. Yet whilst this condition is sufficient, it is not necessary, as the converse (i.e. an incomplete and inconsistent valid SPN) can hold.

Theorem 2.1 (Poon and Domingos, 2011)**.** *An SPN is valid if it is complete and consistent.*

When an SPN S is valid, then $S(*)$ is the partition function, and we can extract the probability of evidence from an SPN by computing $P(X = x) = S(x)/S(*)$. If for every sum node all of their weights are non-negative and sum to one, then the partition function is $S(*) = 1$, and the SPN is the distribution itself.

Corollary 2.1 (Validity recursion; Poon and Domingos, 2011)**.** *If an SPN S is valid, then*

all sub-SPN of S is valid.

Definition 2.5 (Decomposability)**.** *An SPN is decomposable iff no variable appears in more than one child of a product node.*

In other words, an SPN is decomposable if and only if, for every product node, every child node has disjoint scopes with relation to all their other siblings. It is easy to see that decomposability implies in consistency, as there can be no inconsistency between product children since scopes are disjoint. Therefore, a complete and decomposable SPN is valid. Indeed it is much easier to produce decomposable SPNs then only consistent ones, and although this condition may seem too strong and restrictive, Peharz et al., 2015 showed that a consistent SPN is representable by a polynomially larger decomposable SPN.

So far, SPNs are restricted to the discrete domain, as we rely on IVs to define possible valuations to variables. We can generalize SPNs to the continuous by assuming an infinite number of IVs and thus replacing sum nodes whose children are IVs with integral nodes. A leaf node then becomes an integral node with infinite IVs as children. Particularly, it represents an unnormalized univariate probability distribution, such as a Gaussian. The value of this integral node n becomes the fdp $p_n(x)$. This extension brings us to a second definition of SPNs.

Definition 2.6 (Sum-product networks; Gens and Domingos, 2013)**.** *A sum-product network is defined recursively as follows.*

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

This second definition limits our scope to only complete and decomposable SPNs. Note that an IV is also an SPN, as we can assume that an indicator variable is a degenerate tractable univariate distribution, taking a value of one if it agrees with the given evidence and zero otherwise.

2.3 Inference

Throughout this thesis we assume that all sum nodes are normalized and sum to one, meaning the partition function is $S(*) = 1$ and the SPN's value is the probability itself.

Let $X = \{X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k\}$ be a valuation and S be an SPN. We say that X is a complete valuation if $\mathrm{Sc}(X) = \mathrm{Sc}(S)$. That is, X contains a valuation for all variables in S. An incomplete valuation has some variable assignment missing.

Computing the probability of evidence is done through a bottom-up backwards pass through the SPN. To find the value of an SPN, we must know the value of the root node, which depends on all nodes below it. This is done through a topological traversal of the graph.

Finding the value of a leaf node depends on the valuation given. Let n be a leaf node, and $\text{Sc}(n) = \{X_j\}$. Let X be some valuation. Assuming the univariate probability distribution of n has fpd $p_n(x)$, then if X has a valution $X_j = x_j$, the value of node n will be $S_n(X) = p_n(x_j)$. If X has no valuation for variable X_j, then $S_n(X)$ is the distribution's mode. Note that this holds for indicator variables, as if X has a valuation for $X_j = x_j$ and the IV matches with x_j, then $p_n(x_j) = 1$. In case it does not, $p_n(x_j) = 0$. For the incomplete case, the mode of an indicator variable is one, which holds the equivalence.

Once we compute leaf nodes, we can compute each internal node's values by following the topological order until we reach the root. For sum nodes, we compute the weighted sum $S_n(X) = \sum_{j \in \text{Ch}(n)} w_{n,j} S_j(X)$, and for products $S_n(X) = \prod_{j \in \text{Ch}(n)} S_j(X)$.

