DISSERTATION

submitted to the

Combined Faculty for the Natural Sciences

and Mathematics

of

Heidelberg University, Germany

for the degree of
Doctor of Natural Sciences

put forward by

M.Sc. Mattia Desana

born in Genoa, Italy

Date of oral examination: 12.12.2017

Sum-Product Graphical Models: A Graphical Model Perspective on Sum-Product Networks

Advisor: Prof. Dr. Christoph Schnörr

Abstract

The trade off between expressiveness of representation and tractability of inference is a key issue of probabilistic models. On the one hand, probabilistic Graphical Models (GMs) provide a high level representation of distributions, but exact inference with cyclic graphs is in general intractable. On the other hand, Sum-Product Networks (SPNs) allow tractable exact inference with probability distributions that are more complex than tractable GMs, but they employ a low level representation of the underlying distribution, which is much harder to read and interpret than in GMs.

The objective of this thesis is to close this gap and to achieve simultaneously the high level representation of GMs and the efficiency of SPNs. To this aim, new models and procedures are introduced.

We first investigate SPNs that include GMs as a submodule, obtaining a derivation of Expectation-Maximization for SPNs which is the first allowing to learn the GM part alongside the SPN parameters.

Then, we introduce a new architecture called Sum-Product Graphical Model (SPGM). This new architecture is the first to combine the semantics of graphical models with the evaluation efficiency of SPNs: SPGMs always enable tractable inference using a class of models that incorporate context specific independence (like SPNs), and they provide a high-level model interpretation in terms of conditional independence assumptions and corresponding factorizations (like GMs). An algorithm for learning both the structure and the model parameters of SPGMs is also introduced.

Finally, several applications that illustrate and empirically motivate the introduction of the new models are described. SPGMs are applied to real-world discrete density estimation datasets, to augment a graphical model for segmenting scans of the human retina and detecting local pathologies, and to model very large mixtures of Quadtrees for image denoising. Strong empirical results and novel application areas denote promise for future applications of SPGMs.

Zusammenfassung

Der Ausgleich zwischen der Aussagekraft einer Darstellung und der Möglichkeit zur Inferenzberechnung ist eine zentrale Problemstellung bei Graphischen Modellen. Einerseits bieten Graphische Modelle (GMs) eine fortgeschrittene Darstellung von Verteilungen, wobei exakte Inferenz auf zyklischen Graphen im Allgemeinen unmöglich ist. Andererseits erlauben Summen-Produkt Netzwerke (SPNs) berechenbare, exakte Inferenz mit Wahrscheinlichkeitsverteilungen, die komplexer sind als realisierbare GMs. Diese wiederum verwenden eine primitive Darstellung der zugrunde liegenden Verteilung, die im Vergleich zu GMs viel schwerer zu lesen und zu interpretieren ist.

Die Zielsetzung dieser Arbeit ist diese Lücke zu schließen und die fortschrittliche Darstellung von GMs mit der Effektivität von SPNs zu verbinden. Um dieses Ziel zu erreichen werden neue Modelle und Vorgehensweisen entwickelt.

Erst untersuchen wir SPNs, die GMs als Untermodul enthalten, wodurch man eine Art Erwartungs-Maximierung für SPNs erhält, die es erlaubt den GM-Anteil zusammen mit den SPN-Parametern zu lernen.

Danach führen wir eine neue Architektur namens Summen-Produkt Graphisches Modell (SPGM) ein. Diese neue Architektur ist die Erste, die in der Lage ist die Semantik von Graphischen Modellen mit der Auswertungseffektivität von SPNs zu kombinieren: Bei SPGMs ist Inferenz immer realisierbar, wobei eine Klasse von Modellen benutzt wird, die kontextabhängige Unabhängigkeit mit ins Spiel bringt (ähnlich zu SPNs) und die eine fortgeschrittene Interpretierbarkeit des Modells im Sinne von Annahmen bedingter Unabhängigkeit und entsprechender Faktorisierungen (ähnlich zu GMs) bietet. Ein Algorithmus für das Lernen von sowohl der Struktur als auch der Modellparameter von SPGMs wird ebenfalls vorgestellt.

Abschließend werden einige Anwendungen beschrieben, die die Einführung der neuen Modelle illustrieren und empirisch motivieren. SPGMs werden zur diskreten Dichteschätzung realer Daten verwendet, zur Erweiterung eines Graphischen Modells für das Klassifizieren von Aufnahmen der menschlichen Netzhaut und das Erkennen von lokalen Pathologien, und zur Kombination von zahlreichen Quaternärbäumen (Quadtrees) für das Entrauschen von Bildern. Überzeugende empirische Resultate und neuartige Anwendungsbereiche sind vielversprechend im Hinblick auf die zukünftige Verwendung von SPGMs.

Acknowledgments

There are many people to whom I owe a great debt of gratitude, and without whom I would not have been able to start, let alone finish this PhD.

First, I express my many thanks to my supervisor Prof. Christoph Schnörr for his constant insightful guidance, his encouraging support and his belief in my research despite all adversities, which kept me going when times were most difficult. I also wish to thank Prof. Fred Hamprecht for the useful discussion, the interesting courses and the enthusiasm for research.

Then, the group. I thank all my colleagues for sharing the good and the not so good times: Andreas, Vera, Tabea, Jörg, Paul, Matthias, Andreea, Ecaterina, Evelyn, Fabian, Florian, Bogdan, Johannes, Jan, and all the many others. Many thanks to Barbara for the tea, for the croissants, for being such a good listener and such a good person.

A special thought goes to the three people with whom I shared the cabin in our PhD boat. Robert, who was the first to leave, which I thank for showing me what a real German is, for the football, for translating the abstract, and for the jokes. You'll be a wonderful father! Tobias, for the endless discussion that I will always remember as a fun and interesting time, and for being so originally Tobias - this is invaluable. And finally Francesco, who was with me until the very end of this trip: even when it felt like being on the Titanic and going straight for the iceberg, we still played our music pretty well, and at the end it turned out to be fine... I would say: island escape mission succeed!

Then many thanks to Kira, for the badminton, for the emails, for the interesting exchanges of very different point of views, and for making my life in Heidelberg way better than it would have otherwise been. Then thanks to Yue and Davide, a bit of Italy and a bit of China-Italy in Heidelberg, which have been faithful friends through thick and thin. Especially, Yue, for the care and support when I was sick and (briefly) homeless. And thanks to Yifan, and to Marina with the contagious smile. Finally, I thank Grazia, who was my bless and my thorn, my chef and my pupil, my friend and foe. Thank you for everything - which is too much to put into words. You are really quite a girl.

And in the end, the most heartfelt thank you to my beloved parents, to whom I simply owe all. And also to Salvatore, which I hope is fine. And to Pit. And to my uncle, who is always with me in my soul.

Contents

List of Figures

List of Tables

List of Algorithms

Table of Notation

General Notation

$X = \{X_1, X_2, \ldots, X_N\}$	:	A set of random variables, either continuous or discrete depending on the context.
$\Delta(X_s)$	:	Domain (set of values) of variable X_s.
$\Delta(X)$	:	Domain of set of variables X: $\Delta(X) = \{\Delta(X_1) \times \Delta(X_2) \times \cdots \times \Delta(X_N)\}$.
$\sum_{x \in \Delta(X)}, \int_{x \in \Delta(X)}$	:	Sum and integral over the domain of X.
$[X_s]_{x_s}$	:	An indicator variable that has value 1 if the discrete variable X_s takes value x_s and 0 otherwise.
$[X]_x$	:	An indicator variable that has value 1 if the set of discrete variables X takes a joint value x and 0 otherwise.
$\mathcal{G} = \mathcal{V}, \mathcal{E}$	:	A graph $\mathcal{G}$ with vertices $\mathcal{V}$ and edges $\mathcal{G}$.
$pa(s), ch(s)$	:	Parents and children of node $s \in \mathcal{V}$.
$\mu_{t \to s}$	:	Message sent from node $t \in \mathcal{V}$ to node $s \in \mathcal{V}$.
$\varphi_{st}(X_s, X_t)$	:	Factor associated to edge $(s, t) \in \mathcal{E}$.

Sum-Product Networks

S	:	A Sum-Product Network.
S_t	:	Sub Sum-Product Network rooted in node t of S.
$\mathcal{L}(S)$	:	Set of Leaf Nodes in S.
σ	:	A subnetwork of S (definition 3.1.5).
$\lvert S \rvert$	:	Cardinality of S, which is the number of distinct subnetworks obtainable from S.
P_σ	:	Probabilistic model corresponding to σ (Table 3.1.1)
λ_σ	:	Coefficient corresponding to σ (Table 3.1.1)
W	:	Set of all sum node weights in S.
θ	:	Set of all factor parameters in S.

Sum-Product Graphical Models

$\mathbb{S}$	:	A Sum-Product Graphical Model.
$\mathbb{S}_t$	:	Sub Sum-Product Graphical Model rooted in node t of $\mathbb{S}$.
τ	:	A subtree of $\mathbb{S}$ (definition 3.1.5).
$\lvert \mathbb{S} \rvert$	:	Cardinality of $\mathbb{S}$, which is the number of distinct subtrees obtainable from $\mathbb{S}$.
P_τ	:	Probabilistic model corresponding to τ (Table 4.1.1)
λ_τ	:	Coefficient corresponding to τ (Table 4.1.1)

1. Introduction

The Quest for Tractable Inference The computational cost involved in obtaining answers to probabilistic queries (called *inference*) is a crucial aspect in probabilistic models: representing probability distribution with complex structure is of little practical use if inference cannot be performed with a tractable cost both in memory and time. Hence, research in probabilistic architectures devoted much effort in devising models in which inference is *tractable*.

The problem of developing models with tractable inference has been approached in mainly two ways. One way is to find *approximate inference* procedures, which allow to evaluate approximately correct inference from models in which exact inference is intractable. The idea behind this approach is to let the the end user free to design a model according to the desired application, not considering the cost of inference, and to let inference in the model be approximated and thus be made tractable at evaluation time. A prominent example in this field is the family of cyclic graphical models, in which several approximate inference procedures are available (see e.g. Pearl [2000]).

Another way of approaching this problem is to develop models in which inference is tractable by construction (*tractable models*). Models in this family typically require additional constraints with respects to their intractable counterparts, and are hence more complicated to design for the end user. However, they provide the guarantee that any inference query can be answered in tractable time, with no need for approximations. A classical example belonging to this family are tree graphical models, in which inference and learning can be performed tractably, but that require additional care from the end user in ensuring that the underlying graph has a tree structure (Chow and Liu [1968]).

Until recently, research in probabilistic models was heavily focused into the first family of architectures, which could model more complex probability distributions than models in the second family - for instance, a particularly active field of research has been (and still is) developing approximate inference methods for graphical models (Wainwright and Jordan [2008]). However, in recent years the situation has slightly shifted, since new tractable models capable of representing distributions with structural complexity close to (or, at times, surpassing) graphical models have been introduced.

Tractable but Expressive Models The "tractable but expressive" models that are the subject of this thesis (and will be described in detail in the following) have a historically recent origin, which can be found in Arithmetic Circuits (Darwiche [2002]), a model that was originally employed to represent inference in a particular subset of graphical models. This model was later followed by the closely related And/Or graphs (Dechter

and Mateescu [2007]), with a very similar formalism. Finally, with the introduction of Sum-Product Networks (SPNs, Poon and Domingos [2011], again closely related to Arithmetic Circuits) these models started to be designed as "standalone" models in their own right, rather than as a mere tool compilation target for inference graphical models. From this point on, hand crafted applications of this family of architectures flourished in many fields, especially excelling in density estimation (see e.g. Rooshenas and Lowd [2014], Cheng et al. [2014], Amer and Todorovic. [2015], Rahman and Gogate [2016b], Rahman and Gogate [2016a]). In the rest of this thesis we will work with Sum-Product Networks as reference model for this family of architectures.

We will see that all these models admit guaranteed tractable inference, but at the same time allow to express some distribution which do not admit an efficient representation as graphical models. Despite these appealing properties, this family of tractable yet powerful architectures has a major disadvantage with respect to other probabilistic models in that the representation they employ is a very *low level* one. In particular, SPNs lose the factorization due to graphical models, and hence the distribution properties are both difficult to read from the model and it is very hard to design SPNs with a specific task in mind. This aspect will be made clear in the following.

Hence, a trade off arises between the two families of architectures: graphical models provide a high level representation with a rich theoretical framework, but inference is worst case intractable, while SPNs guarantee tractable inference in highly expressive distributions but employ a low level representation which makes it hard to model and understand the distribution's properties.

An Unavoidable Compromise? For the sake of this introductory discussion, we can summarize these crucial aspects - which form the backbone of our subsequent work - with the diagrams in Fig. 1.1.1, leaving further details to the next sections. Essentially, in Fig. 1.1.1, left, we can see that some distributions in which exact inference can be computed efficiently by SPNs cannot be represented efficiently by GMs, and vice versa. On the right, we can see that choosing a certain model over another always entails a trade off of some positive aspect of the representation.

But is this compromise unavoidable? As the ideal model would cover the whole areas in Fig. 1.1.1, it is clear that these representations could be improved. Indeed, several attempts were made in the literature to close the gap between efficiency and high level representation - we discuss related work in Section 1.4. However, until today the basic trade off still stands.

The main objective of this thesis is to *bridge the gap* between Graphical models and Sum-Product Networks, by obtaining models that have the high-level representation of Graphical Models and the efficiency of Sum-Product Networks. Using the diagrams above, our objective can be graphically represented as obtaining a representation whose area is as large as possible, akin to Fig. 1.1.2. To reach this aim we develop several contributions, which we introduce in the remainder of this chapter and that constitute the main content of this thesis.

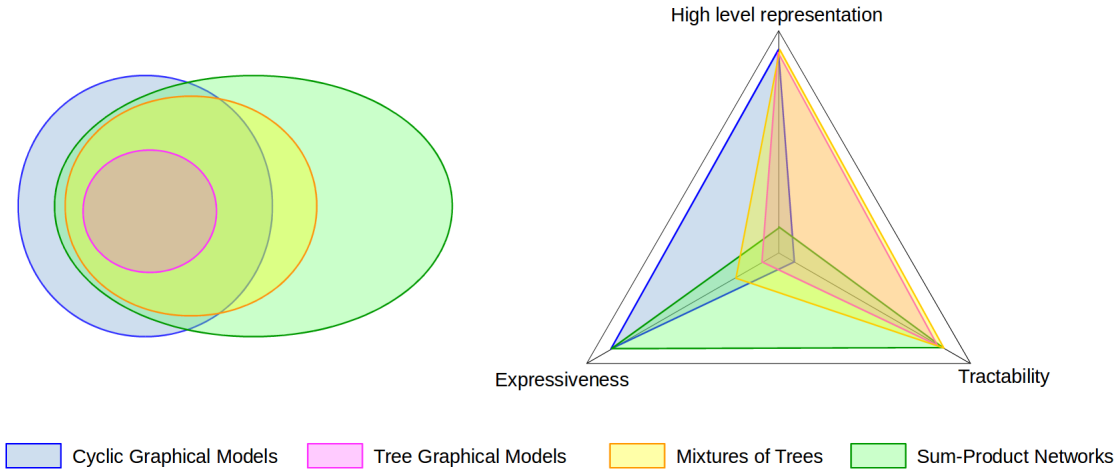


Figure 1.1.1. - Left: Venn diagram representing the set of distributions in which inference is tractable for the four considered models. We anticipate that the set of distributions where exact inference can be represented by graphical models but not by SPNs (the area in blue) corresponds to graphs where inference can be computed only with a maximum flow approach (Kolmogorov and Zabih [2004]); the set of distributions where exact inference can be represented by SPNs but not by GMs (in green) corresponds to distributions that are tractable due to contextual independence.
Right: Qualitative graphical representation of three positive characteristics of the considered model: 1) Expressiveness (how large a sets of distributions can be expressed by the model), 2) Tractability of inference, 3) Usage of a high level representation.

1.1. Graphical Models and Sum-Product Networks

In order to proceed, it is now necessary to describe in more detail the trade off between expressiveness and tractability introduced above and graphically represented in Fig. 1.1.1. We start by providing a preliminary description of the two families of models that constitute the focus of the thesis: Probabilistic Graphical Models and Sum-Product Networks.

- Probabilistic Graphical Models (GMs) are a well established family of models that encode probability distributions through a graph representing conditional independence relationships between variables (example in Fig. 1.1.3a). GMs provide a *high level representation* of the encoded distribution, in which the relations between variables can be easily read through graph theoretical properties, and enjoy a well established analytical framework for inference and learning. Regarding GMs, tractable inference is guaranteed for acyclic graphical models and GMs on cyclic graphs with small treewidth, i.e. on graphs that after triangulation admit a tree-decomposition which induces only maximal cliques of small size (Diestel [2006]). However, except for a subset of discrete graphical models (see, e.g., Kolmogorov and Zabih [2004]) where inference can be reformulated as a maximum flow problem, inference with cyclic graphical models generally suffers from a complexity that

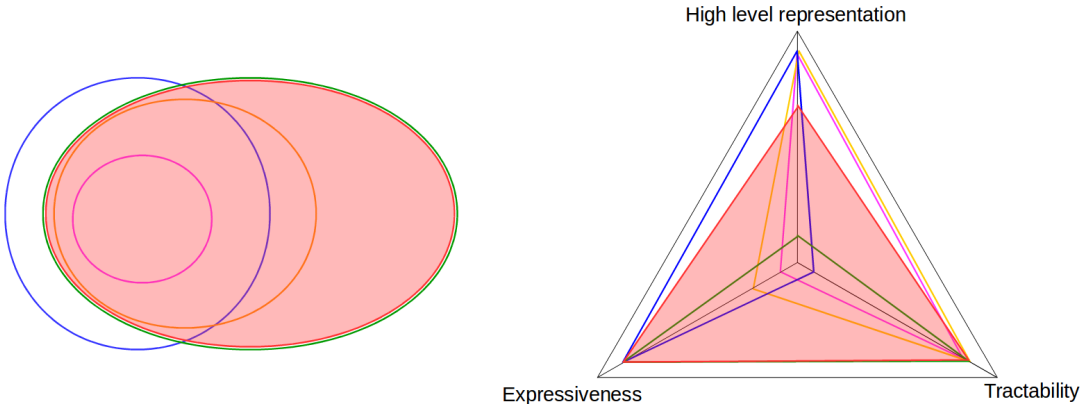
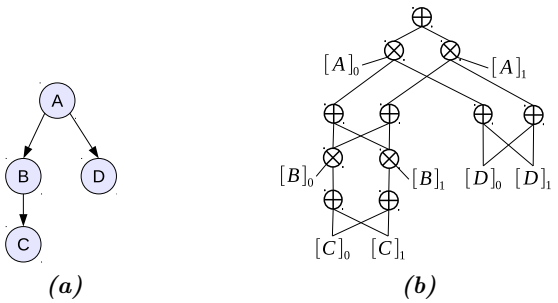


Figure 1.1.2. - The aim of this thesis is to *bridge the gap* between SPNs and Graphical models. Ideally, such models would have areas that completely overlap the ones in Fig. 1.1.1. The main model introduced in this thesis, called Sum-Product Graphical Model (SPGM - in red) , has a shape closer than SPNs to the ideal one by providing a high level representation of SPNs.

exponentially grows with the treewidth of the underlying graph, so that approximate inference is the only viable choice.

- *Sum-Product Networks* (SPNs, Poon and Domingos [2011]), and closely related architectures including Arithmetic Circuits and And-Or Graphs (Darwiche [2002], Dechter and Mateescu [2007], Poon and Domingos [2011]), represent a distribution through a Directed Acyclic Graph with sum and products as internal nodes and probability distributions as leaves (example in Fig. 1.1.3b). Recently, SPNs have received attention in the probabilistic machine learning community, mainly due to two attractive properties:
  - *Efficiency.* These architectures allow to cope with probability distributions that are more complex than tractable graphical models as characterized above. A major reason is that SPNs enable an efficient representation of *contextual independences*: independences between variables that only holds in connection with some assignment of a subset of variables in the model, called "context". Exploiting contextual independence allows to drastically reduce the cost of inference, whenever the modelled distribution justifies this assumption. By contrast, as discussed by Boutilier et al. [1996], GMs cannot represent contextual independence compactly, since the connection between nodes in a GM represent *conditional* independences rather than *contextual* ones. As a result, a significant subset of distributions that would be represented by graphical models with high treewidth (due to the inability to exploit contextual independence) can be represented by SPNs in a tractable way. A detailed example illustrating this key point is provided in Section 1.1.1.
  - *Tractability.* SPNs ensure that the cost of inference is always *linear* in the model



(a) *(b)*

Figure 1.1.3. - Representation properties of Graphical Models (GMs) and Sum-Product Networks (SPNs). The same distribution specified by Eq. (1.1.1) is represented by a GM in panel (a) and by a SPN in panel (b). This illustrates that GMs represent conditional independence more compactly than SPNs.

size and, therefore, inference is always tractable. This aspect greatly simplifies approaches to learning the structure of such models, the complexity of which essentially depends on the complexity of inference as a subroutine. In recent work, it has been shown empirically that structure learning approaches for SPNs produce state of the art results in density estimation (see e.g. Gens and Domingos [2013], Rooshenas and Lowd [2014], Rahman and Gogate [2016b], Rahman and Gogate [2016a]), suggesting that performing exact inference with tractable models might be a better approach than approximate inference using more complex but intractable models.

On the other hand, the ability of SPNs to represent efficiently contextual independency is due to a *low-level* representation of the underlying distribution. This representation comprises a Directed Acyclic Graph with sums and products as internal nodes, and with indicator variables associated to each state of each variable in the model, that become active when a variable is in a certain state (Fig. 1.1.3(b)).

Thus, SPN graphs directly represent the flow of operations performed during the inference procedure, which is much harder to read and interpret than a factorized graphical model due to conditional independence. In particular, the factorization associated to the graphical model is lost after translating the model into a SPN, and can only be retrieved (when possible) through a complex hidden variable restoration procedure (Peharz et al. [2016]). As a consequence of these incompatibilities, research on SPNs has largely evolved without much overlap with work on GMs.