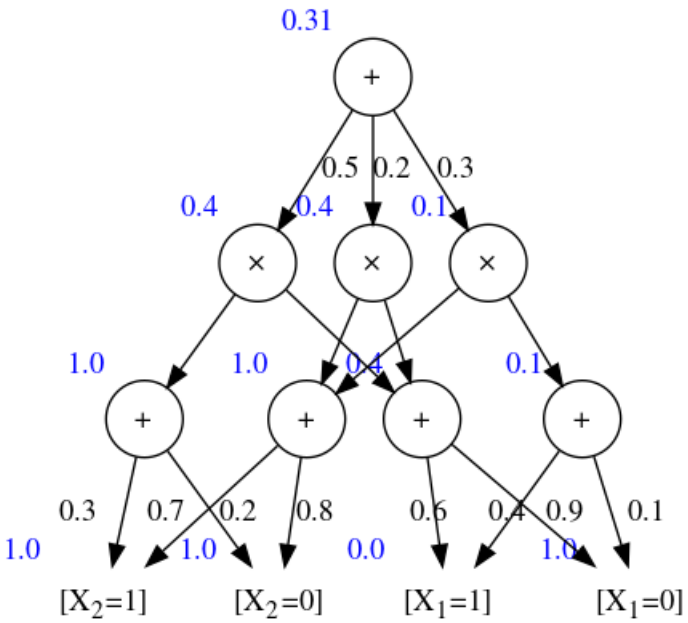

Figure 2.2: *Computing the probability of evidence on a sample SPN.*

Figure 2.2 shows the value of the SPN in Figure 2.1 given a valuation $X = \{X_1 = 0\}$. Values in blue are the values of each sub-SPN. Finding the probability of evidence $P(X = x)$ is fast, as computing the value of an SPN is linear to the number of edges of the graph.

Additionally, we might want to find the probability that maximizes a certain valuation, i.e. the maximum a posteriori probability (MAP). To compute the approximate value of the MAP of some valuation X, we first transform the SPN into a max-product network (MPN) by replacing all sums with max nodes. The value of a max node is the maximum value of its weighted children. More formally, the value of an MPN's max node n is given by $M_n(X) = \max_{j \in \text{Ch}(n)} w_{n,j} M_j(X)$. Other nodes behave identically to an SPN. The computed value of an MPN is an approximation of $\max_y P(X = x, Y = y)$, where X is incomplete and Y is the set of variables that are missing. This is called the max-product algorithm.

In SPNs, computing the exact MAP was shown to be NP-hard (Peharz et al., 2015; Conaty et al., 2017; Mei et al., 2018), and better approximation algorithms were proposed as an alternative to the max-product algorithm described here. However, in this thesis,

when we talk about computing the (approximate) MAP, we are referring to the usual max-product algorithm.

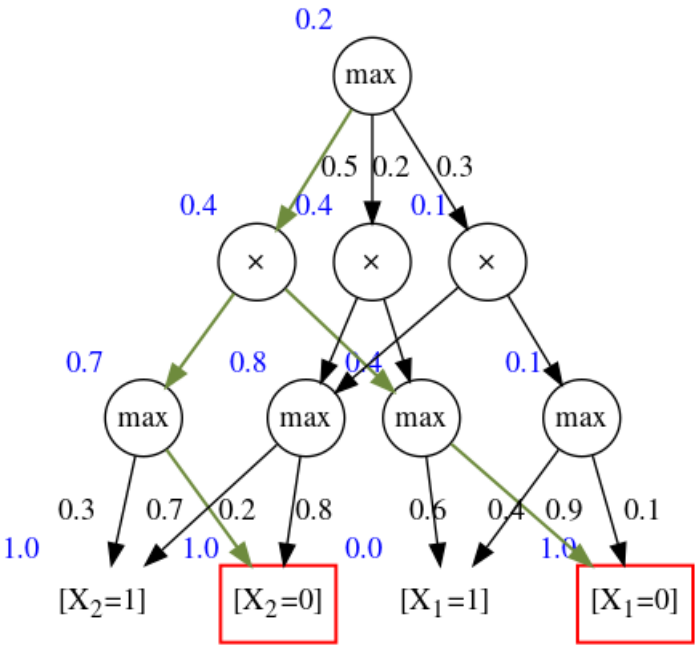

Figure 2.3: *Computing the approximate MAP of an SPN through its MPN.*

Once the MPN values are computed, we can find the most probable explanation (MPE) of the distribution given an evidence. This is done through a top-down forward pass, where we take a maximum sub-circuit path of the MPN by always taking the max path at a max node and taking all paths on a product node. The MPE are the maximum sub-circuit leaves' instantiations.