1.1.1. Tradeoff Between High-Level Representation and Efficient Inference: An Example

Let us now delve into the relationship between Sum-Product Networks and Graphical Models, by providing an example that showcases the trade off involved in the two

representations. We consider a distribution of discrete random variables A, B, C, D in the form:

$$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|A), \tag{1.1.1}$$

which corresponds to Fig. 1.1.3a. Uppercase letters A, B, C, D denote random variables and corresponding lowercase letters a, b, c, d denote values in their domains $\Delta(A)$, $\Delta(B)$, $\Delta(C)$, $\Delta(D)$. We write $\sum_{a,b,c,d}$ for the sum over the joint domain $\Delta(A) \times \Delta(B) \times \Delta(C) \times \Delta(D)$. Using this notation, the distribution $P(A, B, C, D)$ can be written in the form of a *network polynomial* (Darwiche [2003]) as:

$$P(A, B, C, D) = \sum_{a,b,c,d} P(a, b, c, d)[A]_a[B]_b[C]_c[D]_d. \tag{1.1.2}$$

Here $P(a, b, c, d)$ denotes the *value* of P for assignment $A = a, B = b, C = c, D = d$, and $[A]_a, [B]_b, [C]_c, [D]_d \in \{0, 1\}$ denote *indicator variables*. This representation transforms the original density into a polynomial in the indicator variables with coefficients $P(a, b, c, d)$. The distribution can be evaluated by assigning the indicator variables: for instance, to compute the partition function all indicator variables of (1.1.2) are set to 1, and to compute the marginal probability $P(A = 1)$ one sets $[A]_1 = 1, [A]_0 = 0$ and all the remaining indicators to 1.

The next step is to exploit the factorization of P on the right-hand side of (1.1.1) in order to rearrange the sum of (1.1.2) more economically in terms of messages μ, which results in the sum-product message passing formulation equivalent to (1.1.2),

$$P(A, B, C, D) = \sum_{a \in \Delta(A)} P(a)[A]_a \mu_{b,a}(a)\mu_{d,a}(a), \quad \mu_{b,a}(A) = \sum_{b \in \Delta(B)} P(b|A)[B]_b \mu_{c,b}(b), \tag{1.1.3a}$$

$$\mu_{c,b}(B) = \sum_{c \in \Delta(C)} P(c|B)[C]_c, \qquad \mu_{d,a}(A) = \sum_{d \in \Delta(D)} P(d|A)[D]_d. \tag{1.1.3b}$$

As discussed above, the distribution can be represented in two forms: The first one is a directed graphical model (GM) conforming to Eq. (1.1.1), shown in Fig. 1.1.3a. The second one is a Sum-Product Network (SPN) shown by Fig. 1.1.3b, which directly represents the computations expressed by Eqns. (1.1.3), with the coefficients $P(\cdot|\cdot)$ omitted in Fig. 1.1.3b for better visibility. It is evident that the SPN does not clearly display the high level semantics due to conditional independence of the graphical model. On the other hand, the SPN makes explicit the computational structure for efficient inference and encodes more compactly than GMs a class of relevant situations described next.

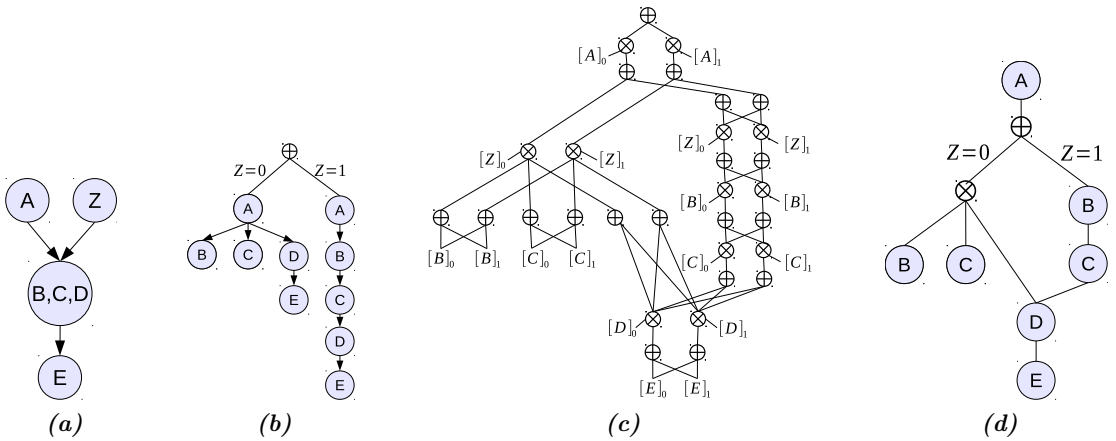


Figure 1.1.4. - The distribution in Eq. 1.1.4 represented (from left to right) as Graphical Model (GM), as mixture of GMs, as Sum-Product Network (SPN) and as Sum-Product Graphical Model (SPGM).

1.1.2. Introducing Contextual Independence

We consider a distribution in the form

$$P(A, B, C, D, E, Z) = P(Z)P(A)P(B, C, D|A, Z)P(E|D) \tag{1.1.4a}$$

with

$$P(B, C, D|A, Z) = \begin{cases} P(Z = 0)P(B|A)P(C|A)P(D|A) & \text{if } Z = 0, \\ P(Z = 1)P(B|A)P(C|B)P(D|C) & \text{if } Z = 1. \end{cases} \tag{1.1.4b}$$

Notice that different independency relations hold depending on the value taken by Z: if $Z = 0$, then B, C and D are conditionally independent given A, whereas if $Z = 1$, then they form a chain. We therefore say that this distribution exhibits *context specific independence* with *context variable* Z.

As in the example before, this distribution can be represented in different ways. Firstly, choosing a graphical model (GM) (Fig. 1.1.4a) requires to model $P(B, C, D|A, Z)$ as a single factor over 5 variables, since GMs cannot directly represent the *if* condition of (1.1.4b).[1]

Secondly, we may represent the distribution as a *mixture* of two tree-structured GMs

[1] A workaround involves factors with a complex structure, similar to SPNs, as done for instance in [Mcallester et al., 2004]. Although this approach would be simple enough in the present example, it generally leads to a representation with the disadvantages of SPNs. See Section 1.4 for further discussion.

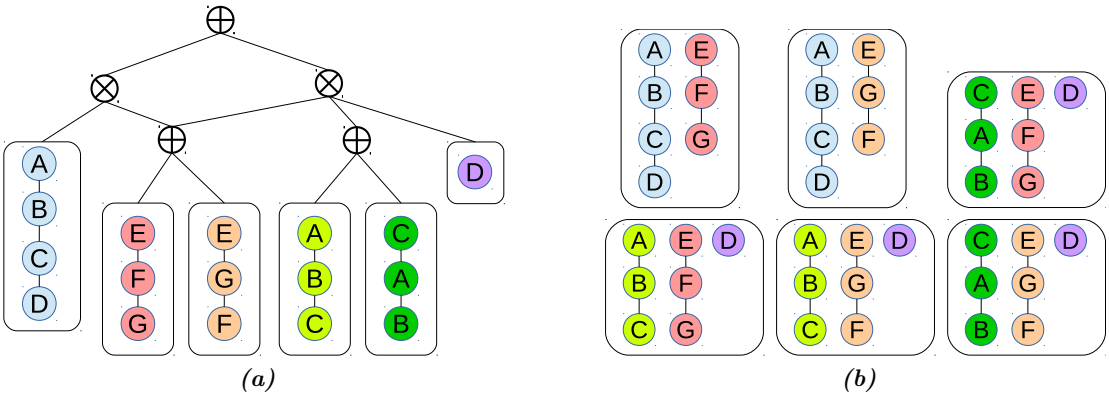


Figure 1.2.1. - A SPN with tree Graphical Models as leaves. 1.2.1a: the SPN. 1.2.1b: components in the corresponding mixture model. Note that each component is a product of tree GMs.

(Fig. 1.1.4b),

$$P(B,C,D|A,Z) = P(Z=0)P(A)P(B|A)P(C|A)P(D|A)P(E|D) \quad (1.1.5a)$$
$$+P(Z=1)P(A)P(B|A)P(C|B)P(D|C)P(E|D). \quad (1.1.5b)$$

However, since some factors, here $P(A)$ and $P(E|B)$, appear in *both* mixture components, this representation generally loses compactness, and computations for inference are unneccessarily *replicated*.

Finally, we may also represent the distribution as SPN (Fig. 1.1.4c) following the procedure outlined above. This represention allows to make explicit the *if* condition due to contextual independence and to share common parts in the two models components. On the other hand, as in the previous example, the probabilistic relations which are easily readable in the other models, are hidden. Furthermore, the SPN representation is considerably more convoluted than the alternatives, and every state of every variable is explicitly represented.

Notice that due to these observations, the representation in Fig. 1.1.1 follows.

1.2. Closing the Gap

Section 1.1.1 showed that SPNs conveniently represent context specific independence and algorithm structures for inference, whereas GMs directly display conditional independency through factorization. The aim of this thesis is to close the gap between the two families. We approach this problems in two subsequent steps, described below. For related approaches in literature, see Section 1.4.

1.2.1. SPNs with Structured Leaf Distributions

A first step to join SPNs and GMs is to consider SPN models having as leaves Graphical Models. Such models can be obtained by replacing the indicator variables at SPN leaves (Fig. 1.1.3b) with GMs, as in the example shown in Fig. 1.2.1a. In general, it can be shown that any form of distribution can be used as a SPN leaf as long as inference with it is tractable at least approximately – this will be made formally clear in Section 3. Indeed, SPNs with structured leaf distributions have been used in several applications of SPNs, such as in Rahman et al. [2014] using tree graphical models as leaves and in Amer and Todorovic. [2015] using Bag-of-Words models.

While this representation allows to employ jointly the GM and SPN representation, thus representing contextual dependences with SPN nodes and conditional independences with graphical models, it does not allow to express all conditional independences in the model with the compact notation of GMs. This holds since the role of GMs is confined to the leaf models, and the parts of internal SPN structure that represent conditional independences still maintain an unnecessary low level representation We introduce the model described in the next section to address this problem.

Furthermore, it is possible to show that this form of SPN corresponds to a mixture model where mixture components are products of the leaf distributions, as shown in Fig. 1.2.1b - this will be formally discussed in Section 3.1.1. This aspect implies, for instance, that the example distribution in Fig. 1.1.4 cannot be expressed compactly by SPNs with leaf GMs, because the part of the model which is a GM but is not a leaf (A, B, C) cannot be represented by a SPN in this form, except by moving it (and replicating it) to the leaves. Once again, the model described in the next section allows to address this problem.

1.2.2. Sum-Product Graphical Models

The second step to close the gap, and the mayor contribution of this thesis, is to introduce a new representation, called *Sum-Product Graphical Model (SPGM)*, that directly *inherits* the favourable traits from both GMs and SPNs.

SPGMs can be seen as an extension of SPN that, along with product and sum nodes as internal nodes, also comprise *variable nodes* which have the same role as usual nodes in graphical models (Fig. 1.2.2a). Alternatively, SPGMs can be seen as an extension of directed GMs by adding sum and product nodes as internal nodes.

- SPGMs exhibit *both* the expressiveness of SPNs and the high level semantics of GMs. Following on the example in Section 1.1.2, The SPGM representing the distribution (1.1.4) is shown by Fig. 1.1.4d. It clearly reveals both the mixture of the two tree-structured subgraphs and the shared components, thus keeping the advantages of both the mixture of tree and SPN representations.

- More generally, every SPGM implements a mixture of trees with shared subparts, as

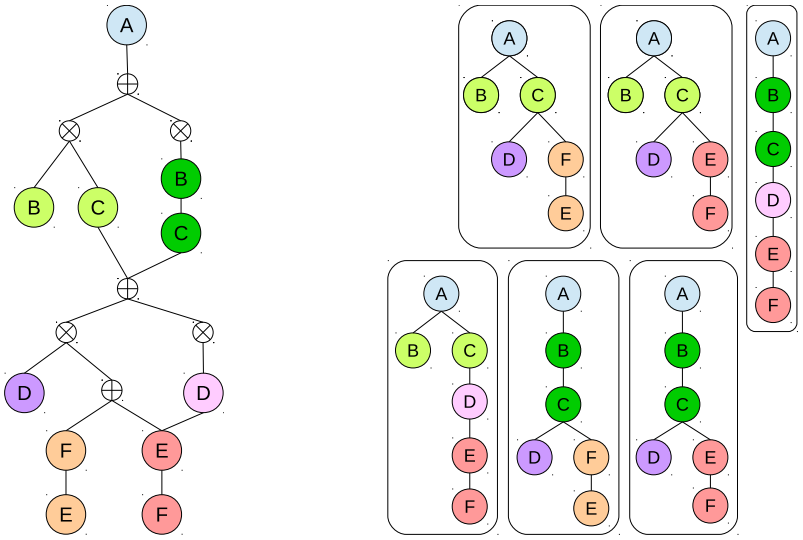


Figure 1.2.2. - A SPGM (left) and the components in the corresponding mixture model (right). Note that each component is a tree GM constructed by connecting subsections of the SPGM. Also note that the mixture components are not factorized, in contrast to Fig. 1.2.1.

in the above example:[2] *Context variables* attached to sum nodes implement *context specific independence* (see Z in Fig. 1.1.4d) and select trees as model components to be combined.

- *Conditional independence* between variables, on the other hand, can be read off from the graph due to D-separation [Cowell et al., 2003]. SPGMs enable to represent in this way *very large* mixtures, whose size grows exponentially with the model size and are thus intractable if represented as a standard mixture model. On the other hand, inference in SPGM has a worst case complexity that merely is *quadratic* in the SPGM size and effectively is quasi-linear only in most practical cases.[3]

- In addition, SPGMs generally provide an equivalent but more compact and high level representation of SPNs, with the additional property that the role of variables with respect to both contextual and conditional independency remains explicit. A compilation procedure through message passing allows to convert the SPGM (Fig. 1.1.4d) into the equivalent SPN (Fig. 1.1.4c) which directly supports computational inference.

- SPGMs are also more expressive than SPNs with GMs as leaf distributions discussed

[2] An extension to mixtures of *junction trees* [Cowell et al., 2003] is straightforward but does not essentially contribute to the present discussion and hence is omitted.

[3] More precisely, the complexity is $O(NM)$, where N is the number of nodes and M is the maximal number of parent nodes, of any node in the model.

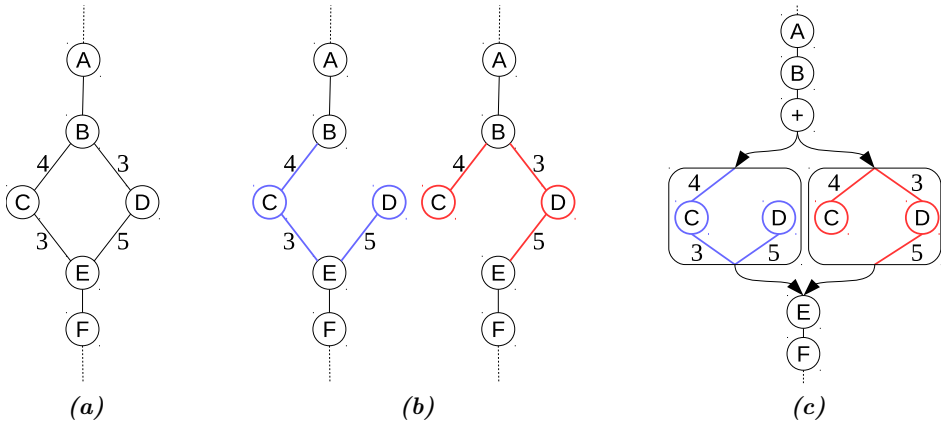


Figure 1.3.1. - Sketch of the structure learning algorithm proposed in this paper. (a) A weighted subgraph on which we compute the maximal spanning trees. (b) Two maximal spanning trees of equal weight to included as mixture components into the SPGM. They differ only in a single edge. (c) The mixture of the two trees represented by sharing all common parts.

previously, since they allow to represent mixture components that are graphs rather than products of the leaves. For instance, Fig. 1.2.2 showns a SPGM and the corresponding mixture model: notice that here the mixture components are trees built by composing parts of the models, rather than factorizations of the leaf distributions as in Fig. 1.2.1.

Due to these properties, the representation in Fig. 1.1.2 follows.

1.3. Structure Learning for SPGMs

Learning the structure of probabilistic models obviously is easier for models with tractable inference than for intractable ones, because any model parameter learning algorithm requires inference as a subroutine. For this reason, tractable probabilistic models and especially SPNs have been widely applied for density estimation [Gens and Domingos, 2013, Rooshenas and Lowd, 2014, Rahman and Gogate, 2016b,a]. It is therefore highly relevant to provide and evaluate a structure learning algorithm for SPGMs, that enable tractable inference as well.

We introduce an algorithm that starts with fitting a single tree in the classical way [Chow and Liu, 1968] and iteratively insert sub-optimal trees that have large weights (in terms of the mutual information of adjacent random variables) and share as many edges as possible with existing tree components. Each insertion is guaranteed not to decrease the global log-likelihood. As a result, all informative edges can be included into the model without compromising computational efficiency, because all shared components are evaluated only once. The former property is not true if a single tree is only fitted [Chow

Table 1.4.1. - Comparison of architectural properties discussed in Section 1.4 for SPGMs and related architectures. We consider the following properties: guaranteed tractable inference (TractInf); same inference efficiency as SPNs (AsSPN); using exponential family factors with no limitation on generality (ExpFam); high level representation of conditional independence (CondInd); representation as a product of factors as in graphical models (FactProd).

Model	TractInf	AsSPN	ExpFam	CondInd	FactProd
SPGM	✓	✓	✓	✓	✓
Graphical Models			✓	✓	✓
Mixtures of Trees [Meila and Jordan, 2000]	✓		✓	✓	✓
Hierarchical Mix. of Trees [Jordan, 1994]	✓		✓	✓	✓
Mix.Markov Model [Fridman, 2003]			✓	✓	✓
Gates [Minka and Winn, 2009]		✓	✓	✓	✓
[Boutilier et al., 1996]			✓	✓	✓
Case-Factor diagrams [Mcallester et al., 2004]	✓	✓			✓
BayesNets local Struct [Chickering et al., 2013]	✓	✓			✓
Learn.Efficient MarkovNets [Gogate et al., 2010]	✓	✓			✓
[Poole and Zhang, 2011]	✓	✓			✓
Value Elimination [Bacchus et al., 2012]	✓	✓			✓
SPN as Bayesian Nets [Zhao et al., 2015]	✓	✓			✓
And/Or Graphs [Dechter and Mateescu, 2007]	✓	✓	✓		
SPN [Poon and Domingos, 2011]	✓	✓	✓		
Arithmetic Circuits [Darwiche, 2002]	✓	✓	✓		
CNets [Rahman et al., 2014]	✓		✓	✓	

and Liu, 1968] whereas working directly with large tree mixtures [Meila and Jordan, 2000] may easily lead to a substantial fraction of redundant computations. The results of a comprehensive experimental evaluation will be reported in Section 5.2.

1.4. Related Work

SPNs with Structured Leaf Distributions Our discussion of SPNs with structured leaf distributions (Section 3.1) connects to papers which use SPNs in this form. In particular, Rahman et al. [2014] use tree graphical models as leaves; Rooshenas and Lowd [2014] use Arithmetic Circuits; Amer and Todorovic. [2015] use Bag-of-Words models. We will derive Expectation-Maximization for such architectures, which is related to a similar derivation by Peharz et al. [2016] on a more restricted case, limited to exponential familiy univariate leaves. This derivation is based on a mixture models interpretation of SPNs which was independently derived by us and Zhao et al. [2016b].

SPGMs The new architecture introduced in thesis, Sum-Product Graphical Models, has several related models due to its connections to both GMs and SPNs. Table 1.4.1 lists and classifies prior work with a similar scope: introducing representations of probability distributions that fill to some extent the gap between GMs and SPNs. The caption of table 1.4.1 lists the properties used to classify related work.

A major aspect of a SPGM is that it encodes a SPN through message passing. This will be made precise formally in Section 4.1.4. As a consequence, SPGMs connect to SPNs and all architectures related to SPNs, which include *Arithmetic Circuits* [Darwiche, 2002], which differ from SPNs only in the way connection weights are represented, and *And/Or Graphs* [Dechter and Mateescu, 2007], which are structurally equivalent to Arithmetic Circuits and thus also to SPNs. We refer to [Dechter and Mateescu, 2007, Section 7.6.1] for a discussion of details. As discussed in Section 1.2, SPGMs encode the computational structure for efficient inference like SPNs and related representations, but also preserve explicitly factorization properties of the underlying distribution due to conditional independence.

SPGMs can represent very large mixtures of trees. In this sense, SPGMs generalize *Hierarchical Mixture of Trees* [Jordan, 1994] by substituting the OR-tree structure used to generate the trees in these models with a more general directed acyclic graph. This allows to not only share part of the trees towards the root but also towards the leaves (Fig. 1.4.1).

SPGMs closely relate to *Gates* [Minka and Winn, 2009] and *Mixed Markov Models* [Fridman, 2003]. These models augment graphical models by a so-called gate unit that implements context specific independence by switching edges on and off depending on the state of some context variables. In this respect, SPGMs may be regarded as Gates - see remark in Section 4.1.3 for corresponding technical aspects. However, unlike Gates and Mixed Markov Models, SPGMs guarantee *tractable exact* inference by construction.

SPGMs related to several further methods that augment GMs by factors with complex structure in order to represent context specific independence [Boutilier et al., 1996, Mcallester et al., 2004, Chickering et al., 2013, Gogate et al., 2010, Bacchus et al., 2012]. These approaches enable to represent product of factors like graphical models. However, the additional model complexity due to contextual independence is simply encapsulated inside the factors, based on models that are equivalent to SPNs and thus exhibit corresponding limitations (Section 1.1.1). In particular, in connection with distributions that combine both conditional and contextual independence, the approaches have to resort to a low-level SPN-like representation. On the other hand, if simpler factors (such as with distributions from the exponential family) were used instead, the model would lose its expressiveness.

Structure Learning in SPNs The structure learning method for SPGMs described in Section 1.3 can be considered as a novel method to learn the structure of a SPN, due to the connection between SPGMs and SPNs. Previous methods for learning the structure of SPNs mostly implement recursive partitioning of the variables into (approximately) independent clusters, to be represented by sum and product nodes [Gens and Domingos, 2013]. Clearly, our greedy method for learning the model structure and parameters based on [Meila and Jordan, 2000] is only locally optimal as well, and focuses on the aforementioned statistical aspects that can be conveniently encoded by SPGMs. However, it employs a completely different perspective based on the relation with mixtures of trees.