Figure 2.3 shows the MPN of the SPN shown in Figure 2.1 given $X = \{X_1 = 0\}$, where the numbers in blue represent the MPN values at each node, green arrows indicate the sub-circuit of maximum value and red boxes indicate the most probable valuations given evidence. The resulting MPE $\arg\max_{y \in \mathcal{Y}} P(X = \{X_1 = 0\}, Y = y)$ is the valuation $\{X_1 = 0, X_2 = 0\}$.

Therefore, computing the probability of evidence, which is also called *soft* inference, of an SPN is done through a single bottom-up pass. Similarly, computing the MAP probability, refered to as *hard* inference, is done through a bottom-up pass on the SPN's MPN. On the other hand, finding the MPE valuations requires a bottom-up pass to first compute the MAP, and then a top-down search to find the most probable instantiations.

We provide next pseudocode for computing both soft and hard inference. We assume as input only valid, (weight) normalized SPNs. However, one could easily extend the included algorithms for unnormalized networks.

Algorithm 1 `SoftInference`: Computes the probability of evidence in SPNs

Input A valid SPN S with normalized weights and a valuation X

Output The soft inference values at each node S_n

Initialize $S_n = 0$
Find topological order T of S
for each node $n \in S$ from T **do**
    if n is a leaf node **then**
        Let $\mathrm{Sc}(n) = \{X_k\}$, $p_n(x)$ be n's fdp and $\hat{p}_n$ be p_n's mode
        if $X_k \in X$ **then**
            Let x_k be X_k's value in X
            $S_n \leftarrow p_n(x_k)$
        else
            $S_n \leftarrow \hat{p}_n$
    else if n is sum node **then**
        for all $j \in \mathrm{Ch}(n)$ **do**
            $S_n \leftarrow S_n + w_{n,j} S_j$
    else
        for all $j \in \mathrm{Ch}(n)$ **do**
            $S_n \leftarrow S_n \cdot S_j$
return each S_n node value

Algorithm 2 `HardInference`: Computes an approximation of the MAP in SPNs

Input A valid SPN S with normalized weights and a valuation X

Output The hard inference values at each node M_n

Let M be S's MPN
Initialize $M_n = 0$
Find topological order T of M
for each node $n \in M$ from T **do**
    if n is a leaf node **then**
        Let $\mathrm{Sc}(n) = \{X_k\}$, $p_n(x)$ be n's fdp and $\hat{p}_n$ be p_n's mode
        if $X_k \in X$ **then**
            Let x_k be X_k's value in X
            $M_n \leftarrow p_n(x_k)$
        else
            $M_n \leftarrow \hat{p}_n$
    else if n is sum node **then**
        for all $j \in \mathrm{Ch}(n)$ **do**
            $M_n \leftarrow \max(M_n, w_{n,j} M_j)$
    else
        for all $j \in \mathrm{Ch}(n)$ **do**
            $M_n \leftarrow M_n \cdot M_j$
return each M_n node value

Finally, we show how to algorithmically compute the MPE given some evidence.