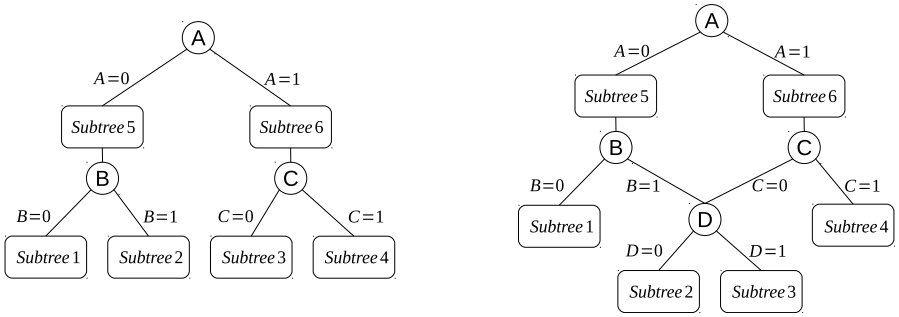


Figure 1.4.1. - Comparison between Hierarchical Mixtures of Trees (HMTs, [Jordan, 1994]) and SPGMs: HMTs are tree-structured (left), while SPGMs enable a directed acyclic structure (right).

1.5. Contributions

With the previous discussion, we are prepared to list the contribution of this thesis, that aim to connect the high level representation of Graphical Models and the efficiency of Sum-Product Networks.

1.5.1. Sum-Product Networks with Graphical Models as Leaves

Learning SPNs with Graphical Model Leaves While learning the internal parameters of SPNs has been amply studied, learning complex leaf distribution is an open problem with only few results available in special cases. The only available approach is discussed in Peharz et al. [2016], and it is limited to exponential familiy univariate leaves. We obtain a derivation of Expectation-Maximization that allows to learn SPNs with *arbitrary* leaf distributions, under mild conditions (Section 3.3.2).

The EM updates have the form of simple weighted maximum likelihood problems, allowing to use any distribution that can be learned with maximum likelihood, even approximately. The algorithm has cost linear in the model size and converges even if only partial optimizations are performed.

We demonstrate this approach with experiments on twenty real-life datasets for density estimation, using tree graphical models as leaves (Section 5.1). Our model outperforms state of the art methods for parameter learning despite consistently using smaller models.

Extending a Graphical Model for Retina Segmentation with SPNs In this contribution (Section 5.3) we describe the first example of using Graphical Models with complex structure as leaf distributions in SPNs. We consider a graphical model for the segmentation of scans of healthy human retinas, presented in Rathke et al. [2014]. This graphical model is tuned to model healthy scans. We enable an extension of this model to local pathological structures in the retina by employing a SPN whose leaves are composed by base models modified by adding pathology-specific shape modifications.

We use the framework of SPNs to find the best combination of modified and unmodified

local models that globally yield the best segmentation. The approach further allows to localize and quantify the pathology. The flexibility and the robustness of our approach is demonstrated by obtaining state of the art results for three different pathologies. In addition, this approach can be easily transfered to new pathologies, as it is designed with no particular pathology in mind and requires no pathological ground truth.

1.5.2. Sum-Product Graphical Models

A Graphical Model Interpretation of SPNs We introduce SPGMs (see 1.3) as an architecture with full expressiveness of SPNs and high level representation of graphical models due to factorization. Table 1.4.1 lists and classifies prior work with a similar scope, showing that SPGMs are the first architecture that enables joinjtly relevant properties of SPNs and GMs, thereby contributing to close the gap between the two fields. We show that SPGMs encode a large mixture of tree GMs with shared subparts, and can be seen as a high level representation of SPNs with richer probabilistic semantics.

Structure Learning We provide a Structure Learning algorithm for SPGMs (see 1.3) based on the novel connection with graphical models. This algorithms starts fitting a single tree in the classical way [Chow and Liu, 1968] and iteratively insert sub-optimal trees that have large weights (in terms of the mutual information of adjacent random variables) and share as many edges as possible with existing tree components. Thus, we model a large mixture of quasi-optimal trees with many shared subparts, enabling efficient inferenfe.

Applications We provide applications to justify and further motivate the introduction of the new architecture. A comprehensive experimental evaluation reported in Section 5.2 shows that the structure learning algorithm for SPGMs obtains state of the art performances in discrete density estimation, despite using an approach radically different from estabilished literature. We also discuss how the architecture can be applied to a flexible range of settings, such as modeling very large mixtures of spanning trees (5.4) and mixtures of image Quadtrees (5.5).

1.6. Organization

This thesis is organized as follows:

- Chapter 2 introduces the notation and methods on which the rest of the thesis is built. We start by providing a compact primer on basic basic probability theory and defining the set of distributions used in the following. Then, we introduce the problem of density estimation through minimization of the Kullback-Leibler divergence. We follow by defining Probabilistic Graphical Models (with a particular attention to tree models) and mixture models, including learning approaches used

in the remainder of the thesis such as the Chow-Liu and Expectation-Maximization algorithms. Finally, we discuss contextual independence and Or trees.

- Chapter 3 presents a comprehensive overview of Sum-Product Networks and introduces our first contribution, Expectation-Maximization for SPNs with structured leaf distributions. We discuss all the main aspects of SPNs, from inference to structure learning, in order both to set the foundations for the following work and to provide a coherent overview of the field, which is currently missing in literature. We start by defining the model and the inference procedure, then we place SPNs between several related architectures. We follow by discussing parameter learning methods and introducing our derivation of Expectation Maximization for SPNs. Finally, we present a detailed overview of influencial structure learning approaches.

- Chapter 4 introduces Sum-Product Graphical Models, the new probabilistic architecture that constitutes the main contribution of this thesis. We start by defining the model and describing the evaluation procedure. Then, we discuss its properties in relation to Graphical Models and Sum-Product Networks, showing that SPGMs represent large mixtures of trees with shared part and that they can be seen as a high level representation of SPNs. Finally, we discuss parameter learning and structure learning for SPGMs, providing a structure learning algorithm that obtains a very large mixture of quasi-optimal tree graphical models with shared parts.

- Chapter 5 contains several example applications of the models and procedures discussed in previous sections, with the aim to further motivate and justify the introduction of our new architecture. We provide five case studies in several settings. First, we test our new Expectation-Maximization algorithm for SPNs and the structure learning algorithm for SPGMs in 20 real life density estimation datasets, reaching state of the art results with SPGMs. Then, we present an application of SPNs using graphical models at leaves, extending a graphical model for the segmentation of images of the human retina to include pathological cases, obtaining excellent empirical results on three publicly available datasets. Finally, we provide two preliminary case studies of SPGMs applied to modeling very large mixtures of spanning trees and mixtures of image Quadtrees (Laferté et al. [2000]).

- Chapter 6 contains concluding remarks and an outlook on future work aimed at extending and providing new applications for the discussed models. We present several possible extensions for SPGMs and a promising application to approximate intractable graphical models within the framework of Tree-Reweighted Message Passing (Wainwright et al. [2002]).

2. Preliminaries

This chapter introduces the tools required for the discussion in the remainder of this thesis. Section 2.1 contains a primer on essential aspects of probability theory and basic probability distributions, followed by an introduction to density estimation. Section 2.3 contains an overview of key points of probabilistic graphical models. Section 2.2 introduces indicator variables and Network Polynomials. Section describes Mixture Models and Expectation-Maximization. Finally, Section 2.5 describes Context Specific Independence.

2.1. Probability Theory

Probability theory is the branch of mathematics that deals with uncertain quantities, associating a real number (the probability) to a random event, which is interpreted as a measure of how likely is the event to happen. Consider a set Ω denoted *sample space,* which represents the set of all possible outcomes of a random experiment. The sample space can be finite (e.g. if the experiment is a coin toss, then $\Omega = \{\text{heads , tails}\}$), countable infinite or uncountable infinite (e.g. for experiments with continuous value). A set $A \subseteq \Omega$ is called an *event.* Let us call Σ the set of all possible events, and consider a function $P : \Sigma \to \mathbb{R}$ defined for each $A \in \Sigma$. $P(A)$ is called a *probability* if the following conditions hold:

- $P(A) \geq 0 \ \forall A \subseteq \Sigma$ - the probability of each event is non negative.
- $P(\Omega) = 1$ - the probability of the whole sample space is 1.
- $P(A \cup B) = P(A) + P(B)$ if $A \cap B = \emptyset$ - the probability of *disjoint* events is the sum of the probabilities of the individual events.

It can be shown that these properties imply that (Σ, Ω) denotes a σ*-algebra* with *measure* P (see Ash and Doleans-Dade [2000]). However, we do not employ this interpretation further in this thesis.

Example 1. Consider as event the outcome of a certain face in rolling a fair die. Associating one number to each face, the sample set is $\Omega = \{1, 2, 3, 4, 5, 6\}$. The set of all possible subsets of Ω is $\Sigma = \{\{1\}, ..., \{6\}, \{1, 2\}, \{1, 3\}, ..., \{1, 2, 3, 4, 5, 6\}\}$. To each $A \in \Sigma$ it is associated a probability. Assuming $P(1) = P(2) = ... = P(6) = \frac{1}{6}$ and following the rules above, then $P(\{1, 2\}) = P(\{1\}) + P(\{2\}) = \frac{1}{3}$ and $P(\{1, 2, 3, 4, 5, 6\}) = 1$.

Random Variables A *random variable* is a variable whose possible values are numerical outcomes of a random phenomenon. A random variable X takes its value from a set of values $\Delta(X)$ called *domain* of X. The set $\Delta(X)$ can either be finite *(discrete random variable)*, countable infinite or uncountable infinite (*continuous random variable*). A relevant case is that of *binary variables,* which take values in the set $\{0, 1\}$.

A set of random variables $\{X_1, X_2, ..., X_K\}$ can be interpreted as a single random variable X taking values in the Cartesian product of the domains of individual variables:

$$\Delta(X) \in \{\Delta(X_1) \times \Delta(X_2) \times \cdots \times \Delta(X_K)\}. \tag{2.1.1}$$

In the following, we use uppercase letters (X) to denote both random variables or sets of random variables, leaving the distinction between the two cases to be read from the context. We use corresponding lowercase letters to denote elements in the respective domain (e.g. $x \in \Delta(X)$).

Probability Density Function A probability density function of a random variable X is a function $f : \Delta(X) \to \mathbb{R}^+$ such that for every $A \subseteq \Delta(X)$ it holds $P(A) = \int_{x \in A} f(x)dx$. Thus, a density function specifies a probability distribution, and in the following we will only specify probabilities through their density function. Note that the axioms of probability imply that $f(x) \geq 0$ and $\int_{x \in \Omega} f(x)dx = 1$.

In the discrete case, the integral is replaced with a sum and it holds that $P(X) = \sum_{x \in X} f(x)$. This implies that $f(x) = P(x)$: the density at X is the probability of event X. Hence, when the distributions considered in this thesis fall in the discrete case we use $P(x)$ in place of $f(x)$.

Basic Rules of Probability We report here basic properties that can be derived from the axioms of probability, referring to the discrete case (the continuous case is obtained replacing sums with integrals).

Marginal Probability. The probability of event A can be computed from the joint probability $P(A, B)$ as follows:

$$P(A) = \sum_{b \in \Delta(B)} P(A, b). \tag{2.1.2}$$

Conditional probability. The conditional probability $P(A|B)$ denotes the probability of event A given that event B happened, and is evaluated as follows:

$$P(A|B) = \frac{P(A, B)}{P(B)}. \tag{2.1.3}$$

Putting together Eq. 2.1.3 and 2.1.2 leads to *Bayes's theorem:*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.1.4}$$

Independent Events. Two events are said independent if their joint probability distribution satisfies:

$$P(A, B) = P(A)P(B). \tag{2.1.5}$$

Conditional Independence. Two events A, B are said conditionally independent given event C if:

$$P(A, B|C) = P(A|C)P(B|C). \tag{2.1.6}$$

Or, equivalently:

$$P(A|B, C) = P(A|C). \tag{2.1.7}$$

Mean. The mean of a function $g(X)$ with respect to density $f(x)$ is written $\mathbb{E}_f(g)$ and is computed as:

$$\mathbb{E}_f(g) = \sum_{x \in \Delta(X)} g(x)f(x)dx. \tag{2.1.8}$$

2.1.1. Inference

Probabilistic queries can be answered by evaluating a probability distribution in a process called *inference*. We report here some standard probabilistic queries, that will be used in the rest of the thesis.

Marginalization involves computing the *marginal probability* $P(X)$ from a joint distribution $P(X, Y)$ by summing out Y:

$$P(X) = \sum_{y \in \Delta(Y)} P(X, y).$$

Note that if Y is a set of variables, the cost of marginalization is in principle *exponential* in the number of variables $|Y|$, since Y takes value in the Cartesian product of the individual domains (Eq. 2.1.1). This property makes marginalization intractable in general multivariate distributions. However, this summation can be computed tractably exploiting the particular structure of some probability distributions, such as tree Graphical Models and Sum-Product Networks (Sections 2.3 and 3.1).

Conditioning involves computing marginals after some variables have been *observed,* i.e. set to a fixed value:

$$P(Y|X = x) = \frac{P(x, Y)}{P(x)} = \frac{P(x, Y)}{\sum_{y \in \Delta(Y)} P(x, y)}.$$

$P(Y|X)$ is called *conditional or a posteriori distribution.* Since conditioning involves marginalization, evaluating the conditional distribution has in principle an exponential

cost.

Maximum A Posteriori (*MAP*) inference involves computing the maximum of the a posteriori distribution $P(Y|X = x)$ obtained after conditioning, that is: $\arg\max_Y P(Y|X)$. Due to the same reasons as in marginalization, also MAP inference has in principle an exponential cost in the number of variables in the model.

2.1.2. Density Estimation

The aim of density estimation is to find a distribution $Q(X)$ that is "similar" (according to some criterion) to another distribution $P(X)$, which is typically available only in the form of a set of independent identically distributed samples. In the following sections we describe the tools used to find such an approximation of $P(X)$ in the rest of the thesis.

Empirical Distribution Let $\{x_i\}_{i=1}^N$be a set of N *independent identically distributed (i.i.d) samples* drawn from the distribution $P(X)$. In this framework, we suppose that the set of samples is the only knowledge we have about the original distribution $P(X)$.

Let us associate an *empirical distribution* to the samples, by assigning to each sample an equal probability $\frac{1}{N}$. The empirical distribution has the following density:

$$P_{emp}(X) = \frac{N_x(X)}{N}\delta(x \in \{x_n\}),$$

where $N_x(X)$ counts the number of times element x appears in $\{x_i\}_{i=1}^N$ and δ is a Kronecker delta. Computing the mean of a function $g(X)$ w.r.t the empirical distribution (Eq. 2.1.8) we obtain the *empirical mean of* $g(X)$*:*

$$\mathbb{E}_{emp}\left[g(x)\right] = \frac{1}{N}\sum_i g(x_i).$$

It is easy to prove that the empirical mean is an unbiased estimator of the mean of g w.r.t the original distribution P, namely:

$$\mathbb{E}_P\left[\mathbb{E}_{emp}\left[g(x)\right]\right] = \mathbb{E}_P\left[g(x)\right].$$

Similarly, one can obtain empirical variance and other higher order moments.

Kullback-Leibler Divergence Approximating a probability distribution P with distribution Q requires defining a measure of the similarity between P and Q. Several measures have been proposed for this task. In this thesis we will employ the widely used Kullback-Leibler (KL) divergence, defined for the discrete domain as:

$$D_{KL}(P|Q) = \sum_x P(x)\ln\frac{P(x)}{Q(x)}. \tag{2.1.9}$$

In the continuous case the sums can be replaced with integrals, and the KL divergence reads:

$$D_{KL}(P|Q) = \int_x P(x) \ln \frac{P(x)}{Q(x)} dx. \tag{2.1.10}$$

We now introduce the Shannon *entropy* $H(P)$ and the *cross entropy* $H(P,Q)$, defined as follows:

$$H(P) = -\sum_x P(x) \ln P(x), \tag{2.1.11}$$

$$H(P,Q) = -\sum_x P(x) \ln Q(x). \tag{2.1.12}$$

By employing these quantities the KL divergence can also be written as:

$$D_{KL}(P|Q) = H(P,Q) - H(P).$$

It can be proven that the cross entropy has a maximum for $P \equiv Q$ almost everywhere (see Ash and Doleans-Dade [2000]). Therefore, the KL divergence has a minimum for $P \equiv Q$ almost everywhere which can be immediately evaluated as 0, which implies that $D_{KL} \geq 0$.

The Kullback-Leibler divergence is a non symmetric measure, in the sense that $D_{KL}(P|Q) \neq D_{KL}(Q|P)$. It is interesting to node that if we aim to find Q as a good approximation of P in the sense that D_{KL} is close to 0, using $D_{KL}(P|Q)$ or $D_{KL}(Q|P)$ creates radically different approximations: using $D_{KL}(P|Q)$ regions of the distribution where $Q(x) \approx 0, P(x) > 0$ are heavily penalized (by $\ln \frac{P(x)}{Q(x)}$), and conversely in $D_{KL}(Q|P)$ regions where $P(x) \approx 0, Q(x) > 0$ are heavily penalized. So, in the first case a good approximation Q will have the primary objective of *covering all the regions* where P is nonzero, and in the second case of *avoiding the regions* where P has low probability (resulting in a sparser distribution).

Maximum Likelihood Let us find an approximation of $P(X)$ by minimizing the KL divergence $D_{KL}(P|Q) = H(P,Q) - H(P)$ with respect to a distribution $Q(X|\theta)$ governed by parameters θ:

$$\begin{aligned} \arg\min_\theta D_{KL}(P(X)|Q(X|\theta)) &= \arg\min_\theta H(P(X), Q(X|\theta)) - H(P) \\ &= \arg\min_\theta H(P, Q(X|\theta)) = \arg\max_\theta \sum_{x \in \Delta(X)} P(x) \log Q(x|\theta)\, dx. \end{aligned} \tag{2.1.13}$$

The quantity $\int_x P(x) \log Q(x|\theta)\, dx$ is called *Log-Likelihood* (LL) and Eq. 2.1.13 is the *Maximum Likelihood* (ML) problem. As we have seen above, maximizing the Log-Likelihood corresponds to *minimizing the KL divergence*.

In the particular case of P specified by an empirical distribution with N samples the ML problem becomes:

$$\arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} P_{emp}(x_i) \ln Q(x_i|\theta), \tag{2.1.14}$$

that can also be written as:

$$\arg\max_{\theta} LL(Q) = \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ln Q(x_i|\theta) = \arg\max_{\theta} \sum_{i=1}^{N} \ln Q(x_i|\theta). \tag{2.1.15}$$

Maximum Likelihood with Weighted Datasets It is often useful to give more importance to some samples over others when computing the Log-Likelihood. In particular, this case arises in the Expectation-Maximization algorithm, discussed in Section 2.4.3. One way to attain this is to define a weighted variant of Log-Likelihood as follows:

$$\arg\max_{\theta} \sum_{i=1}^{N} w_i \ln Q(x_i, \theta), \tag{2.1.16}$$
$$s.t. w_i \in \mathbb{R}, w_i \geq 0.$$

In this case each data sample in the data set is paired with a nonnegative weight that specifies its relevance in the computation of Log-Likelihood. Weight w_i can be interpreted as a soft-count of the number of times a certain sample x_i appears in the dataset.

2.1.3. Basic Probability Distributions

This section contains a brief description of classical probability distributions that will be used as base models for the architectures considered in Sections 3 and 4.

Bernoulli and Categorical Distributions

A Bernoulli distribution is defined over a binary random variable X, associating a probability $p \in \mathbb{R}$ s.t. $0 \leq p \leq 1$ to the state $X = 0$ and $1 - p$ to the state $X = 1$. For our following discussion it is convenient to write this distribution as:

$$P(X) = (1 - p)[X]_0 + p[X]_1, \tag{2.1.17}$$

where $[X]_j$ is an *indicator variable,* which assumes value 1 if $X = j$ and 0 otherwise. A *categorical* distribution is the generalization of Bernoulli distributions to the case of

non-binary discrete variables. It can be written as:

$$P(X) = \sum_{x \in \Delta(X)} \lambda_x [X]_x, \qquad (2.1.18)$$
$$s.t. \lambda_x \geq 0, \sum_x \lambda_x = 1,$$

where $[X]_j = 1$ if $X = x$ and 0 otherwise. It follows immediately that $\lambda_x = P(X = x)$. The mean and MAP state of the Categorical Distribution can be computed by direct evaluation. The Maximum Likelihood solution for parameters λ can be obtained from Eq. 2.1.14 noting that $\max \sum P_{emp}(X) \ln P(X)$ is a cross entropy, which has a maximum when $P_{emp}(X) = P(X)$ (Section 2.1.2). Thus the ML solution (Section 2.1.2) is:

$$\lambda_x^* = \frac{N_x}{N}.$$

Similarly, the solution for weighted Maximum Likelihood (Section 2.1.2) is:

$$\lambda_x^* = \frac{\sum_{i=1}^N w_i \delta(x_n = x)}{N}.$$

Gaussian Distribution

The Gaussian or Normal density function is a widely used model for distributions with continuous variables. Let $X \in \mathbb{R}^m$ be a vector of continuous random variables. A normal distribution with parameters $\mu \in \mathbb{R}^m, \Sigma \in \mathbb{R}^{m \times m}$ is defined as:

$$\mathcal{N}(X|\mu,\theta) = \frac{1}{\sqrt{(2\pi)^k det(\Sigma)}} \exp\left(-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)\right),$$

where Σ is a symmetric positive definite matrix and $det(\Sigma)$ denotes the determinant of Σ. It is possible to prove that μ is both the mean and the mode of $\mathcal{N}(X|\mu,\theta)$. Computing the *marginal* distribution over a subset of variables is straightforward and efficient in Gaussian distributions: one only needs to drop the rows and columns corresponding to the variables that are marginalized out from the mean vector μ and the covariance matrix Σ. Other operations such as conditioning and MAP computation can be obtained efficiently in close form - we omit them here because they are not used in the thesis. The ML solution parameters can be found as follows (see Murphy [2012, 11.4.2]):

$$\mu^* = \frac{1}{N} \sum_{i=1}^N x_i,$$
$$\Sigma^* = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)(x_i - \mu)^T.$$

Similarly, the solution for weighted Maximum Likelihood (Section 2.1.2) is:

$$\mu^* = \frac{\sum_{i=1}^N w_i x_i}{\sum_{i=1}^N w_i},$$
$$\Sigma^* = \frac{\sum_{i=1}^N w_i \left(x_i - \mu\right)\left(x_i - \mu\right)^T}{\sum_{i=1}^N w_i}.$$

Exponential Family

The *exponential family* is a class of distributions that encompasses and generalizes many commonly used distributions, and includes the distributions seen above. A distribution is said to belong to the exponential family if it can be written as:

$$P(X|\eta) = h(X)g(\eta)\exp(\eta^T u(X)),$$

where $\eta \in \mathbb{R}^m$ are the *natural parameters,* $g(\eta) : \mathbb{R}^m \to \mathbb{R}$ is a *normalization coefficient*, and $u(X) : \Delta(X) \to \mathbb{R}^m$ is a given function.

A crucial property of exponential families is that the Maximum Likelihood problem is *convex* and can be found as the solution of the following "moment matching" equation:

$$-\nabla \ln g(\eta) = \mathbb{E}_{emp}[u(x)],$$

where ∇ denotes the gradient w.r.t. natural parameters η and $\mathbb{E}_{emp}$ is the empirical mean (Section 2.1.2). Thus, provided that the gradient of the partition function can be computed, once can always find the distribution's ML solution by moment matching.

2.2. Indicator Variables and Network Polynomials

We formally define indicator variables, that were already used in Eq. 2.1.17 and 2.1.18.

Definition 2.2.1. Assignment of Indicator Variables. Let $Y \subseteq X$ be a subset of variables to which the values $y \in \Delta(Y)$ are assigned: $X_v = y_v$, $\forall X_v \in Y$. Based on the assignment y, we associated with every variable $X_s \in X$ and every value $i \in \Delta(X_s)$ the *indicator variable* $[X_s]_i \in \{0, 1\}$ defined by

$$[X_s]_i = \begin{cases} 1 & \text{if } (X_s \in Y \text{ and } y_s = i) \text{ or } (X_s \notin Y), \\ 0 & \text{otherwise.} \end{cases} \tag{2.2.1}$$

We can also define, for compactness of notation, indicator variables over sets, which are simply the product of all indicator variables in the set.

Definition 2.2.2. Indicator Variables over Sets. Let $Y \subseteq X$ be a subset of variables to which the values $\{y_s\} \in \Delta(Y)$ are assigned. that $X_s = y_s$, $\forall X_s \in Y$. The indicator variable over set Y, written $[Y]_y$, denotes the product of all indicator variables in the set:

$$[Y]_y = \prod_{X_s \in Y} [X_s]_{y_s}. \tag{2.2.2}$$

Indicator variables can be used to represent distributions in the form of a Network Polynomial, defined as follows.

Definition 2.2.3. Network Polynomial. Let $P(X)$ be a distribution over a set of discrete variables X. The network polynomial of $P(X)$ is defined as:

$$\sum_{x \in \Delta(X)} P(x)[X]_x, \tag{2.2.3}$$

where $[X]_x$ is an indicator variable over set X as in Definition 2.2.2.

The network polynomial transforms the distribution into a polynomial in the indicator variables $[X]_x$, where terms $P(x)$ act as coefficients. Inference in the distribution $P(X)$ can be performed by evaluating the network polynomial assigning the indicator variables according to Definition 2.2.1.

Example 2. Consider a distribution of discrete random variables A, B, C, D in the form:

$$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|A), \tag{2.2.4}$$

which corresponds to Fig. 1.1.3a. Uppercase letters A, B, C, D denote random variables and corresponding lowercase letters a, b, c, d denote values in their domains $\Delta(A)$, $\Delta(B)$, $\Delta(C)$, $\Delta(D)$. We write $\sum_{a,b,c,d}$ for the sum over the joint domain $\Delta(A) \times \Delta(B) \times \Delta(C) \times \Delta(D)$. The network polynomial is as follows:

$$\sum_{a,b,c,d} P(a, b, c, d)[A]_a[B]_b[C]_c[D]_d. \tag{2.2.5}$$

Here $P(a, b, c, d)$ denotes the *value* of P for assignment $A = a, B = b, C = c, D = d$, and $[A]_a, [B]_b, [C]_c, [D]_d \in \{0, 1\}$. The distribution can be evaluated by assigning the indicator variables: for instance, to compute the partition function all indicator variables of (2.2.5) are set to 1, and to compute the marginal probability $P(A = 1)$ one sets $[A]_1 = 1, [A]_0 = 0$ and all the remaining indicators to 1.

2.3. Graphical Models

Probabilistic Graphical Models (GMs) are a family of models that encode probability distributions through a graph defining dependences between the variables. This section

provides an introduction to GMs limited to the key aspects needed in the remainder of the thesis.

2.3.1. Probabilistic Graphical Models

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with vertex set $\mathcal{V} = \{1, 2, \ldots, N\}$ and edge set $\mathcal{E}$.[1] We associate to each vertex $s \in \mathcal{V}$ a discrete random variable X_s taking values in the finite domain $\Delta(X_s)$, and $X = \{X_s\}_{s\in\mathcal{V}}$ denotes the set of all variables of the model, taking values in the Cartesian product $\Delta(X) := \Delta(X_1) \times \Delta(X_2) \times \cdots \times \Delta(X_N)$.

Undirected GMs An *Undirected Graphical Model* on an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprises unary factors $\varphi_s(X_s)$ for each vertex $s \in \mathcal{V}$ and pairwise factors $\varphi_{s,t}(X_s, X_t)$ $P_{s,t}(X_t|X_s)$ for every undirected edge $(s,t) \in \mathcal{E}$, and encodes the distribution

$$P(X) = \frac{1}{Z} \prod_{s\in\mathcal{V}} \varphi_s(X_s) \prod_{(s,t)\in\mathcal{E}} \varphi_{s,t}(X_s, X_t), \tag{2.3.1}$$

$$Z = \sum_{x\in\Delta(X)} P(X). \tag{2.3.2}$$

Directed GMs A *Directed Graphical Model (Directed GM)* on a directed graph $\mathcal{G}$ comprises conditional probabilities $P_{s,t}(X_t|X_s)$ for every directed edge $(s,t) \in \mathcal{E}$ and unary probabilities $P_r(X_r)$ for each vertex $r \in \mathcal{V}$ with no parent. We denote by $pa(s)$ the parents of s in $\mathcal{G}$: $pa(s) = \{r \in \mathcal{V} \colon (r,s) \in \mathcal{E}\}$. With this notation, a directed GM encodes the distribution

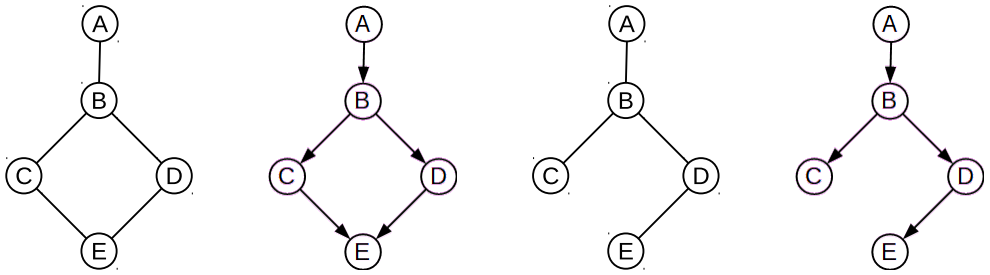
$$P(X) = \prod_{r\in\mathcal{V}\colon pa(r)=\emptyset} P_r(X_r) \prod_{(s,t)\in\mathcal{E}} P_{s,t}(X_t|X_s). \tag{2.3.3}$$

Example 3. Some example directed and undirected graphical models are shown in Fig. 2.3.1.

Remark 4. Notice that both directed and undirected GMs entail a representation as a *product of factors* taken over unary variables or pairs of variables.

Conditional Independence via D-Separation Graphical models conveniently encode conditional independence properties of a distribution. Conditional independence between variables in GMs are induced from the graph by D-separation (see, e.g., Pearl [2000]). We do not employ the general concept of D-separation except in the particular case of tree GMs, discussed in Section 2.3.2. For tree graphical models, the D-separation criterion becomes particularly simple: if the path between variables A and B contains C, then A is conditionally independent from B given C.

[1] We use the same symbol for directed and undirected graphs not to clutter the notation. The distinction in roles will be clear from the context



(a) Undirected GM. *(b)* Directed GM. *(c)* Undirected tree GM *(d)* Directed tree GM.

Figure 2.3.1. - Graphical representation of directed and undirected probabilistic graphical models.

Inference in General GMs Marginalization and Maximum a Posteriori (MAP) inference in general directed GMs has a cost that is *exponential* in the treewidth of the triangulated graph obtained by moralization of the original graph, and thus is *intractable* for graphs with cycles of non trivial size [Cowell et al., 2003, Diestel, 2006]. This has a notable exception in tree GMs, in which inference can be computed efficiently with message passing (Section 2.3.2).

2.3.2. Tree Graphical Models

A *Directed Tree Graphical Model* is a GM where the underlying graph $\mathcal{G} = \mathcal{T}$ is a directed tree $\mathcal{T}$ with root r. Each vertex s has at most one parent $pa(s)$ in the tree, hence the distribution (2.3.3) reads

$$P(X) = P_r\left(X_r\right) \prod_{s \in \mathcal{V}:\ pa(s) \neq \emptyset} P_{pa(s),s}\left(X_s | X_{pa(s)}\right). \tag{2.3.4}$$

An *Undirected Tree Graphical Model*, in which the underlying graph $\mathcal{G} = \mathcal{T}$ is an undirected tree, represents the following distribution:

$$P(X) = \prod_{t \in \mathcal{V}} P_t(X_t) \prod_{(s,t) \in \mathcal{E}} \frac{P_{st}(X_s, X_t)}{P_s(X_s) P_t(X_t)}. \tag{2.3.5}$$

It can be proven that in case of trees the directed or undirected parameterization are equivalent and simply entail a re-parametrization, since the undirected form it is obtained from 2.3.4 by multiplying and dividing by $\prod_{s \in \mathcal{V}} P_s(X_s)$ and employing the equality $P_{s,t}\left(X_t | X_s\right) = P_{st}\left(X_t, X_s\right) / P_s(X_s)$.

Note that this implies that in the undirected representations tree graphical models allow the explicit use of marginal probabilities P_s and P_{st} rather than using factors as in Eq. 2.3.3. In addition, the distribution is normalized, thus there is no need for a normalization coefficient.

Inference Marginalization and Maximum a Posteriori (MAP) inference in tree GMs can be computed efficiently with *message passing*. Let $Y \subseteq X$ be a set of observed variables with assignment $y \in \Delta(Y)$. Let $[X_s]_j,\ s \in \mathcal{V},\ j \in \Delta(X_s)$ denote indicator variables due to Definition 2.2.1. Node t sends a message $\mu_{t\to s;j}$ to its parent s for each state $j \in \Delta(X_s)$ given by

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)[X_t]_k \prod_{(t,q)\in\mathcal{E}} \mu_{q\to t;k}. \tag{2.3.6}$$

Setting $Z = X \setminus Y$ and $x = (y, z)$, marginal probabilities $P(Y = y) = \sum_{z\in\Delta(z)} P(y, z)$ can be computed using the distribution (2.3.4) by first setting the indicator variables according to the assignment y (Definition 2.2.1), then passing messages for every node in reverse topological order (from leaves to the root), and finally returning the value of the root message. MAP queries are computed in the very same way after substituting sums with the max operation in Eq. (2.3.6). Since message passing in trees only requires computing one message per each node, the procedure has complexity $O(|\mathcal{V}|\Delta_{max}^2)$, where $\Delta_{max} = \max\{|\Delta(X_s)|\colon s \in \mathcal{V}\}$ is the maximum domain size. As a consequence, tree GMs enable *tractable* inference.

2.3.3. Junction Trees

Junction Trees are an extension of tree graphical models that allows to have multiple variables at each node. Formally, a *Junction Tree (JT)* $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ is a directed rooted tree which satisfies the following *properties*:

1. Each node $s \in \mathcal{V}$ is associated to a *set* of variables $X_s \subseteq X$.
2. A factor $\varphi_{st}\left(X_s, X_t\right)$ is associated to each vertex $(s, t) \in \mathcal{E}$.
3. Running intersection: for each pair of nodes $v \in \mathcal{V}, u \in \mathcal{V}$ such that $X_v \cap X_u = R$, it holds that $R \subseteq X_k$ for all nodes k in the path between u and v.

Similarly to tree GMs, inference in a JT can be computed with message passing. Node t sends a message $\mu_{t\to s;j}$ to its parent s for each state $x_s \in \Delta(X_s)$ given by

$$\mu_{t\to s;x_s} = \sum_{y\in\Delta(X_t\setminus X_s)} \varphi_{st}\left(y, x_s\right)[X_t \setminus X_s]_y \prod_{(t,q)\in\mathcal{E}} \mu_{q\to t;y} \tag{2.3.7}$$

where $[X_t \setminus X_s]_x$ denotes indicator variables for variable sets due to Definition 2.2.2.The marginal probability $P(Y = y) = \sum_{z\in\Delta(z)} P(y, z)$ is computed by assigning the indicator variables according to an evidence set Y with assignment $y \in \Delta(Y)$ (as in Section 2.3.2) and evaluating messages for all nodes from the leaves to the root allows to compute . Similarly, MAP inference can be computed substituting sums with max.

From the sum over state $\sum_{y\in\Delta(X_t\setminus X_s)}$ in Eq. 2.3.7, and considering that one message is passed for each node, it is immediate that message passing in Junction Trees has a cost

$$O(|\mathcal{V}|\exp(W)), \tag{2.3.8}$$

where $|\mathcal{V}|$ is the number of variables and W is the size of the largest set of variables in the Junction T*ree.*

Inference in GMs through Junction Trees Junction Trees are mainly used as an intermediate architecture to compute inference in graphical models. Inference in general graphical models can be computed by converting the graph into a Junction Tree and passing message in the resulting tree, with a complex procedure described e.g. in Aji and McEliece [2000].

Crucially, the conversion procedure generates a JT in which the minimum variable set size at any node (W in Eq. 2.3.8) is lower limited by the *treewidth* of the graph. Hence, inference in the JT corresponding to a given GM has a cost *exponential in the treewidth* of the graph, as discussed in Section 2.3.1.

2.3.4. Learning Tree Graphical Models

Finding the structure and parameters of a graphical model with minimal KL divergence w.r.t. a given empirical distribution is, in general, NP-hard (Chickering [1996]). However, in case of tree GMs the Chow-Liu Tree algorithm (Chow and Liu [1968]) allows to solve this problem requiring only $O(N|\mathcal{V}|^2 + |\mathcal{V}|^2 \ln |\mathcal{V}|)$ operations . We report its derivation here, in preparation for our extension of the algorithm in Section 4.2.

First, we can write the Log-Likelihood (Eq. 2.1.15) using the undirected tree representation (Eq. 2.3.5) as

$$\sum_{i=1}^{N} \ln \prod_{t\in\mathcal{V}} P_t(x_t^i) \prod_{(s,t)\in\mathcal{E}} \frac{P_{st}(x_s^i, x_t^i)}{P_s(x_s^i)P_t(x_t^i)} = \sum_i \sum_{t\in\mathcal{V}} \ln P_t(x_t^i) + \sum_i \sum_{(s,t)\in\mathcal{E}} \ln \frac{P_{st}(x_s^i)}{P_s(x_s^i)P_t(x_t^i)}, \tag{2.3.9}$$

where x_t^i is the value of X_t in sample x^i. Since $P_t(x_t^i)$ assumes the same value all samples i such that $x_t^i = k$ (and similarly $P_s(x_s^i)$), we can pass from to a sum over states rather than over samples, by introducing terms N_{tk} counting the number of times node t is in state k in the dataset, and terms N_{stjk} counting the number of times node s is in state j and node t is in state k.

Let us now consider a parameter vector θ such that $\theta_{st} \in \theta, \theta_s \in \theta$ are parameters governing respectively probabilities P_s and $P_{s,t}$. We compute ML by maximizing over the optimal tree structure $\mathcal{T}$ and parameters θ, as follows:

$$\arg\max_{\mathcal{T},\theta} \sum_{t\in\mathcal{V}} \sum_{k\in\Delta(X_t)} N_{tk} \ln P_t(k) + \sum_{(s,t)\in\mathcal{E}} \sum_{k,j\in\Delta(X_s,X_t)} N_{stjk} \ln \frac{P_{st}(k,j)}{P_s(k)P_t(j)}.$$

Since the maximization acts over a disjoint set of parameters for each edge $(s,t) \in \mathcal{E}$ and vertex $t \in \mathcal{V}$, the max over θ can be divided in terms θ_{st} and brought inside the sums. We can now express N_{tk} as empirical distribution by normalizing by $\bar{P}_{st}(k,j) = N_{st;jk}/N_{s;j}$,

as $\bar{P} = N_{t;k}/N$. Denoting with $\bar{P}$ the empirical distribution, the expression becomes:

$$\max_{\mathcal{T}} \left(\sum_{t\in\mathcal{V}} \max_{\theta_t} \sum_{k\in\Delta(X_t)} N\bar{P}_t(k) \ln P_t(k) + \sum_{(s,t)\in\mathcal{E}} \max_{\theta_{s,t}} \sum_{k,j\in\Delta(X_s,X_t)} N\bar{P}_{st}(k,j) \ln \frac{P_{st}(k,j)}{P_s(k)P_t(j)} \right).$$

Now, we can divide by N since doing so does not change the arg max. Let us remind that the cross entropy $H(P,Q) = \sum_x P(x) \ln Q(x)$ (Eq. 2.1.12) has a minimum for $P = Q$, which in this case means matching the empirical distribution. By this rule we can compute each $\max_{\theta_{s,t}}$ obtaining:

$$\begin{aligned} &\max_{\mathcal{T}} \left(\sum_{t\in\mathcal{V}} \sum_{k\in\Delta(X_t)} \bar{P}_t(k) \ln \bar{P}_t(k) + \sum_{(s,t)\in\mathcal{E}} \underbrace{\sum_{k,j\in\Delta(X_s,X_t)} \bar{P}_{st}(k,j) \ln \frac{\bar{P}_{st}(k,j)}{\bar{P}_s(k)\bar{P}_t(j)}}_{\mathbb{I}(X_s,X_t)} \right) \\ =&\sum_{t\in\mathcal{V}} \sum_{k\in\Delta(X_t)} \bar{P}_t(k) \ln \bar{P}_t(k) + \max_{\mathcal{T}} \sum_{(s,t)\in\mathcal{E}} \mathbb{I}(X_s, X_t). \end{aligned}$$

Here we noted that the first term is independent from the tree structure, thus we moved $\max_{\mathcal{T}}$ inside the sum. The term $\mathbb{I}(X_s, X_t)$ is called *mutual information*. Thus the tree structure maximizing the Log-Likelihood is the maximum spanning tree where the edge weights are given by the mutual information $\mathbb{I}$. The MST can be computed wth cost $O(|\mathcal{V}| \ln |\mathcal{V}|)$ using classical Prim's or Kruskal's alorithm.

Weighted Maximum Likelihood

It is possible to find the optimal tree maximizing a weighted ML problem as in Eq. 2.1.16 by modifying slightly the Chow-Liu algorithm. The only difference in the whole procedure is that the data counts have to be computed including the weights, as follows:

$$\begin{aligned} N_w &= \sum_{n=1}^{N} w_n, \\ N_{tk} &= \sum_{n=1}^{N} \delta(x_t^n = k) \frac{w_n}{N_w}, \\ N_{stjk} &= \sum_{n=1}^{N} \delta(x_s^n = j)\delta(x_t^n = k) \frac{w_n}{N_w}. \end{aligned}$$

2.4. Mixture Models

2.4.1. Basic Definitions

A mixture model with K component is a distribution which can be written as a weighted sum of distributions $\{P_k(X|\theta_k)\}_{k=1}^K$, as follows:

$$\begin{aligned} P(X|\theta) &= \sum_{k=1}^{K} \lambda_k P_k(X|\theta_k), \\ s.t. &\lambda_k \in \mathbb{R}, \lambda_k \geq 0 \, \forall k \in \{1, 2, \ldots, K\}, \\ &\sum_{k=1}^{K} \lambda_k = 1. \end{aligned} \tag{2.4.1}$$

Here, terms $\{\lambda_k\}_{k=1}^K$ are denoted mixture *coefficients* and terms $\{P_k(X|\theta_k)\}_{k=1}^K$ are denoted mixture *components*; each component P_k is governed by parameters $\theta_k \in \theta$. It is easy to show that $P(X)$ is normalized as long as each mixture component is normalized. Inference in a mixture model has cost $O(KC)$ where C is an upper bound to the cost of evaluating mixture components, since it involves summing the results of inference procedures in each component individually. For instance, the partition function is evaluated as follows:

$$\sum_{x \in \Delta(X)} P(X) = \sum_{x \in \Delta(X)} \sum_{k=1}^{K} \lambda_k P_k(X) = \sum_{k=1}^{K} \lambda_k \left(\sum_{x \in \Delta(X)} P_k(X) \right).$$

The presence of the sum $\sum_{k=1}^K$ implies that mixture models *cannot* be written in factorized form even when the single mixture components are *factorized* (this is the case for example of GMs, see Remark 4). This entails that Maximum Likelihood approaches become intractable even when ML is tractable for the single mixture components.

This problem can be seen by assuming that each mixture component is in the exponential family, and thus its ML solution can be found by moment matching (Section 2.1.3). Evaluating the Maximum Likelihood we obtain:

$$\max_{\theta} \sum_{i=1}^{N} \log P(x_i|\theta) = \sum_{i=1}^{N} \log \max_{\theta} \sum_{k=1}^{K} \lambda_k P_k(X).$$

The term $\sum_{k=1}^K \lambda_k P_k(X)$ is not in the exponential family, being a sum of exponential families. Hence, ML cannot be computed by moment matching.

In general, it can be shown that finding ML solutions for mixture models is an NP hard problem (Murphy [2012]). A local optimum of ML for mixture models can be found using the Expectation Maximization algorithm, discussed in Section 2.4.3.

2.4.2. Interpretation as Latent Variable Models

A mixture model can also be expressed through a *hidden (or latent) variable* Z where $\Delta(Z) = \{1, 2, \ldots, K\}$ by writing:

$$P(X, Z) = \prod_{k=1}^{K} \left(\lambda_k T_k\left(X\right)\right)^{\delta(Z=k)}. \tag{2.4.2}$$

Applying the basic probability rules, it holds that $P(X) = \sum_{z\in\Delta(Z)} P(X, Z)$, $P(Z = k) = \lambda_k$ and $P(X|Z = k) = \lambda_k T_k\left(X\right)$.

Remark 5. Considering a mixture of trees $P(X|Z = k) = \lambda_k T_k\left(X\right)$, different values of Z entail different independences due to different tree structures. Hence, mixtures of trees can represent context specific independence with context variable Z as defined in Section 2.5.