Algorithm 3 `ArgMaxSPN`: Finds the MPE of a valuation on an SPN

Input A valid SPN S with normalized weights and a valuation X

Output The arg max values of each variable according to X

$M \leftarrow$ `HardInference`(S, X)
Let Y be a copy of X
Let Q be a queue
Push S into Q
for each node $n \in M$ in Q **do**
    if n is a leaf node **then**
        Let $\mathrm{Sc}(n) = \{X_k\}$ and $p_n(x)$ be n's fdp
        Let $\hat{x} = \arg\max_{x_k} p_n(x_k)$ be p_n's maximum valuation
        if $X_k \notin X$ **then**
            $Y \leftarrow Y \cup \{X_k = \hat{x}\}$
    else if n is sum node **then**
        Push maximum child M_j, $j \in \mathrm{Ch}(n)$ into Q
    else
        Push all children $j \in \mathrm{Ch}(n)$ into Q
return Y

Chapter 3

Parameter learning

The objective of this chapter is to expose the ideas behind generative and discriminative gradient descent for parameter learning of sum-product networks. We first show how to derive the SPN with respect to its nods and weights so that we can find the gradient of the SPN wrt its parameters (i.e. weights). This allows us to find the weight updates needed for gradient descent on SPNs. We then describe how to perform generative stochastic gradient descent, and finally discriminative gradient descent.

3.1 Derivatives

Let S be an SPN. We are only interested in finding the derivative of internal nodes, as leaf nodes have no weights to be updated. Our objective is to find the derivative $\partial S/\partial W$ so that we can compute the gradient at each weight and update parameters accordingly.

At each weighted edge $(n \to j, w_{n,j})$, the derivative $\partial S/\partial w_{n,j}$ takes the form

$$\frac{\partial S}{\partial w_{n,j}}(X) = \frac{\partial S}{\partial S_n}\frac{\partial S_n}{\partial w_{n,j}}(X) = \frac{\partial S}{\partial S_n}\frac{\partial}{\partial w_{n,j}}\left(\sum_{i\in \mathrm{Ch}(n)} w_{n,i}S_i(X)\right) = \frac{\partial S}{\partial S_n}S_j(X). \tag{3.1}$$

The term $\partial S/\partial S_n$ appears because of chain rule, since S_n is a function of S. This can be intuitively interpreted as taking into account the change in all nodes "above" n. So to compute the derivative wrt a weight, we need to find the derivative $\partial S/\partial S_j$ for each internal node j.

Finding $\partial S/\partial S_j$ requires analyzing two possible cases: sum and product parents of j. We now that S is a multilinear function of X, since in reality S is just a function made of sums and products. In particular, if we apply chain rule on $\partial S/\partial S_j$, we have that

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

We expand each term at a time. Starting with the sum parents case, we can substitute the value of $S_n(X)$ with the corresponding expansion.

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

We do the same for the product case.

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

Which brings us to the final form.

$$\frac{\partial S}{\partial S_j}(X) = \sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: sum}}} \frac{\partial S}{\partial S_n} w_{n,j} + \sum_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: product}}} \frac{\partial S}{\partial S_n} \prod_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k \tag{3.2}$$

Note how each $\partial S / \partial S_j$ depends on the derivative of its parents. This dependency goes all the way up to the root, where $\partial S / \partial S = 1$. This derivation lends itself neatly to an algorithmic format.

Algorithm 4 `BackpropSPN`: 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**
            $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + \frac{\partial S}{\partial S_n} \prod_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k$

Computing all derivatives and forward passes is fast, as it takes linear time in the

number of edges. However, these values suffer from gradient diffusion, as their signal dwindles the deeper the network, eventually becoming zero.

A possible solution to this issue is replacing soft derivation with hard derivation. This is done by finding the derivatives of the MPN of the network instead of the SPN. This guarantees that the signal remains constant throughout the structure, at the cost of slower convergence rate.