2.4.3. Expectation Maximization

Expectation Maximization (EM) is a widely used method for finding Maximum Likelihood solutions for models with latent variables (see e.g. Murphy [2012, 11.4]). For the sake of this thesis it is of interest to apply EM to learn mixture models through their latent variable interpretation (Eq. 2.4.2).

Derivation of EM Let us consider a distribution with latent variable Z governed by parameters θ, as follows:

$$P(X|\theta) = \sum_{z\in\Delta(Z)} P(X, Z|\theta),$$

and let us write down the Maximum Likelihood (Eq. 2.1.15):

$$\arg\max_{\theta} LL(X|\theta) = \arg\max_{\theta} \sum_{i=1}^{N} \ln P(x_i|\theta) = \arg\max_{\theta} \sum_{n=1}^{N} \ln \sum_{z_n\in\Delta(Z)} P\left(x_n, z_n|\theta\right). \tag{2.4.3}$$

The presence of the sum $\sum_{z_n\in\Delta(Z)}$ entails that the argument of ln does not factorize. Hence, the formula cannot be further simplified due to factorization, and the maximization is then intractable in the general case. To address this problem we introduce a distribution $Q(Z|\theta) = \prod_{i=1}^{N} Q_i(Z_i|\theta)$; the shape of Q shall be determined later. With simple algebraic manipulations we can rewrite the Log-Likelihood as follows:

$$LL(X|\theta) = \sum_{i=1}^{N} \ln P(x_i|\theta) = \mathcal{L}(Q(Z)|\theta) + \sum_{i=1}^{N} D_{KL}\left(Q_i(Z)|P\left(Z|x_i,\theta\right)\right), \tag{2.4.4}$$

$$\mathcal{L}(Q(Z)|\theta) = \sum_{i=1}^{N} \sum_{z\in\Delta(Z)} Q_i(z) \ln \frac{P(x_i,z|\theta)}{Q(z)}. \tag{2.4.5}$$

Note that since $D_{KL} \geq 0$ then $LL(X|\theta) \geq \mathcal{L}(Q(Z)|\theta)$. But we can also write, further rearranging the formula:

$$\mathcal{L}(Q|\theta) = -\sum_{i=1}^{N} D_{KL}\left(Q_i\left(Z\right)|P\left(Z|x_i,\theta\right)\right) + LL(X|\theta). \tag{2.4.6}$$

One can find a local maximum of Eq. 2.4.4 with a coordinate ascent method. We propose to show later that this procedure is guaranteed to increase the LL at each step. Suppose that an initial parameter configuration θ_{old} is given. The following steps are performed:

1. *E step.* Find the maximum $Q^* = \arg\max_Q \mathcal{L}(Q|\theta_{old})$ w.r.t. the functional Q *only.* The solution can be found as $Q_i^*(Z) = P(Z|x_i,\theta_{old})$ by direct inspection of Eq. 2.4.6.

2. *M step.* Find the maximum $\theta_{new} = \arg\max_\theta \mathcal{L}(Q^*|\theta)$:

$$\begin{aligned}
\theta_{new} &= \arg\max_\theta \mathcal{L}(Q^*|\theta) \\
&= \arg\max_\theta \sum_{i=1}^{N} \sum_{z_i\in\Delta(Z_i)} P(z_i|x_i,\theta_{old}) \ln \frac{P(x_n,z_n|\theta)}{P(z_n|x_n,\theta_{old})} \\
&= \arg\max_\theta -\sum_{n=1}^{N} \sum_{z_i\in\Delta(Z_i)} P(z|x_n,\theta_{old}) \ln P(x_n,z|\theta).
\end{aligned}$$

3. Repeat steps $1, 2$ until Log-Likelihood convergence.

The EM algorithm is summarized in Table 2.4.1.

Convergence to a Local Maximum

By simple algebraic manipulations, and noting that $\mathcal{L}(Q^*|\theta_{old}) = LL(\theta_{new})$ since $Q^* = P\left(Z|X,\theta_{old}\right)$, it follows that:

$$LL(\theta_{new}) \geq \mathcal{L}(Q^*|\theta_{new}) \geq \mathcal{L}(Q^*|\theta_{old}) = LL(\theta_{old}).$$

$$\text{E-step} \quad \text{compute } P\left(z|x_n, \theta_{old}\right). \tag{2.4.7}$$

$$\text{M-step} \quad \theta_{new} = \arg\max_{\theta} \mathcal{Q}\left(\theta, \theta_{old}\right). \tag{2.4.8}$$

$$\text{where} \quad \mathcal{Q}\left(\theta, \theta_{old}\right) = \sum_{n=1}^{N} \sum_{z \in \Delta(Z)} P\left(z|x_n, \theta_{old}\right) \ln P\left(x_n, z|\theta\right). \tag{2.4.9}$$

Table 2.4.1. - The EM algorithm iterates E and M steps until convergence.

where the last equality holds due to Eq. 2.4.6:

$$\mathcal{L}(Q^*|\theta_{old}) = -\sum_{i=1}^{N} \underbrace{D_{KL}\left(Q_i^*\left(Z\right)|P\left(Z|x_i, \theta\right)\right)}_{0} + LL(\theta_{old}) = LL(\theta_{old}).$$

Hence, the Log-Likelihood is guaranteed to increase at each EM step, and the algorithm converges to a local maximum. This convergence property acts as a powerful debugging tool for proof checking the correctness of the algorithm in practical implementation.

Efficient Application of EM

Eq. 2.4.8 can be easily maximized whenever two conditions are satisfied:

1. $P(X, Z|\theta)$, including the latent variables, *factorizes as* $P(X, Z|\theta) = \prod_{n=1}^{N} P(x_n|Z_n, \theta)$ (one term per each sample).

2. The parameters θ are disjoint for each mixture component: $P(X|Z = k, \theta) = P(X|Z = k, \theta_k)$ s.t. $\{\theta_k\}_{k=1}^{|\Delta(Z)|}$ form a partition of θ. In this case, the maximizations can be performed independently for each term in the product.

When conditions 1 and 2 above are satisfied, Eq. 2.4.8 simplifies as

$$\begin{aligned}\max_{\theta} \mathcal{Q}(\theta) &= \max_{\theta} \sum_{n=1}^{N} \sum_{z \in \Delta(Z)} P\left(z|x_n, \theta_{old}\right) \ln P\left(x_n, z|\theta\right) \\ &= \sum_{n=1}^{N} \max_{\theta_n} \sum_{z \in \Delta(Z)} P\left(z|x_n, \theta_{old}\right) \ln P\left(x_n, z|\theta_n\right).\end{aligned}$$

Hence, separate maximizations are required for each sample. Noting that this problem is, for fixed θ_{old}, an instance of weighted maximum likelihood (Section 2.1.2), we have seen that in many practical cases this can be done tractably. This is the case, for instance, for mixture models where mixture components are in the exponential families. In particular,

as we have seen for mixtures of Gaussians (Sections 2.1.3) and mixtures of trees (Section 2.3.2) there are efficient close form ML solutions.

2.5. Context Specific Independence

We have seen that conditional independence relations forms the basis of graphical models. However, not all forms of independence can be captured efficiently by conditional independence. We consider here the case in which independences hold given a certain assignment of a subset of variables, denoted as contextual (or context specific) independence.

Definition 2.5.1. *Contextual Independence.* Variables A and B are contextually independent given Z and context $c \in \Delta(Z)$ if $P(A|B, Z = c) = P(A|Z = c)$.

It can be shown that Graphical Models, being designed to model conditional independences between variables according to the graph structure, cannot compactly represent distributions with contextual independence. This aspect was analyzed in depth in Boutilier et al. [1996]. For the sake of our discussion, it is sufficient to have an intuitive grasp of the inability of graphical models to express contextual independence through the example of Section 1.1.2. Since exploiting contextual independence allows to greatly simplify inference in distributions where it is present, this preoperty mainly motivates the class of distributions described in the following sections.

3. Sum-Product Networks

Sum-Product Networks (SPNs) are a family of probabilistic models with two crucial properties: firstly, inference has a cost linear in the model size and is therefore always tractable; secondly, they enable an efficient representation of distributions with context specific independence, which cannot be modeled tractably by graphical models.

SPNs form the basis of our subsequent discussion, since the model proposed in Chapter 4.1 merges and integrates aspects of SPNs and graphical models. It is therefore necessary to provide a detailed overview of these models.

However, since this research field is quite new and most publications appeared in conference proceedings, an organized discussion of relevant aspect of SPNs is not available. A partial organization of this material was done in Peharz [2015], which still does not include several approaches more recent than 2015 and does not include structure learning. There is therefore the need of a comprehensive description of the essential aspects of SPNs in order to proceed with our discussion.

This chapter provides such comprehensive discussion, with the aim to form a solid base for the following sections. In addition, it also introduces one of the main contributions of this thesis: a novel derivation of Expectation Maximization for SPNs that easily extends to leaf models with arbitrary structure and to weight sharing situations (Section 3.3.2).

Structure of the Chapter SPNs are introduced formally in section 3.1. Section 3.1.1 describes how inference in SPN is performed, and introduces properties of SPNs, such as the probabilistic interpretation of SPNs and the inference procedure. Section 3.2 analyzes the family of distributions represented by SPNs in connection to related models. Section 3.3 describes parameter learning algorithm for SPNs, and introduces our derivation of Expectation-Maximization that enables learning of structured leaf distributions. Section 3.4 describes structure learning approaches for SPNs. Finally, Section 3.5 presents an overview of applications of SPNs.

3.1. Model Description

Let us start by fixing some definitions and notation for the rest of the chapter. Uppercase letters A, B, C, D denote random variables and corresponding lowercase letters a, b, c, d denote values in their domains $\Delta(A), \Delta(B), \Delta(C), \Delta(D)$. X, Y denote sets of random variables, with corresponding domains $y, x \in \Delta(X), \Delta(Y)$. As described in Eq. 2.1.1, the domain of a set of variables is the Cartesian product of the domains of individual variables.

Definition 3.1.1. A *tractable probability distribution* $\varphi(X)$ is a distribution over X in which evaluating any probabilistic query requires a polynomial number of operations.

Remark 6. Examples of tractable distributions are Gaussian distributions and tree graphical models (Section 2.1 and 2.3.2).

Definition 3.1.2. *Scope* of a graph. Consider a rooted directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which each node $q \in \mathcal{V}$ is associated to a (possibly empty) set of variables X_q. The union of all the variables appearing in the graph $(\bigcup_{q \in \mathcal{V}} X_q)$ is called *scope* of the graph.

Definition 3.1.3. Sum-Product Network. Let us consider the following quantities:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a rooted Directed Acyclic Graph (DAG).
- X denotes a set of continuous of discrete variables; W denotes a set of non-negative real weights.
- $\Phi = \{\varphi_q(X_q|\theta_q)\}_{q \in \mathcal{V}}$ where $X_q \subseteq X$ denotes a set of tractable probability distributions with parameters $\theta = \left\{\bigcup_{q \in \mathcal{V}} \theta_q\right\}$.

A *Sum-Product Network* $S(X|\mathcal{G}, \Phi, W, \theta)$ (simply $S(X)$ for compactness) is a rooted DAG where:

- An internal node $q \in \mathcal{V}$ can be either
  - a *Product Node* $\otimes$
  - a *Sum Node* $\oplus$ s.t. each outgoing edge $(q, i) \in \mathcal{E}$ is associated to a real weight $w_{q,i} \in W$
- A leaf node $q \in \mathcal{V}$ is associated to a variable set $X_q \subseteq X$ and to a tractable probability distribution $\varphi_q(X_q|\theta_q)$ with parameters $\theta_q \subseteq \theta$

In addition, the following conditions are satisfied:

1. *Decomposability.* The subgraphs rooted in any two children of a Product Node have disjoint scope.
2. *Completeness.* The subgraphs rooted in any two children of a Sum Node have identical scope.

Remark 7. Decomposability can be relaxed when considering a special case of SPNs with indicator variables as leaves. However, Decomposable SPNs as in the above definition are as general as non decomposable ones (Peharz [2015]).

Remark 8. *Graphical Representation.* In the graphical representation of a SPN, Sum Nodes are represented with the symbol $\oplus$, Product Nodes with the symbol $\otimes$, and leaf nodes with the mathematical symbol of the associated distribution: for instance, a Gaussian leaf node q will be represented as $\mathcal{N}(X_q|\mu_q, \Sigma_q)$ (section 2.1), and a tree

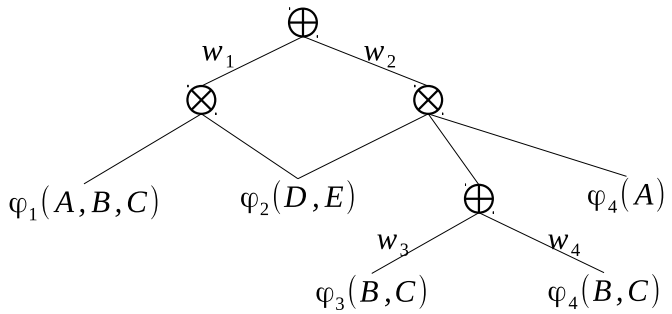


Figure 3.1.1. - A Sum-Product Network $S(X|\mathcal{G},\Phi,W,\theta)$, where: $X = \{A,B,C,D,E\}$,$\Phi = \{\varphi_1,\varphi_2,\varphi_3,\varphi_4,\varphi_5\}$,$W = \{w_1,w_2,w_3,w_4\}$. Note that the conditions of definition 3.1.3 are satisfied.

graphical model leaf can be represented as $T_q(X_q)$ (section 2.3.2). Children nodes are drawn *below the parents,* and directional arrow heads pointing downwards are not drawn. An example SPN is provided in figure 3.1.1.

The relevance of SPNs is strictly related to their evaluation procedure, described next.

Definition 3.1.4. Evaluation of a Sum-Product Network. Let $S(X)$ be a SPN, and let $Y \subseteq X$ be a set of observed variables with assignment $y \in \Delta(Y)$. Let $Y_c = X \setminus Y$ be the complementary set to Y. The evaluation of S for assignment $Y = y$, written as $S(y)$, proceeds evaluating each node $q \in \mathcal{V}$ in inverse topological order (from the leaves to the root of the DAG) with the following rules:

1. if q is a leaf node, let $Y_q = Y \cap X_q$ be the subset of observed variables appearing in X_q. Evaluate the node by marginalizing over the non observed variables:

$$S_q(y_q) = \int_{y_c \in \Delta(Y_c \cap X_q)} \varphi_q(Y_c \cap X_q = y_c, Y_q = y_q) dy_c. \tag{3.1.1}$$

   The integration is substituted by summation for discrete variables.

2. if q is a Product Node, compute the product of children values:

$$S_q(x) = \prod_{(q,i)\in\mathcal{E}} S_i(x_i). \tag{3.1.2}$$

3. if q is a Sum Node, compute the weighted sum of children values:

$$S_q(x) = \sum_{(q,i)\in\mathcal{E}} w_{q,i} S_i(x). \tag{3.1.3}$$

The value of $S(x)$ is the value of the root of S.

We will show in the following that evaluating a SPN corresponds to computing marginals in a probability distribution.

Proposition 9. *Evaluating $S(X)$ takes $O(|\mathcal{E}| + N_l K)$ time and memory, where N_l denotes the number of leaves and K denotes the worst case cost of evaluating leaves (Definition 3.1.1).*

Proof. Evaluating internal nodes has cost $O(|\mathcal{E}|)$ since each node is evaluated exactly once, and the operations for each node are linear in the number of edges outgoing from the node. Each leaf is evaluated exactly once, hence evaluation of the leaves has cost $O(N_l K)$. Hence the result. □

From the previous proposition it follows that evaluating a SPN is always tractable, since the leaf models are tractable by Definition 3.1.3.

Notice that the evaluation of $S(X)$ involves iteratively evaluating the SPNs rooted at each internal node q. Hence, it is possible to compute the derivative of the value of the root ($S(x)$) with respect to the values of internal nodes, as follows. This quantity is crucial in parameter learning approaches.

Proposition 3.1.1. *Let $S(X)$ be a SPN, and let $S_q(X_q)$ denote the SPN rooted at node $q \in \mathcal{V}$ of S. The derivative of the value $S(x)$ w.r.t the value $S_q(x)$, denoted as $\frac{\partial S(x)}{\partial S_q}$, can be computed with the following recursive equations, noting that $\frac{\partial S(x)}{\partial S_{root}} = 1$:*

$$\frac{\partial S\left(x\right)}{\partial S_q} = \sum_{(k,q)\in\mathcal{E}} \frac{\partial S\left(x\right)}{\partial S_k} \frac{\partial S_k\left(x\right)}{\partial S_q}, \tag{3.1.4}$$

$$\frac{\partial S_k\left(x\right)}{\partial S_q} = \begin{cases} \prod_{(i,q)\in\mathcal{E}:\, i\neq k} S_i, & q \text{ product node, } (k,q)\in\mathcal{E} \\ w_{k,q}. & q \text{ sum node, } (k,q)\in\mathcal{E} \end{cases} \tag{3.1.5}$$

Proof. First, the quantity $S_q(x)$ appears in the computation of $S(x)$ only through its influence to terms $S_k\left(x\right)$ for each parent $k \in pa(q)$, hence applying the rule of total derivatives Eq. 3.1.4 follows. Then, it is straightforward to see by direct derivation of Eqs. 3.1.2 and 3.1.3 that Eq. 3.1.5 holds. □(the sum can be replaced with integral for continuous variables)

Remark 10. Note that this operation requires first evaluating the SPN with a recursive "upward pass" from the leaves to the root, in order to compute the quantity $S_q(x)$ for each node $q \in \mathcal{V}$, then to compute the derivatives with a recursive "downward pass" from the root to the leaves. Hence, the values and derivatives of *all* nodes in the networks are computed in a *single* up-and-down pass, therefore by Proposition 9 the cost of computing derivatives is $O(|\mathcal{E}| + N_l K)$.

3.1.1. Interpretation as Mixture Model

This section contains the interpretation of SPNs as very large mixture model. This interpretation allows to specify the family of distributions represented by SPNs, and to deduce properties useful to compute quantities of interest in the following.

While it was implied since the very first paper on SPNs that any SPN could be transformed into an equivalent SPN corresponding to a very large mixture model (see Poon and Domingos [2011]), this interpretation was first exploited in Dennis and Ventura [2015] and then formalized independently by Zhao et al. [2016b] and in Desana and Schnörr [2016]. We include here the essential steps of this discussion.

We start by introducing the concept of subnetwork its properties.

Definition 3.1.5. Let $S(X)$ be a SPN. A *subnetwork* σ is a SPN defined on a subtree of the DAG $\mathcal{G}$ underlying S (cf. Def. 3.1.3). It is recursively constructed by first including the root of S in σ, then processing each node q included in σ as follows:

1. If q is a Sum Node, include in σ one child $i \in ch\,(q)$ with relative weight w_i^q. Process the included child.

2. If q is a Product Node, include in σ all the children $ch\,(q)$. Process the included children.

We denote by $\Sigma(S)$ the set of all subtrees of S.

Example: Fig. 3.1.3, left. The term "subnetwork" was first introduced in Gens and Domingos [2012] that informally mentions this concept.

Proposition 3.1.2. *Any subnetwork $\sigma \in \Sigma\,(S)$ is a tree SPN.*

Proof. Only Product Nodes in $\sigma \in \Sigma\,(S)$ can have multiple children, since Sum Nodes have a single child in σ by Definition 3.1.5, and children of Product Nodes have disjoint graphs by Definition 3.1.3., case 3. Therefore σ contains no cycles. A rooted graph without cycles is a tree. □

Proposition 3.1.3. *The number $|\Sigma\,(S)\,|$ of subtrees of S grows as $O(\exp(|\mathcal{E}|))$.*

Proof. In Zhao et al. [2016b]. It can be shown by inspection considering a SPN constructed as in Fig. 3.1.2: if there are M Sum Nodes and each Sum Node has C children, then the number of edges is $2NC$ and the number of subnetworks is C^N (combinations of C choices at each node). □

SPNs as Mixtures of Subnetworks

In this section, we show that SPNs can be interpreted as mixtures of factorizations of the leaf distributions. Table 3.1.1 lists the notation and probabilistic (sub-)models relevant in this context.

As a first step, we show that inference in a subtree σ due to Definition 3.1.5 is equivalent to inference in the distribution $P_\sigma\,(X)$ (Eq. 3.1.8), multiplied for a constant factor determined by the product of all edge weights nodes in the subnetwork.

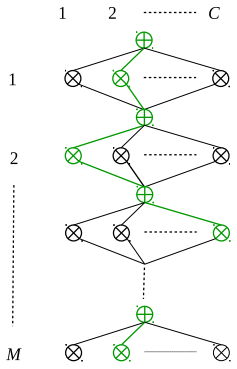


Figure 3.1.2. - Graphical representation of the proof of Proposition 3.1.3. A subnetwork of this SPN is highlighted in green.

$$P(X) = \sum_{\sigma \in \Sigma(S)} \lambda_\sigma P_\sigma(X), \tag{3.1.6}$$

$$\lambda_\sigma = \prod_{(q,j) \in \mathcal{E}(\sigma)} w_{q,j}, \tag{3.1.7}$$

$$P_\sigma(X) = \prod_{l \in \mathcal{L}(\sigma)} \varphi_l(X_l | \theta_l). \tag{3.1.8}$$

Table 3.1.1. - Probabilistic model of a SPN $S(X)$. The symbol $\mathcal{L}(\sigma)$ denotes the set of leaves in a subnetwork $\sigma \in \Sigma(S)$. Evaluation of $\mathbb{S}$ (Definition 3.1.4) is equivalent to inference using the distribution (3.1.6).

Proposition 3.1.4. *Let $S(X)$ be a given SPN, and let $\sigma \in \Sigma(S)$ be a subnetwork (Def. 3.1.5). Evaluating σ with assignment $Y = y, Y \subseteq X$ (Definition 3.1.4) is equivalent to marginal inference using the distribution*

$$\lambda_\sigma P_\sigma(Y = y), \tag{3.1.9}$$

where terms λ_σ and P_σ are as in Eq. 3.1.7 and 3.1.8.

Proof. In Zhao et al. [2016b]. Immediate, considering that σ is a tree in which Sum Nodes have only one child and hence contribute only with a multiplicative edge weight, and that leaf nodes evaluate due to Definition 3.1.4.

□

Remark 11. Note that the mixture coefficients are products of all the sum weights in a subnetwork σ, and mixture components are factorizations obtained as products of leaves in σ (see Fig. 3.1.3, left).

The second step consists in noting that S can be written equivalently as the mixture of all its subtrees.

Proposition 3.1.5. *Evaluating a SPN $S(X)$ is equivalent to evaluating a SPN $S'(X) = \sum_{\sigma \in \Sigma(S)} \sigma(X)$.*

Proof. In Zhao et al. [2016b].

□

Intuitively, this result follows from the fact that one can write a sequence of nested sums as a single "top level" sum, as in $\sum_a f(a) \sum_b f(b) \sum_c f(c) = \sum_{a,b,c} f(a)f(b)f(c)$. Correspondingly, Sum Nodes that are effectively eliminated from subtrees (having a single child) are substituted by a single root Sum Node with many children.
We are now prepared to state the main result of this section.

Proposition 3.1.6. *Let $Y \subseteq X$ denote evidence variables with assignment $y \in \Delta(Y)$, and denote by $x_{\setminus y} \in \Delta(X \setminus Y)$ assignments to the remaining variables. Evaluating a SPN $S(X)$ with assignment $Y = y$ (Definition 3.1.4) is equivalent to performing marginal inference with respect to the distribution* (3.1.6) *as follows:*

$$P(Y = y) = \sum_{x_{\setminus y} \in \Delta(X \setminus Y)} P\big((X \setminus Y) = x_{\setminus y}, Y = y\big). \tag{3.1.10}$$

Proof. The result follows by concatenating Propositions 3.1.4 and 3.1.5. □

Remark 12. The propositions above entail the crucial result that the probabilistic model of a SPN is a mixture model where the mixture size grows *exponentially* with the SPN size (Proposition 3.1.3), but in which the inference cost grows only *linearly* (Proposition 9). Hence, *very large* mixtures models can then be modelled *tractably*. This property is obtained by combining shared leaf models in different mixture components, and by computing inference in shared parts only once (cf. the example in Fig. 1.2.2).

Remark 13. Proposition 9 and 3.1.6 also entail that the partition function can be always computed with a linear cost in the number of edges in the SPN.

Evaluating Subsets of the Encoded Mixture The following proposition constitutes one of the main contributions of this work, and it allows to derive several results in the following.

Proposition 3.1.7. *Consider a SPN $S(X)$, a Sum Node $q \in S$ and a node $i \in ch(q)$. The following relation holds:*

$$\sum_{\sigma \in \Sigma(S):(q,i) \in \mathcal{E}} \lambda_\sigma P_\sigma(X) = w_i^q \frac{\partial S(X)}{\partial S_q} S_i(X), \tag{3.1.11}$$

where $\sum_{\sigma \in \Sigma(S):(q,i) \in \mathcal{E}}$ denotes the sum over all the subnetworks σ that include the edge (q, i).

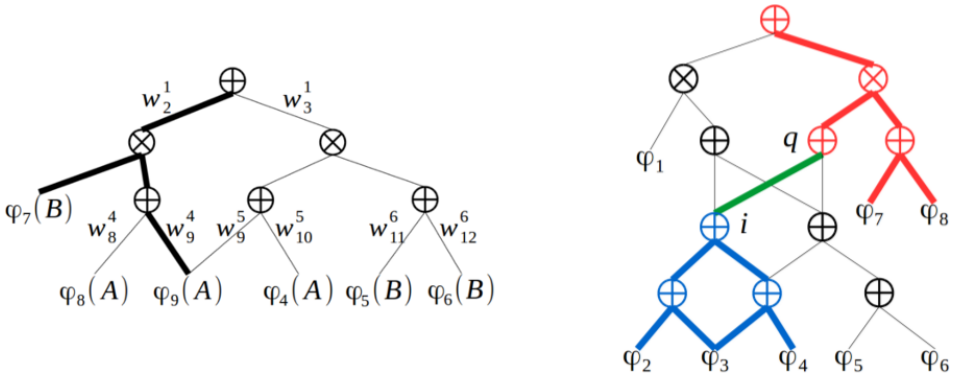


Figure 3.1.3. - (Left): A SPN $S(A, B)$ in which a subnetwork σ is highlighted. This subnetwork corresponds to a mixture coefficient $\lambda_\sigma = w_2^1 w_9^4$ and component $P_\sigma(A, B) = \varphi_7(B)\,\varphi_9(A)$. (Right): Visualization of Lemma 3.1.7. The colored part is the set of edges traversed by subnetworks crossing (q, i). The blue part represents S_i and the red part covers terms appearing in $\frac{\partial S(X)}{\partial S_q}$.

Proof: in Appendix A.2.1.

This Lemma relates the evaluation of subset of the mixture model corresponding to subnetworks σ that include (q, i) and the value and derivative of the SPN nodes q and i. Note that evaluating the left hand sum has a cost $O(|\Sigma(S)|)$ (intractable) but the right hand term has a cost $O(\mathcal{E})$ (tractable). Since in the derivation of EM one needs to evaluate such subsets of solution, this Lemma allows to compute the quantity of interest in a single SPN evaluation.

Furthermore, a single evaluation and derivation of S, which enables to compute all values $S_i(X)$ and $\frac{\partial S(X)}{\partial S_q}$ jointly for all nodes, is required to compute this sum over each subset of subnetworks.

Note also that $\sum_{\sigma\in\Sigma(S):(q,i)\in\mathcal{E}} \lambda_\sigma P_\sigma(X)$ corresponds to the evaluation of a non-normalized SPN which is a subset of S - e.g. the colored part in Fig. 3.1.3, right.

Computing Unary Marginals Lemma 3.1.7 allows to derive the following proposition:

Proposition 3.1.8. *The marginal distribution w.r.t. a variable $A \in X$, namely $P(A) = \sum_{x_{\setminus a}\in\Delta(X\setminus A)} P(A, x_{\setminus a})$, can be computed as follows:*

$$P(A) = \sum_{k\subseteq\mathcal{L}(S):A\in k} \frac{\partial S(X)}{\partial S_k} \left(\sum_{x_k\in\Delta(X_k\setminus A)} \varphi_k(A, x_k) \right), \tag{3.1.12}$$

where $\sum_{k\subseteq\mathcal{L}(S):A\in k}$ denotes the sum over all leaf nodes $k \in \mathcal{V}$ which contain A.

Proof. In Appendix A.1. This result is important since it allows to compute the marginals of *all* the variables jointly by evaluating and differentiating the SPN *only once*, since the only quantity required from the internal SPN nodes is $\frac{\partial S(X)}{\partial S_k}$ which is not dependent on V. A very similar property holds for message passing in Junction Trees,

where all marginals can be computed with two iterations of message passing over the tree (Pearl [2000]).

3.1.2. Max-SPNs

It can be proved that all the properties derived above for *Sum*-Product Networks translate immediately to *Max*-Product Networks: SPNs in which the sums in Eq. 3.1.3 and the integration in Eq. 3.1.1 are substituted by the max operator.

Proposition 3.1.9. *Let $Y \subseteq X$ denote evidence variables with assignment $y \in \Delta(Y)$, and denote by $x_{\setminus y} \in \Delta(X \setminus Y)$ assignments to the remaining variables. Evaluating a Max-SPN $S(X)$ with assignment $Y = y$ (Definition 3.1.4) is equivalent to performing MAP inference with respect to the distribution* (3.1.6) *as follows:*

$$\max_{x_{\setminus y} \in \Delta(X \setminus Y)} P\big((X \setminus Y) = x_{\setminus y}, Y = y\big). \tag{3.1.13}$$

This follows since the only properties that were used in deriving the proofs were that Sum-Product operations form a *semiring,* and hence any operations forming a semiring (including the Max-Product one) can be used. This aspect has also a correspondence in literature on message passing for graphical models, where sums can be substituted with max to compute Maximum-A-Posteriori (MAP) inference (Pearl [2000]).

3.1.3. Normalized SPNs

Definition 3.1.6. Normalized SPN. A SPN is normalized if for each Sum Node w^q the nonnegative weights w^q sum to 1, and each leaf distribution is normalized.

Proposition 3.1.10. *A normalized SPN encodes a normalized distribution.*

Proof. The proof proceeds by induction, noting that Sum Nodes compute a convex combination of normalized distributions, which is a normalized distribution, and that Product Nodes compute the product of normalized distribution over disjoint sets of variables, which is a normalized distribution. □

The fact that the partition function in a SPN is always tractable (due to Proposition 3.1.6) guarantees that any SPN can be normalized without loss of generality, adding the normalization term as a multiplicative constant to the root node. This is formalized in the following result, taken from Peharz [2015].

Proposition 3.1.11. *For each SPN $\hat{S}\left(X, G, \hat{W}, \theta\right)$, there exists a locally normalized SPN $S\left(X, G, W, \theta\right)$, such that $S(x) = \frac{\hat{S}(x)}{Z}$ where Z is the partition function of $\hat{S}$.*

Proof. In Peharz [2015]. □

3.1.4. SPNs with Indicator Variable Leaves

SPNs with indicator variable leaves are an important special case both for an historical reasons, since the first proposed SPN model fell in this case, and a practical one, since several applications of SPNs use this subclass of models.

Proposition 3.1.12. *A SPN with indicator variable leaves encodes a network polynomial.*

Proof. Substituting the indicator variables as leaves in Eq. 3.1.6 we obtain:

$$P(X) = \sum_{\sigma \in \Sigma(S)} \underbrace{\prod_{(q,j) \in \mathcal{E}(\sigma)} w_{q,j}} \prod_{l \in \mathcal{L}(\sigma)} [X_l]_{x_l},$$

which is a Network Polynomial (Definition 2.2.3). □

Since the sum $\sum_{\sigma \in \Sigma(S)}$ is intractable if explicitly represented (see Proposition 3.1.3), but the evaluation of the associated SPN is tractable (Proposition 9), it follows that SPNs are a compact way to express Network Polynomials.

Example 14. For an example of how a Network Polynomial can be represented efficiently as a SPN, see Section 1.1.1.

3.2. SPNs and Related Architectures

We saw in the previous section that any SPN represents a very large mixture model in which inference is tractable. It needs to be discussed, then, how SPN compare and connect to related families of architectures. This analysis goes through three steps. First, we discuss the connection between SPNs and And/Or search graphs. Then, we further discuss SPNs as a potentially very large, hierarchical mixture model, whose components are combinations of the distributions at the leaves. Finally, we put SPNs in relation with graphical models, discussing advantages and disadvantages of the two representations.

3.2.1. SPNs and OR trees

The connection with OR trees is based on the concept of Observed Product Node, discussed next.

Definition 3.2.1. Observed Product Node. A Product Node $s \in \mathcal{V}$ is an *observed Product Node* if it has at least one child which is an indicator variable $[X_s]_i$ and one child which is not an indicator variable. X_s is called *observed variable*.

An Observed Product Node s has the effect of "activating" the sub-SPN rooted in s if the observed variable has state i (it is multiplied by the associated indicator that has value 1), and deactivating it otherwise (the associated indicator has value 0).

OR Trees OR trees represents the search space explored during probabilistic inference (see e.g. Luger [2004]). An OR tree is a directed tree graph $\mathcal{T} = \overline{\mathcal{V}}, \overline{\mathcal{E}}$ where each node $v \in \overline{\mathcal{V}}$ is associated to a variable $X_v \in X$, and the i-th child of v corresponds to the i-th state of X_v (Fig. 3.2.1, Left). Each path from the root to node v represents conditioning on the set of states $y = \{X_a = x_a, X_b = x_b, \dots\}$ represented by the set of variables $Y = \{X_a, X_b, \dots\}$ traversed by the path, and models at leaves corresponds to probabilities $P(X \backslash Y | Y = y)$.

An OR tree is evaluated for a given evidence $Y = y$ by performing the sum over values of the children for which the state is compatible with evidence variables. Any discrete distribution can be represented using a OR Tree by simply conditioning on each state of each variable in turn, but this requires in the worst case $O(D^N)$ parameters where D is the maximum domain size and N is the number of variables in the distribution. This is clearly not an efficient representation. However, OR trees enable to represent contextual independence compactly by potentially branching on different values of different variables at each node.

The interesting property of OR trees is that each path from the root to the leaves corresponds to a context, OR trees are particularly apt at representing contextual dependences, and as such some distributions can be expressed efficiently by OR trees but not by Graphical Models (see Section 1.1.2).

And/Or Trees

AND/OR trees are an extension of OR trees that include AND nodes, that multiply the values of their children. Clearly, these nodes correspond to Product Nodes in SPNs and thus SPNs can represent an AND/OR trees.

SPNs and OR trees It is immediate to see that both OR and AND nodes can be represented as SPNs by using observed variables, as shown e.g. in Fig. 3.2.1, right.

However, SPNs are more general than both OR and AND/OR trees in that their structure is not limited to a tree but can be a more general DAG. In this way, some distributions can be expressed exponentially more compactly than by using a tree structure (see Proposition 3.1.3).

Furthermore, SPNs generalize the concept of And/Or graph by allowing arbitrary leaf distributions to be placed as leaves rather than only indicator variables. An example is shown in figure 3.2.1, where it can be also seen that different conditional independences between variables (represented by the tree leaves) are present depending on the *context,* that is on the state of the indicator variables traversed in the path from the root to each leaf.

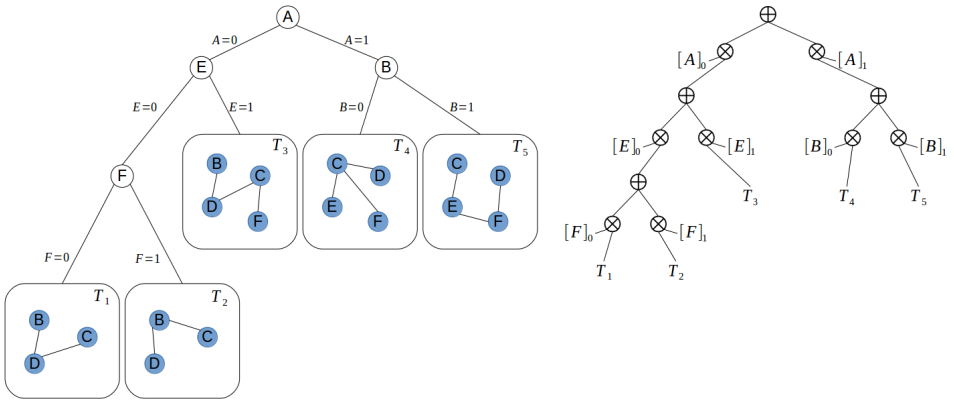


Figure 3.2.1. - Left: An OR tree with tree graphical models as leaves (also called Cutset Network). Right: representation of the OR tree as a SPN using observed variables (the unitary sum weights are not shown).

3.2.2. SPNs, And/Or graphs and Arithmetic Circuits

There is significant amount of overlap between Sum-Product Networks and some related architectures which are inspired by And/Or trees. This led to the repeated discovery of the same algorithms, and to a situation of confusion between closely related literature that bear different names. Let us briefly reviews these relationships:

Arithmetic Circuits Sum Product Networks with indicator leaves are closely connected Arithmetic Circuits, introduced previously to SPNs in Darwiche [2002]. This correspondence has been noted already in Poon and Domingos [2011], and descends immediately from the definitions of the two architectures.

In fact, the only distinction between SPNs with indicator variable leaves and Arithmetic Circuits is the way in which weights are represented: in SPNs they are associated to Sum Node edges, while in Arithmetic Circuits they are associated to leaf nodes and removed from the edges. However, it is possible to pass from one representation to another without loss of generality with a straightforward algorithm.

In practice, another distinction between SPNs and Arithmetic Circuits is that the former have been proposed and used as a standalone architecture, while the latter have been used as a tool to perform inference in graphical models.

Also note that SPNs with tractable leaves as in Definition 3.1.3 (which is not the one originally introduced in Poon and Domingos [2011]) are more general than SPNs with indicator variable leaves, which can be obtained as a special case.

AND/OR graphs Sum Product Networks with indicator variable leaves are also AND/OR graphs Dechter and Mateescu [2007], which generalize AND/OR trees to a graph structure (as opposed as a tree), and employs them to perform inference in graphical models. This result descends from the equivalence between And/Or graphs and Arithmetic Cir-

cuits, which is discussed in Dechter and Mateescu [2007], and the equivalence between Arithmetic Circuits and SPNs.

Cutset Networks Cutset Networks Rahman et al. [2014] are Or trees which have at leaves Tree Graphical models, and can therefore be identified as SPNs as discussed above (Fig. 3.2.1). Hence, Cutset Networks are SPNs where the DAG has no directed cycles.

3.2.3. Graphical Models and SPNs

Junction Trees and SPNs

Exact inference in Graphical Models can be computed efficiently with the Junction Tree algorithm: a Junction Tree is obtained from the given graphical model, then inference in the Junction Tree is computed as described in section 2.3.3. The procedure to obtain a Junction Tree from a given graphical model is described e.g. in Murphy [2012]. It can be shown that inference in the optimal Junction Tree has a cost lower bounded by $\exp(T)$ where T is the treewidth of the graph. Hence, the Junction Tree algorithm can be applied tractably only provided that the treewidth of the original graph is *small*. A typical example of tractable graphs are tree graphs where the treewidth is 2.

The message passing procedure associated to a Junction Tree can be directly interpreted as a SPN, since it can be shown that Eq. 2.3.7 encodes a valid SPN according to Definition 3.1.3. An example of the interpretation of message passing as SPN was shown in Section 1.1.1; we refer to Darwiche [2003] for further details. We also visualize the transformation from GM to Junction Tree to SPN in Fig. 3.2.2.

Notice that the inference procedure has the same cost for both the Junction Tree and the corresponding SPN.

Advantages and Disadvantages of SPNs over Graphical Models

The trade off between SPNs and graphical models was discussed in Section 1.1.1. From the relation with Junction Trees it is once more evident that the SPN representation loses the high level representation of the distribution in terms of factorization, which is an appealing property of graphical models (see Fig. 3.2.2). In addition, SPNs in this context are not able to express the distribution more efficiently than Junction Trees, since the inference cost is the same.

The use of SPNs is motivated by modeling distributions that exhibit contextual independence, and that therefore cannot be modeled efficiently as a graphical model in the first place. When a distribution includes only contextual independences, it is more convenient to use the graphical model representation. The need to represent distributions with both contextual and conditional independence mainly motivates the class of distributions described in Section 4.1.

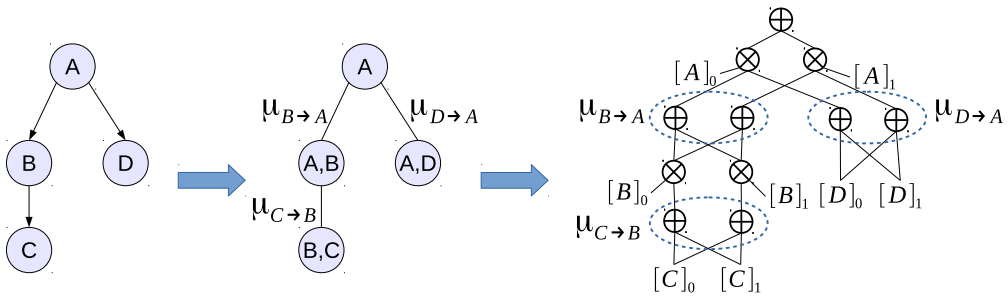


Figure 3.2.2. - Conversion of a tractable Graphical Model into a Sum-Product Network via the Junction Tree algorithm. The graphical model (left), corresponding Junction Tree (middle) and SPN (right). This SPN represents the Network Polynomial in Eq. 1.1.1.

3.3. Parameter Learning

A SPN $S(X|\mathcal{G}, \Phi, W, \theta)$ (Definition 3.1.3) is governed by four set of parameters: graph structure $\mathcal{G}$; edge weights W; choice of leaf distributions Φ and corresponding leaf distribution parameters θ. Learning in SPNs involves finding Maximum Likelihood solutions for these parameters, given a dataset of observations $\{x_i\}_{i=1}^N$ (Section 2.1.2). In this section we consider the structure $\mathcal{G}$ and the choice of leaf distributions Φ fixed, and we concentrate on learning the *parameters* W, θ. This problem is known in literature as *parameter learning*.

3.3.1. Gradient Ascent Based Approaches

The earliest proposed training methods for SPNs were gradient ascent based approaches (Poon and Domingos [2011], Gens and Domingos [2012]). We report here a compact description of these methods for historical reasons and to prepare for Section 3.3.2.

Gradient Ascent Gradient ascent is an iterative procedure that finds the local maximum of a given function $f(\theta)$ by iteratively performing "small" steps in the parameter space towards the direction of maximum ascent, locally given by the gradient $\frac{\partial f(\theta)}{\partial \theta}$ (see e.g. Murphy [2012]). A step of the gradient ascent procedure takes the form:

$$\theta_{new} = \theta + \eta \frac{\partial f(\theta)}{\partial \theta},$$

where η is a "small" positive coefficient which is a hyperparameter of the algorithm. In our case, we are interested to find a maximum of the Log Likelihood of $S(X)$ ($LL(S|\theta)$, Eq. 2.1.15), hence an update assumes the form:

$$\theta_{new} = \theta + \eta \frac{\partial}{\partial \theta} \left(LL(S|\theta) \right),$$

In the following, $S(x_n)$ denotes the evaluation of $S(X)$ for sample x_n (see Definition

3.1.4). Computing the gradient of the Log Likelihood w.r.t. a weight $w_{q,i}$ (corresponding to the edge (q, i) emanating from a Sum Node $q \in \mathcal{V}$) we obtain:

$$\frac{\partial}{\partial w_{q,i}}\left(\sum_{n=1}^{N} \ln S(x_n)\right) = \sum_{n=1}^{N} \frac{\partial}{\partial w_{q,i}} \ln S(x_n) = \sum_{i=1}^{N} \frac{1}{S(x_n)} \left(\frac{\partial S(x_n)}{\partial w_{q,i}}\right).$$

Hence the derivative $\frac{\partial S(x_n)}{\partial w_{q,i}}$ is required. Similarly, for leaf distribution parameters$\theta_k \in \theta$ (appearing at leaf nodes) the derivative $\frac{\partial S(x_n)}{\partial \theta_q}$ is required.