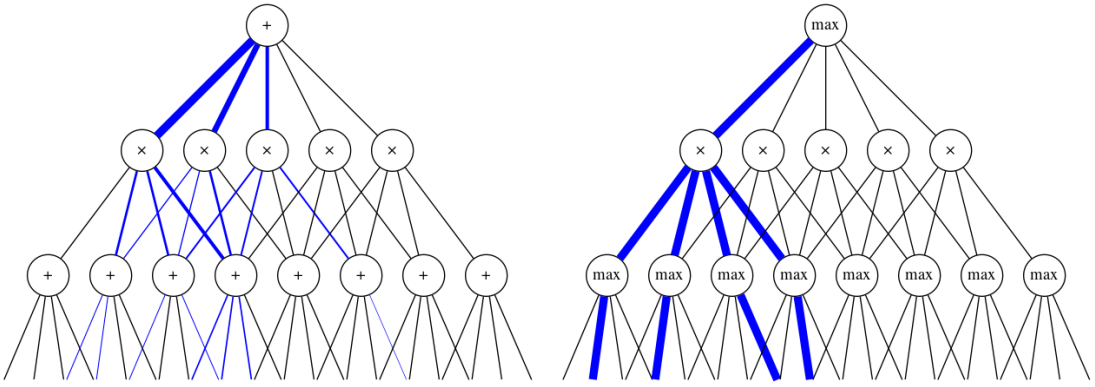

Figure 3.1: *Signal difference between soft and hard derivation.*

Figure 3.1 gives a visual representation of the difference between soft and hard derivation in gradient descent.

To compute the hard derivatives of an SPN, we take its MPN and find its derivatives in a similar way as in soft derivation. Let M be an MPN. We shall call W the multiset of weights that a forward pass through M visits. The value of M is $M(X) = \prod_{w_i \in W} w_i^{c_i}$, where c_i is the number of times w_i appears in W. We can then take the logarithm of the MPN to end up with a friendlier expression.

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

If we assume that weights are strictly positive, the resulting expression results in the final hard derivative

$$\frac{\partial \log M}{\partial w_{n,j}} = c_{n,j} \frac{w_{n,j}^{c_{n,j}-1}}{w_{n,j}^{c_{n,j}}} = \frac{c_{n,j}}{w_{n,j}}. \tag{3.3}$$

3.2 Generative gradient descent

Once computed all derivatives, we update each node with the resulting gradient. For generative gradient descent, where we are learning a joint probability distribution $P(X, Y)$, our objective is to find the gradient of the log-likelihood

$$\frac{\partial}{\partial W}\log P(X,Y) = \frac{\partial}{\partial W}\log S(X,Y) = \frac{1}{S(X,Y)}\frac{\partial S}{\partial W}(X,Y) \propto \frac{\partial S}{\partial W}(X,Y).$$

Since the gradient is proportional to the derivative of the weights, our weight update becomes

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

where η is the learning rate. An L2 regularization factor can be added to the expression above, leaving us with the final generative gradient descent weight update

$$\Delta w_{n,j} = \eta \frac{\partial S}{\partial w_{n,j}}(X,Y) - 2\lambda w_{n,j}, \tag{3.4}$$

where λ is the regularization constant.

Bibliography

[Conaty et al. 2017] Diarmaid Conaty, Denis D. Mauá, and Cassio P. de Campos. "Approximation Complexity of Maximum A Posteriori Inference in Sum-Product Networks". In: *Uncertainty in Artificial Intelligence 2017 (UAI 2017)* (2017) (cit. on p. 7).

[Darwiche 2003] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: (2003) (cit. on p. 3).

[Gens and Domingos 2013] Robert Gens and Pedro Domingos. "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30 (2013) (cit. on pp. 4, 6).

[Mei et al. 2018] Jun Mei, Yong Jiang, and Kewei Tu. "Maximum a Posteriori Inference in Sum-Product Networks". In: *Association for the Advancement of Artificiall Intelligence 32 (AAAI 2018)* (2018) (cit. on p. 7).

[Poon and Domingos 2011] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27 (2011) (cit. on pp. 4, 5).

[Pearl 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988 (cit. on p. 3).

[Peharz et al. 2015] Robert Peharz, Sebastian Tschiatschek, Frank Pernkopf, and Pedro Domingos. "On Theoretical Properties of Sum-Product Networks". In: *International Conference on Artificial Intelligence and Statistics 18 (AISTATS 2015)* (2015) (cit. on pp. 6, 7).