Computing $\frac{\partial S(x_n)}{\partial w_{q,i}}$ and $\frac{\partial S(x_n)}{\partial \theta_q}$ These derivatives can be computed efficiently. First, assuming that parameters w_{qk} and θ_q appear only in node q, then the derivatives are computed as follows (see Proposition 3.1.1):

$$\frac{\partial S\left(x\right)}{\partial w_{qk}} = \frac{\partial S\left(x\right)}{\partial S_q}\frac{\partial S_q\left(x\right)}{\partial w_{qk}} = \frac{\partial S\left(x\right)}{\partial S_q} S_k(X),$$
$$\frac{\partial S\left(x\right)}{\partial \theta_q} = \frac{\partial S\left(x\right)}{\partial S_q}\frac{\partial S_q\left(x\right)}{\partial \theta_q} = \frac{\partial S\left(x\right)}{\partial S_q}\frac{\partial \varphi_q(x)}{\partial \theta_q}.$$

Furthermore, if the term w_{qk} is associated to multiple edges $(q, k) \in \mathcal{E}$ due to weight sharing constraints (or the term θ_q is associated to multiple nodes $q \in \mathcal{V}$), we can collect all edges that share the same parameter w_{qk} in a set $\overline{\mathcal{E}}$ (similarly, all leaf nodes sharing θ_q can be collected in a set $\overline{\mathcal{V}}$), and applying the rule of total derivatives we obtain the gradient as the sum of the gradients of element in R:

$$\frac{\partial S\left(x\right)}{\partial w_{qk}} = \sum_{(i,k)\in\overline{\mathcal{E}}} \frac{\partial S\left(x\right)}{\partial S_i} S_k(X),$$
$$\frac{\partial S\left(x\right)}{\partial \theta_q} = \sum_{q\in\overline{\mathcal{V}}} \frac{\partial S\left(x\right)}{\partial S_i}\frac{\partial \varphi_i(x)}{\partial \theta_q}.$$

Hence, evaluating this quantity requires evaluating and derivating the SPN only once.

Problems of Gradient Ascent and Workarounds Gradient ascent approaches suffer from the well known problem of vanishing gradients in the "deep" regions of the SPN (i.e., the nodes further away from the root). This happens since the magnitude of the updates decreases exponentially at each step, up to a point where updates at deep nodes are negligible. To see this, it is enough to consider the computation of the derivative (Proposition 3.1.1) noting that the gradient gets multiplied by terms $P_i \leq 1$ at each Product Node (Eq. 3.1.5), and as such it gets exponentially smaller with increasing node depth.

To tackle the problem of gradient diffusion and allow to train deep SPNs effectively, Gens and Domingos [2012] suggested to compute a rough approximation of the gradient, obtained by counting how many times a certain edge was traversed during the MAP evaluation of the SPN (Section 3.1.2), and dividing by the number of samples in the dataset.

This procedure allowed to train SPNs with good empirical performances in Gens and Domingos [2012], however it involves an arbitrary approximation of the original objective and no analysis of the approximation error is provided. We do not describe this procedure in detail since it is only empirically justified and it has been recently replaced by Expectation Maximization for SPNs, which allows to generate updates that do not vanish at deeper nodes.

3.3.2. Expectation Maximization

Expectation Maximization is an elegant and widely used method for training mixture models (see Section 2.4.3). Given the interpretation of a SPN as a very large mixture model due to Eq. 3.1.6, it is tempting to try to apply EM to SPNs. However, doing so involves finding ways of exploiting the structure of the SPN in order not to *explicitly* compute EM on the mixture model, which can be intractably large.

Several independent attempts were made to derive Expectation Maximization on the intractably large mixture model encoded by a SPN by efficiently exploiting the shared structure of the mixture. The very first attempt contained (Poon and Domingos [2011]) contained an error in the derivation, pointed out by Peharz et al. [2016]; then, subsequent methods obtained the correct EM update for SPN weights (Zhao et al. [2016b],Peharz et al. [2016]) and for univariate leaves in the exponential family (Peharz et al. [2016]).

We describe here an independent derivation of EM for SPN, which:

1. Employs a simpler derivation than the alternate versions, directly translating the standard EM algorithm to SPNs.

2. Is more general that existing methods, since it can be applied to learn SPNs with arbitrary leaf distributions and extends to the case of shared parameters.

3. Provides guarantees of convergence under mild condition for a very large class of leaf distributions. In particular, the M-step results in a simple weighted Log Likelihood maximization for each leaf distribution, which can be solved also sub-optimally and can be performed straightforwardly for a wide class of distributions.

Derivation We want to apply EM to the mixture encoded by a SPN, which is in principle intractably large. First, let us use the relation between SPN and encoded mixture model in Table 3.1.1, and identify the probabilities required in the classical EM algorithm. First, let us denote as z_σ the state of the sum node variables associated to subnetwork σ. Then, we identify the following probabilities:

$$P(Z = z_\sigma, x_n|\theta) = \lambda_\sigma(W) P_\sigma(x_n|\theta),$$
$$P(x_n|W_{old}, \theta_{old}) = S(x_n|W_{old}, \theta_{old}).$$

and therefore:

$$P(Z = z_\sigma|X = x_n, W_{old}, \theta_{old}) = \frac{P(Z = z_\sigma, X = x_n|W_{old}, \theta_{old})}{P(X = x_n|W_{old}, \theta_{old})} = \frac{\lambda_\sigma(W) P_\sigma(X = x_n|\theta_{old})}{S(X = x_n|W_{old}, \theta_{old})}.$$

Applying these substitutions and dropping the dependency on W_{old}, θ_{old} for compactness, the EM objective function $Q(W, \theta|W_{old}, \theta_{old})$ defined in section 2.4.3 becomes:

$$Q(W, \theta) = \sum_{n=1}^{N} \sum_{\sigma \in \Sigma(S)} \frac{\lambda_\sigma P_\sigma(x_n)}{S(x_n)} \ln \lambda_\sigma(W) P_\sigma(x_n|\theta). \tag{3.3.1}$$

In the following sections we will efficiently maximize $Q(W, \theta)$ for W and θ, trying in particular to get rid of the intractable sum $\sum_{\sigma \in \Sigma(S)}$.

3.3.3. Weights Update

We can simplify $Q(W, \theta)$ through the use of Proposition 3.1.7 (derivation in Appendix A.2.2), ending up with the maximization of the following objective function:

$$W^* = \arg\max_W \mathcal{Q}_W(W), \tag{3.3.2}$$
$$\mathcal{Q}_W(W) = \sum_{q \in \mathcal{N}(S)} \sum_{i \in ch(q)} \beta_i^q \ln w_i^q, \tag{3.3.3}$$
$$\beta_i^q = w_{i,old}^q \sum_{n=1}^{N} S(x_n)^{-1} \frac{\partial S(x_n)}{\partial S_q} S_i(x_n). \tag{3.3.4}$$

The evaluation of terms β_i^q, which depend only on W_{old}, θ_{old} and are therefore constants in the optimization, is the *E step* of the EM algorithm. We now maximize $\mathcal{Q}_W(W)$ subject to $\sum_i w_i^q = 1 \forall q \in \mathcal{N}(S)$ (*M step*).

Non shared weights. If weights at each node q are disjoint, then we can move the max inside the sum, obtaining separated maximizations each in the form \sum_{c=1}^{C}

$$\arg\max_{w^q} \sum_{i \in ch(q)} \beta_i^q \ln w_i^q.$$

where w^q is the set of weights outgoing from q. Now, the same maximum is attained multiplying by $k = \frac{1}{\sum_i \beta_i^q}$. Then, defining $\bar{\beta}_i^q = k\beta_i^q$, we can equivalently find $\arg\max_{w^q} \sum_{i \in ch(q)} \bar{\beta}_i^q \ln w_i^q$, where $\bar{\beta}_i^q$ is positive and sums to 1 and therefore can be

interpreted as a discrete distribution. This is then the maximum of the cross entropy $\arg\max_{w^q}\left(-\mathbb{H}\left(\bar{\beta}_i^q, w_i^q\right)\right)$ defined e.g. in Murphy [2012, 2.8.2], attained for $w_i^q = \bar{\beta}_i^q$, which corresponds to the following update:

$$w_j^{q*} = \beta_j^q / \sum_i \beta_i^q. \tag{3.3.5}$$

Remark: this update for *non-shared weights only* was already derived (with radically different approaches) in Peharz et al. [2016], Zhao et al. [2016b].

Shared weights. In some SPN applications it is necessary to share weights between different Sum Nodes, for instance when a convolutional architecture is used (see e.g. Cheng et al. [2014]). To keep notation simple let us consider only two nodes q_1, q_2 constrained to share weights, that is $w_i^{q_1} = w_i^{q_2} = \hat{w}$ for every child i, where $\hat{w}$ is the set of shared weights. We then rewrite $\mathcal{Q}_W(W)$ insulating the part depending on $\hat{w}$ as $\mathcal{Q}_W(W) = \sum_{i\in ch(q)} \beta_i^{q_1} \ln w_i^{q_1} + \sum_{i\in ch(q)} \beta_i^{q_2} \ln w_i^{q_2} + const$ (the constant includes terms not depending on w^q). Then, employing the weight sharing constraint, maximization of $\mathcal{Q}_W$ for $\hat{w}$ becomes $\arg\max_{\hat{w}} \sum_{i\in ch(q)} \left(\beta_i^{q_1} + \beta_i^{q_2}\right) \ln \hat{w}$. As in the non-shared case, we end up maximizing the cross entropy $-\mathbb{H}\left(k\left(\beta_i^{q_1} + \beta_i^{q_2}\right), \hat{w}_i\right)$. Generalizing for an arbitrary set of nodes D such that any node $q \in D$ has shared weights $\hat{w}$, the weight update for $\hat{w}$ is as follows:

$$\hat{w}_j^* = \frac{\sum_{q\in D} \beta_j^q}{\sum_i \sum_{q\in D} \beta_i^q}. \tag{3.3.6}$$

Note that in both the shared and non shared cases, the EM weight update does not suffer from vanishing updates even in deep nodes thanks to the normalization term. Finally, we note that since all the quantities required in a weight update can be computed with a single forward-downward pass on the SPN, an EM iteration for W has cost *linear in the number of edges.*

3.3.4. Leaf Distribution Updates

We now consider learning leaf distributions. Simplifying $\mathcal{Q}(W, \theta)$ through the use of Proposition 3.1.7 (Appendix A.2.3), the objective function for θ becomes:

$$\theta^* = \arg\max_\theta \mathcal{Q}_\theta(\theta), \tag{3.3.7}$$

$$\mathcal{Q}_\theta(\theta) = \sum_{l\in\mathcal{L}(S)} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\theta_l), \tag{3.3.8}$$

$$\alpha_{ln} = S(x_n)^{-1} \frac{\partial S(x_n)}{\partial S_l} S_l(x_n).$$

where $\mathcal{L}(S)$ denotes the set of leaves of S. The evaluation of terms α_{ln}, which are constant coefficients in the optimization since they depend only on W_{old}, θ_{old}, is the E

step and can be seen as computing the *responsibility* that leaf distribution φ_l assigns to the n-th data point, just as in EM for classical mixture models. Importantly, we note that the maximization problem Eq. 3.3.7 is *concave* as long as $\ln \varphi_l(X_l|\theta)$ is concave, in which case there is an *unique global optimum*. Also note that normalizing α_l in Eq. 3.3.8 dividing each α_{ln} by $\sum_n \alpha_{ln}$ we attain the same maximum and avoid numerical problems due to the usage of very small values.

Non shared parameters. Introducing the hypothesis that parameters θ_l are disjoint at each leaf l, we obtain separate maximizations in the form:

$$\theta_l^* = \arg\max_{\theta_l} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\theta_l)\,. \tag{3.3.9}$$

In this formulation one can recognize a *weighted maximum likelihood* problem, where each data sample n is weighted by a soft-count coefficient α_{ln}.

Shared parameters. Let us consider two leaf nodes k, j associated to distributions $\varphi_k(X_k|\theta_k), \varphi_j(X_j|\theta_j)$ respectively, such that $\theta_k = \theta_j = \hat{\theta}$ are shared parameters. Eq. 3.3.7 for k, j becomes $\mathcal{Q}_\theta(\theta) = \sum_{n=1}^{N} \alpha_{kn} \ln \varphi_k(x_n|\hat{\theta}) + \sum_{n=1}^{N} \alpha_{jn} \ln \varphi_j(x_n|\hat{\theta}) + const(\hat{\theta})$. Generalizing to an arbitrary set of leaves D such that each leaf $l \in D$ has a distribution $\varphi\left(\hat{X_l|\theta}\right)$ and dropping the constant term, we obtain:

$$\hat{\theta}^* = \arg\max_{\hat{\theta}} \sum_{l \in D} \left(\sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l(x_n|\hat{\theta}) \right). \tag{3.3.10}$$

The objective function now contains a sum of logarithms, therefore it cannot be maximized as separate problems over each leaf as in the non shared case. However, it is still *concave* in θ as long as $\ln \varphi_l(X_l|\theta)$ is concave, in which case there is an *unique global optimum* (this holds for exponential families, discussed next). Then, the optimal solution can be found with iterative methods such as gradient ascent or second order methods.

Exponential Family Leaves. For distributions in the exponential family Eq. 3.3.7 is *concave* and therefore a *global optimum* can be reached (see e.g. Murphy [2012, 11.3.2]). Additionally, the solution is often available efficiently in *closed form*. Let us consider two relevant examples. If $\varphi_l(X_l)$ is a *multivariate Gaussian* $\mathcal{N}(\mu_l, \Sigma_l)$, the solution of Eq. 3.3.9 is obtained e.g. in Murphy [2012, 11.4.2] as $r_l = \sum_{n=1}^{N} \alpha_{ln}$, $\mu_l = \frac{\sum_{n=1}^{N} \alpha_{ln} x_n}{r_l}$ and $\Sigma_l = \frac{\sum_{n=1}^{N} \alpha_{ln}(x_n - \mu_l)(x_n - \mu_l)^T}{r_l}$. In this case, EM for SPNs generalizes EM for Gaussian mixture models. If $\varphi_l(X_l)$ is a *tree graphical model* over discrete variables, the solution of Eq. 3.3.9 can be found with the Chow-Liu algorithm (Chow and Liu [1968]) adapted for weighted likelihood (see Meila and Jordan [2000]). The algorithm has a cost quadratic on the cardinality of X and allows to learn jointly the optimal tree structure and potentials.

3.3.5. Convergence for General Leaf Distributions

The EM algorithm proceeds by iterating E-and-M steps (pseudocode in Algorithm 3.2) until convergence. The training set Log Likelihood is guaranteed not to decrease at each step as long as the M-step maximization can be done at least partially Neal and Hinton [1998]: namely, calling $\theta_{l,new}$ and $\theta_{l,old}$ the current and previous parameters of leaf l, this implies EM *converges* if the update at each leaf satisfies:

$$\sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l \left(x_n | \theta_{l,new}\right) \geq \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l \left(x_n | \theta_{l,old}\right). \tag{3.3.11}$$

This condition is very non-constraining, as it simply requires that weighted Log Likelihood can be at least approximately optimized. Note that weighted Log Likelihood maximization requires minor modifications from standard maximum-likelihood. If approximate methods are used, a simple check on the bound Eq. (3.3.11) ensures that the approximate learning procedure did not decrease the lower bound (Algorithm 3.2 row 8).

This allows a very *broad family* of distributions to be used as leaves: for instance, approximate maximum likelihood methods are available for intractable graphical models (Wainwright and Jordan [2008]), probabilistic Neural Networks (**?**), probabilistic Support Vector Machines (**?**) and several non parametric models (see e.g. Geman and Hwang [1982] and Cule et al. [2010]). EM leaf distribution updates can be straightforwardly applied to each of these models. Note that depending on the tractability of the leaf distribution, some operations might not be tractable (e.g. exact marginalization in general graphical models) - whether to use certain distributions as leaves depends on the kind of queries one needs to answer and it is an application specific decision.

Cost. All the quantities required in a EM updates can be computed with a single forward-downward pass on the SPN, thus an EM iteration has cost *linear in the number of edges*. The cost of the maximization for each leaf depends on the leaf type, and it is an application specific problem. For instance, with Gaussian and tree graphical model leaves it is linear in the number of samples.

Algorithm 3.1 Compute $\alpha, \beta\ (S, \{x_1, x_2, ..., x_N\})$

1: **Input:** SPN $S(W, \theta)$, samples $\{x_1, x_2, ..., x_N\}$
2: set $\beta_i^q = 0$ for each $(q, i) \in \mathcal{E}(S)$
3: compute $S_k(x_n), \frac{\partial S(x_n)}{\partial S_k}$ for each node k in S
4: **for** each Sum Node $q \in \mathcal{V}$ **do**
5: **for** each node $i \in ch(q)$ **do**
6: $\beta_i^q \leftarrow \beta_i^q + \frac{1}{N} w_i^q S_i(x_n) \frac{\partial S(x_n)}{\partial S_q} S(x_n)^{-1}$
7: **for** each leaf node $l \in \mathcal{V}$ **do**
8: $\alpha_{ln} \leftarrow S(x_n)^{-1} \frac{\partial S(x_n)}{\partial S_l} S_l(x_n)$

Algorithm 3.2 EMstep($S, \{x_1, x_2, ..., x_N\}$)

Input: SPN $S(W, \theta)$, samples $\{x_1, x_2, ..., x_N\}$
$[\alpha, \beta] \leftarrow$ Compute $\alpha, \beta\,(S, \{x_1, ..., x_N\})$
for each Sum Node q in S and each node $i \in ch(q)$ **do**
    $w_i^q \leftarrow \beta_i^q / \sum_{i \in ch(q)} \beta_i^q$
for each leaf node l in S **do**
    $\theta_l \leftarrow \arg\max_{\theta_l} \sum_{n=1}^{N} \alpha_{ln} \ln \varphi_l\,(x_n|\theta_l)$
    if Eq. 3.3.11 is not satisfied, discard the update

3.4. Structure Learning

Structure learning is a natural application for SPNs due to the tractability of the inference procedure, since the complexity of structure learning essentially depends on the complexity of inference as a subroutine. In addition, learning the structure of models where inference is in general intractable (such as graphical models) has the additional constraint that inference in the resulting model must be tractable (at least approximately).

In recent work, it has been shown empirically that structure learning approaches for SPNs produce state of the art results in density estimation (see e.g. [Gens and Domingos, 2013], [Rooshenas and Lowd, 2014], [Rahman and Gogate, 2016b], [Rahman and Gogate, 2016a]), suggesting that performing exact inference with simpler but tractable models might be a better approach than approximate inference using more complex but intractable models.

In the following we will describe the most relevant methods for learning the structure of SPNs, describing the approaches in chronological order. We will focus on the methods that provided relevant improvements in the density estimation, according to the empirical test-set Log Likelihood results in the benchmark datasets described in the following.

We organize the discussion underlining the *similarities* between the discussed papers rather than the differences due to varying nomenclature and approaches in the paper themselves. In particular, we will see that a split-and-recurse algorithmic structure is a constant characteristic of most of the methods. We also highlight limitations in the described approaches, mainly that the procedures involve a greedy optimization phase where convergence is not guaranteed due to the many employed approximations.

Quantitative Comparison

The structure learning papers considered in this section compare results on 20 binary datasets for density estimation, first introduced in the current form in Gens and Domingos [2013]. The structure of the datasets is provided in Table 3.4.1. All the datasets involve discrete binary variables, in number ranging from 16 to 1556. The number of data samples in the training set ranges from 1670 to 291326. As such, these represent a challenging task where the algorithm is required to be flexible to work both with low and

high dimensional datasets and to scale efficiently with a quite large number of samples. The datasets are divided in a fixed training set, validation set and test set as described in Table 3.4.1.

We report the test set Log Likelihood results for the density estimation models evaluated on each Dataset. Alongside several SPN based approaches, we include in our results a comparison with structure learning algorithms for graphical model of increasing complexity: the ChowLiu algorithm for learning an optimal tree model, Mixtures of Trees trained with Expectation Maximization (section 2.4.3), and cyclic graphical models learned with Microsoft's WinMine Toolkit[1].

We do not report learning and evaluation time for SPNs, since the papers often do not provide this information. However, SPNs are typically much faster to evaluate than other intractable models, since approximate inference methods must be used. For instance, in one measured case (LearnSPN, Gens and Domingos [2013]) the cost of probabilistic queries in SPNs is about two order of magnitudes faster than in WinMine - namely, queries in LearnSPN have an average time of 23 ms and in WinMine 1629 ms. It is worth noting that even allowing Gibbs sampling to run for a very long time did not close the gap in accuracy with SPNs.

It is remarkable that simple structure learning methods obtaining tractable-inference graphical models, namely the Chow-Liu tree and a Mixtures of Trees, can often reach better performances than the complex structure learning methods used in WinMine (see table 3.4.1, columns ChowLiu and MT respectively). This indicates that keeping the structure *simple enough* to permit exact inference and simpler learning can lead to better results than having a complex structure but resorting to approximate inference.

Finally, we note that both tree graphs and mixture of trees can be interpreted as SPNs (section 3.2.3). Therefore, the ChowLiu and the Mixture of Tree methods can be seen also as elementary structure learning algorithms *for SPNs*.

3.4.1. Learning Arithmetic Circuits (Lowd and Domingos [2012])

We start our review of SPN structure learning algorithms by briefly describing a method for learning Arithmetic Circuits, called ACMN (Arithmetic Circuits for learning Markov Networks). We do this for two reasons: First, Arithmetic Circuits are a special case of Sum-Product Networks with indicator variables as leaves (see section 3.2.1), and therefore this can be considered a SPN structure learning algorithm. Second, this method has historical and practical importance, being one of the first competitive structure learning methods for SPN-related architectures and employing a radically different approach with respect to the algorithms that we will describe in the following.

Algorithm Description. The pseudocode for ACMN is shown in algorithm 3.3. We provide here an intuitive description of it based on binary variables for simplicity. Let

[1] Documentation can be found in http://research.microsoft.com/en-us/um/people/dmax/WinMine/Tooldoc.htm. Details of the experiments with WinMine are given in Gens and Domingos [2013]).

$F = \{f_i\}_{i=1}^{M}$ be a set of SPNs (called "features" in the paper). The feature set induces a SPN $S(X) = \prod_{j=1}^{M} f_j(X_j)$.

The algorithm is initialized with a set of features where each feature is a Bernoulli distribution for each variable $X_j \in X : f_j = \mu_j[X_j]_0 + (1 - \mu_j)[X_j]_1$.

At each iterative step of the algorithm, the structure is updated by taking an existing feature $f_j(X_j)$ and combining it with a variable A s.t. $A \cap X_j = \emptyset$, creating two new features $f_j(X_j) \otimes [A]_1$ and $f_j(X_j) \otimes [A]_0$. This operation is referred to as a "split", since it takes an existing feature f and splits it into three cases: $f_j(X_j)$, $f \otimes [A]_1$ and $f \otimes [A]_0$. We denote a split as triplet $s = \{f_j, A\}$. A split s can be inserted in S by creating a Sum Node having has children SPNs $f_j(X_j) \otimes [A]_1$ and $f_j(X_j) \otimes [A]_0$. The Sum Node weights are optimized with line search - we leave the details to Lowd and Domingos [2012].

The selection of *which* split to create during a step of the algorithm is governed by a greedy method, looking for the split with higher gain $G(s)$ as defined as follows:

$$G(s) = \Delta_{LL}(s) - \gamma\Delta_{size}(s). \tag{3.4.1}$$

Here $\Delta_{LL}(s)$ denotes the variation generated on the Log Likelihood of $S(X)$ after inserting s in S, and $\Delta_{size}(s)$ is the variation produced on the number of edges of S. The algorithm terminates once all possible splits are included, or a fixed maximum number of edges has been reached.

Approximating $\Delta_{LL}(s)$ and $\Delta_{size}(s)$. Not only the algorithm is based on a greedy method, but the quantity needed for this method cannot be computed exactly and thus need approximations. Assuming that Δ_{LL} and Δ_{size} could be computed exactly, then the algorithm would converge to the maximum likelihood solution. However, computing these quantity exactly would require jointly optimizing all model parameters along with the parameters for the two new features, which is unfeasible. Therefore, several simplifications are made to compute Δ_{LL} and Δ_{size}.

First, Δ_{LL} is taken as the Log Likelihood gain obtained by modifying only the weights of the two new features, keeping all others fixed. This provides a lower bound on the actual Log Likelihood gain. Second, ACMN employ the assumption that, as learning progresses, the score of any given split decreases monotonically. The score of a split can decrease for two reasons. While this assumption does not always hold, it allows to evaluate only a small fraction of the available splits in each iteration.

Discussion. This algorithm obtained state of the art results on the benchmark datasets at the time of publication (Table 3.4.1, column ACMN). However, it has a number of drawbacks, since employs a number of empirically justified approximations in order to keep the run time tractable, it depends on many hyper parameters, and it tends to generate very large models in comparison to competing approaches. An interesting aspect is that the splitting procedure generates nodes with observed variables at internal Sum Nodes, in contrast to most of the methods described subsequently.

Algorithm 3.3 LearnACMN(D)

initialize SPN S as product of marginals computed over D
initialize priority queue Q by ranking initial splits with Eq. (3.4.1)
loop until Log Likelihood convergence
    Update edge gain Δe and likelihood gain Δll for each possible split in Q
    $s = Q$.pop() // Select best split s
    sn = InsertSplit(S, s) // Insert the best split in S creating Sum Node sn
    OptimizeWeights(sn) // Optimize the weights of the new Sum Node
    for each variable $A \in V$ **do**
        Add new features $(f \otimes [A]_1)$ and $(f \otimes [A]_0)$ to Q.
return (M, C)

3.4.2. LearnSPN (Gens and Domingos [2013])

The first structure learning method developed specifically for SPNs appeared in Gens and Domingos [2013]. The simple split-and-recurse procedure employed by this algorithm influenced many subsequent papers which employ small variations of this method, and it is therefore worth describing here in detail. The pseudocode for this method is shown in Algorithm 3.4. We report here an intuitive description of the algorithm.

Consider a set of variables $V = \{X_k\}_{k=1}^{M}$, and a set of i.i.d. samples $T = \{x^i\}_{i=1}^{N}$. Let the samples be organized in a matrix $D \in \mathbb{R}^{N \times M}$ with one sample per row, where the element D_{ij} corresponds to the j-th variable of the i-th sample. At each recursive step LearnSPN(D) employs these rules:

1. If it is possible to *partition* the variables V in approximately independent subsets $\{D_k\}_{k=1}^{K}$, then create a product of the SPNs obtained calling LearnSPN on each subset: $\prod_{k=1}^{K} LearnSPN(D_k)$.

2. If it is possible to *cluster* the samples (rows of D) into groups of "similar" instances, create a mixture model over the SPNs obtained calling LearnSPN on each subset: $\sum_{k=1}^{K} w_k LearnSPN(D_k)$, where weights w_k are proportional to the number of instances in subset D_k.

3. If a certain termination condition is met, then create a SPN representing the product of empirical marginals distributions computed over variables in D.

The authors prove that in the extreme case when the variables are independent and instances in the same group are identical, then LearnSPN attains the optimal Maximum Likelihood estimator. However, these conditions are almost never satisfied in practice. Hence, this approach becomes a greedy algorithm in which there is there is no proof of convergence. Despite these limitations, LearnSPN managed to obtain very good empirical results on density estimation. Test set Log Likelihood results corresponding to this method on the benchmark datasets are presented in Table 3.4.1, column LearnSPN.

LearnSPN as an Algorithm Schema. Rather than being a single algorithm, LearnSPN is an *algorithm schema* which depends on the choices of the functions which test for independence, similarity and termination. The choice of such functions is then a crucial design step that allows for flexible applications of the algorithm: for instance, variations of this base schema have been employed in the algorithms described in sections 3.4.3,3.4.4, 3.4.5 and 3.4.6.

Gens and Domingos [2013] used simple but very fast tests of independence, similarity and termination:

- The termination condition is $|V| = 1$, that is, the data is recursively split until a single variable is present.
- The search for partition over almost independent subsets of variables is performed as follows. First, we create a matrix $G \in \mathbb{R}^{M \times M}$ in which element $G_{i,j}$ is a measure of the pairwise dependence between the i-th and j-th variables. The chosen measure is the G-test of pairwise independence, defined as:

$$G(A, B) = 2 \sum_{a \in \Delta(A)} \sum_{b \in \Delta(B)} N_{ab}\left(A = a, B = b\right) \frac{\log N_{ab}\left(A = a, B = b\right) |T|}{\log N_a\left(A = a\right) N_b\left(B = b\right)}.$$

  Here, $N_{ab}\left(A = a, B = b\right)$ counts the number of times the setting of variables $A = a, B = b$ appears in the dataset (similarly for $N_a\left(A = a\right)$). Then, the elements of I which are below a threshold $k > 0$ (which is a hyperparameter) are set to zero, and the resulting matrix is interpreted as a connection matrix defining a weighted graph. Finally, this graph is partitioned in sets of connected component, and each set is taken to represent a subset of approximatively independent variables.
- To cluster instances the authors use incremental EM (Neal and Hinton [1998]) over a naive Bayes mixture model with exponential prior $P(S) \propto e - \lambda C|V|$, where C is the number of clusters and λ is the cluster penalty.

This procedure heavily depends on the choice of hyerparameters k and λ, tuned with cross validation.

3.4.3. SPNs with Direct and Indirect Variable Interactions (IDSPN, Rooshenas and Lowd [2014])

The structure learning method described in this section combines the approaches described in Sections 3.4.2 (SPNLearn) and 3.4.1 (ACMN). As discussed in Section 3.4.2, LearnSPN allows is an algorithm schema that allows to employ arbitrary leaf models when the termination conditions are met (base case in algorithm 3.4). Intuitively, IDSPN employs a variation of LearnSPN where the leaf modes are ACMNs as discussed in Section 3.4.1. The rationale behind this idea is to combine the advantages of LearnSPN (learning a large, hierarchical mixture model) and of ACMN (which employs observed latent variables

Algorithm 3.4 LearnSPN(T, V)

Input: set of instances T and set of variables V
Output: an SPN representing a distribution over V learned from T

if conditions for inducing the base case are satisfied **then**
    // 1. Base Case
    return LearnLeafModel(T, V)
else if V can be partitioned into approximatively independent subsets V_j **then**
    // 2. Variable Splitting Step
    return $\prod_j$ LearnSPN(T, V_j)
else// 3. Clustering Step
    Partition T into subsets of similar instances T_i
    return $\sum_i \frac{|T_i|}{|T|} LearnSPN(T_i, V)$

and thus learns different kind of relationships between variables). These concepts are not well defined: as often happens in machine learning models, this approach is based on intuitive ideas and extensive empirical evaluation rather than on a mathematically motivated analysis.

Since this method combines the two procedures above, it also inherits all their hyperparameters, plus some additional ones that regulate when to perform the base step. Furthermore, it tends to create much larger models than LearnSPN - for details see Zhao et al. [2016b]. Its use has been justified by the fact that it improved performances on the benchmark datasets, possibly due to the combination of the two different approaches in an unique setting. We will avoid reporting the algorithm details because it is quite convoluted, rich of hyperparameters, and we are not aware of subsequent papers that improve on it. Test set Log Likelihood results are shown in Table 3.4.1, column IDSPN.

3.4.4. Learning the Structure of SPNs by Finding Low Rank Submatrices (SPD-SVN, Adel et al. [2015])

The method described in Adel et al. [2015] (SPD-SVN) is an application of the LearnSPN scheme described in Section 3.4.2, similarly to the methods described previously. In this case, the tests for independence and clustering are based on finding the submatrix of data matrix $D \in \mathbb{R}^{N \times M}$ of rank closest to 1. The authors *postulated* that finding a low rank submatrix of D corresponds to finding a subset of both variables and samples in which samples are strongly correlated: for instance, in the binary case if the rank is 1 all the samples are equal (being linearly dependent and binary variables). However, no formal argument is provided for the general case. The search for rank 1 sub matrices is performed with a greedy, convergent algorithm called Hilbert-Schmidt independence criterion (Gretton et al. [2005]), which includes a penalty term for penalizing smaller sub matrices (trivially, any 1×1 matrix has at most rank 1) and has asymptotic cost $O(M \log M)$ and thus quickly converges to a solution.

Once a low rank submatrix has been found, then the variable split and sample clustering is performed as graphically represented in Fig. 3.4.1 and described in Algorithm 3.5. Notice that instead than looking for variables that are not correlated and hence should be considered independent, this algorithm looks for variables that are maximally correlated.

Test set Log Likelihood results of this method are shown in Table 3.4.1, column SPD-SVN. This approach is empirically very competitive, and the method itself is generally faster than competing methods due to the cheapness of the rank-1 matrix search. However, we have to remark that *no implementation* of the code was made available by the authors, and that we failed to reproduce their results rewriting our version of their algorithm. In addition, there is no follow up paper and thus we failed to find any independent verification of the results of this paper.

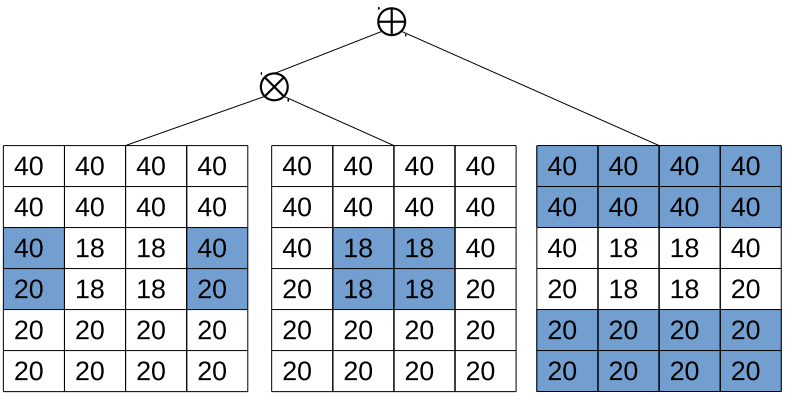
Algorithm 3.5 SPN-SVD(D)

Input: Instance matrix $D \in \mathbb{R}^{M \times N}$
Output: SPN S

//extract row (R) and column (C) indices of the submatrix of D with lowest rank
$[R\ ,\ C]$ = ExtractRank1(D, γ)
//1. Base Case
if D is a vector **or** no submatrix was found **then**
    return product of empirical univariate marginals of D
Set Cc as the columns of D excluding C
Set Rc as the rows of D excluding R
Set $B_1 = D(R, C)$, $B_2 = D(R, Cc)$ and $B_3 = A(Rc, :)$
S_1 = SPN-SVD(B_1)
S_2 = SPN-SVD(B_2)
//2. Variable Splitting Step
Create Product Node Pn = Prd(S_1 , S_2)
S_3 = SPN-SVD(B_3)
//3. Clustering Step
Create Sum Node Sn = Sum(S_3, Pn) with weights $(\frac{N-|R|}{N}, \frac{|R|}{N})$
Create SPN S with root Sn
return S

3.4.5. Learning Cutset Networks (CNets) and Ensembles of CNets (Rahman et al. [2014])

Cutset Networks (CNets, Rahman et al. [2014]) are an architecture that can be seen as a special case of SPNs (see section 3.2.1), even though they were recently introduced as a new architecture alongside a structure learning algorithm. Structure learning for CNets proceeds using a recursive procedure following similar steps to the one described in section 3.4.2. We provide here an intuitive description of the algorithm, whose pseudocode is shown in Algorithm 3.6.



40	40	40	40
40	40	40	40
40	18	18	40
20	18	18	20
20	20	20	20
20	20	20	20

(a) The data matrix D and a minimum rank subset of D (in gray).

40	40	40	40	40	40	40	40	40	40	40	40
40	40	40	40	40	40	40	40	40	40	40	40
40	18	18	40	40	18	18	40	40	18	18	40
20	18	18	20	20	18	18	20	20	18	18	20
20	20	20	20	20	20	20	20	20	20	20	20
20	20	20	20	20	20	20	20	20	20	20	20

(b) The SPN structure resulting from one step of SPD-SVN over D. SPN-SVD is then called recursively on the submatrices in blue, which form a partition of D.

Figure 3.4.1. - Graphical representation of a recursive step of SPD-SVN.

As in previous approaches, let $D \in \mathbb{R}^{N \times M}$ be a binary data matrix where samples are arranged into rows, and let V be the set of all variables in the model. The algorithm includes two cases:

- If some termination condition on D is satisfied, a leaf model is created. The leaf model is a tree graphical model obtained by running the Chow-Liu algorithm on D.

- If the terminating condition is not satisfied a *variable split* is performed: given $A \in V$, D is partitioned in two matrices $D_{A,1}$ and $D_{A,0}$ that contain respectively all the samples in which A has value 1 and 0, and do not contain the column corresponding to A. Then, the model is called recursively in the two subsets $D_{A,1}$ and $D_{A,0}$ according to the following SPN structure: $LearnCNET(D_{A,0}) \otimes [A]_0 \oplus LearnCNET(D_{A,1}) \otimes [A]_1$. Note that this method employs *observed variables* due to the presence of indicator variables at the Product Nodes (section 3.2.1).

A crucial choice in this algorithm is finding a good heuristic for the variable split. The authors propose to find the split that minimizes the sum of the empirical entropies of the matrices $D_{A,1}$ and $D_{A,0}$. However, as there is often not enough data to reliably measure the joint entropy, the authors propose to approximate it with the average per variable entropy $\hat{H}(D) = \frac{1}{|V|} \sum_{A \in V} H_D(A)$, where $H_D(V)$ denotes the empirical entropy of variable V in the data matrix D. Then, the splitting heuristic results in choosing the variable V that maximizes the following gain, representing the expected decrease in entropy given a split over variable A:

$$Gain_D(A) = \hat{H}(D) - \sum_{a \in \Delta(A)} \frac{|D_{A=a}|}{|D|} \hat{H}(D_{A=a}).$$

Taking inspiration from the literature on Or trees, an additional pruning step can be optionally performed after the CNet has been constructed, by substituting branches of the CNet with a single Chow-Liu tree if the validation Log Likelihood increases.

Ensembles of CNets

Mixtures of CNets can be learned using the Expectation-Maximization algorithm for mixture models (section 2.4.3). At each iterative step of EM, the M-step is performed by learning CNets with weighted data, where the weights are the responsibilities computed in the E-step. The extension of LearnCNet to weighted data can be obtained by computing the weighted entropy (section 2.1.2) and weighted Chow-Liu Tree (section 2.3.4) rather than the corresponding unweighted versions. Test set Log Likelihood of mixtures of CNets in the benchmark datasets far exceed results of the single CNets (Table 3.4.1, column MCnet).

An algorithm that learns ensembles of CNets by boosting is described in Rahman and Gogate [2016a]. This structure learning method can choose between both observed and unobserved variable splits, and as a result it has several hyper parameters. It is mostly justified by its excellent empirical performances, since has state of the art results in many of the benchmark datasets (Table 3.4.1, column ECnet).

Algorithm 3.6 LearnCNet(T, V)

Input: set of instances T and set of variables V
Output: an SPN representing a distribution over V learned from T

if conditions for inducing the base case are satisfied **then**
  //Base Case
  return ChowLiuTree(T, V)
else
  //Heuristically select the variable A which splits the data
  //in submatrices of minimum entropy
  A =BestSplit(T, V)
  $T_0 = T^i \in T : A = 0$ in T_i
  $T_1 = T^i \in T : A = 1$ in T_i
  create weights $w_0 = \frac{|T_0|}{|T|}, w_1 = \frac{|T_1|}{|T|}$
  $S_0 = LearnCNet(T_0, V \setminus A)$
  $S_1 = LearnCNet(T_1, V \setminus A)$
  //Variable Splitting and Clustering Steps
  create SPN S as follows: $S = w_0[A]_0 S_0 + w_1[A]_1 S_1$
  return S

3.4.6. Learning Graph SPNs by Merging (Rahman and Gogate [2016b])

The structure learning that we have discussed in this section produce SPNs with a tree structure: this is due to the fact that these algorithms recursively split the dataset and separate sub-SPNs for each split. However, using graph SPNs rather than tree SPNs is an attractive idea for two reasons. First, some SPNs can be represented exponentially more compactly as a graph than as a tree, by exploiting reuse of shared parts of the DAG (see Section 3.1.1). Second, graph SPNs can potentially improve the generalization performance by addressing a problem intrinsic to the previous algorithms: as the depth of the node increases, the number of training examples used to learn the sub-SPN rooted at the node decreases exponentially, and thus parameter estimates are more noisy.

One approach to solve this problem is to post-process SPNs obtained with LearnSPN (Section 3.4.2) by merging sub-SPNs structures. Merging increases the number of examples available to learn, and in turn reduces the variance of the parameter estimates while having no effect on their bias (section 2.1) and can therefore improve generalization. This intuition is formalized by the following proposition, proved in Rahman and Gogate [2016b]:

Proposition 3.4.1. *Let $S_1(X)$, $S_2(X)$ and $S_{1,2}(X)$ be SPNs having the same graph but whose parameters are estimated from training examples T_1, T_2 and $T_{1,2} = T_1 \cup T_2$ respectively. Then assuming that the datasets are generated uniformly at random from a distribution whose structure decomposes according to S_1 (and thus also according to S_2 and $S_{1,2}$), the variance of samples generated from $S_{1,2}$ is smaller than the variance of samples from S_1 and S_2.*

The main idea of the approach in Rahman and Gogate [2016b] is to relax the identity condition employed in the proposition above and merge sub-SPNs that are *similar*. This creates the necessity to deploy methods for detecting which SPNs to merge, and how to merge them. For detecting which SPNs to merge, the authors define the following approximation of the distance between two distributions P and Q over variables V, which assumes approximate factorization of the distribution over each variable:

$$D\left(P|Q\right) \approx \frac{1}{|V|} \sum_{V_i \in V} D\left(P(V_i)|Q(V_i)\right).$$

where D is a distance function (e.g. the Kullback-Leibler divergence described in section 2.1.2). Since single-variable marginals distributions in each SPN can be computed in time and memory that is linear in the number of nodes of the SPN, measuring this distance takes also linear time and memory.

The pseudocode for the proposed post-processing algorithm is in Algorithm 3.7. Taking as input a SPN S, the algorithm begins by initializing a second SPN S' identical to S and repeats the following steps until convergence: First, take the set of sub-SPNs $\{S_i\}$ of S that are defined over exactly i variables, and partition this set into subsets $\{\rho_j\}$ such that all the SPNs in ρ_j have the same scope. Then, merge all sub-SPNs S_1, S_2 which have the

same scope and such that the distance $D(S_1|S_2)$ is less than some constant value ϵ (using the subroutine Merge(S_1, S_2) described later). Another option is to merge two sub-SPNs if the accuracy on the validation set improves (in experiments, both approaches were used).

The subroutine Merge(S_1, S_2) consists in re-learning a single SPN from the union of the data that was used to learn the SPNs S_1, S_2. The authors propose to re-learn the structure only if both SPNs are a Chow-Liu tree leaf, and otherwise to keep the structure of one of the merged SPNs and re-learn the weights only.

The results of this algorithm in test set Log Likelihood show improved generalization over the tree SPN counterparts. Results on the benchmark datasets are shown in Table 3.4.1 under the column MergeSPN.

Algorithm 3.7 MERGE-SPN (S, V, ϵ)

Input: SPN S
Output: Merged SPN S

```
S' = S
repeat
    for i = 1 to |V| do
        S_i = sub-SPNs in S having exactly i variables in their scope
        ρ = Partition S_i into cells having identical scopes
        for each ρ_j ∈ ρ do
            Merge all sub-SPNs in ρ_j such that the
            distance between them is bounded by ε and S is a DAG
            Merge(S_1, S_2)
until S does not change
return S
```

3.5. Overview of Applications of SPNs

The application oriented papers exploiting Sum-Product Networks are still limited, since this field has developed just in the last six years. However, relevant results in several fields, where SPNs outperformed all other probabilistic and deterministic methods, aroused interest in the theoretical and practical properties of these models.

Density Estimation has been the most relevant application field for SPNs. Papers that use SPNs for density estimation typically propose a new structural learning procedure to learn the SPN from the given dataset and test it on the benchmark datasets described in Section 3.4. The main contributions in this field, which were already discussed in detail in section 3.4, are Lowd and Domingos [2012], Gens and Domingos [2013], Rahman et al. [2014], Adel et al. [2015], Rahman and Gogate [2016b,a].

SPN were first applied to *Image Processing* in the original SPN paper (Poon and Domingos [2011]), and subsequently in Gens and Domingos [2012], Wang et al. [2014]

and Adel et al. [2015]. A particularly successful image processing application of SPNs was on recognizing activities from images, described in Amer and Todorovic [2016]. SPNs are suited to this task because they can compactly encode mixtures of a subset of base parts. Furthermore, their possibility of obtaining samples from the model allows to have generative image models.

SPNs were also used with some success in the field of *Language Processing* (Cheng et al. [2014]Peharz et al. [2014]) and *Sequence Modeling* Melibari et al. [2015], where however still achieve performances far from the state of the art methods.

Table 3.4.1. - Test set Log Likelihood on 20 benchmark datasets for the structure learning algorithms discussed in Section 3: ACMN (Lowd and Domingos [2012]), LearnSPN (Gens and Domingos [2013]), IDSPN(Rooshenas and Lowd [2014]), MergeSPN (Rahman and Gogate [2016b]), CNet and MCnet (Rahman et al. [2014]), ECnet (Rahman and Gogate [2016a]), Mixture of Trees (Meila and Jordan [2000]), ChowLiu (Chow and Liu [1968]), WinMine Corporation,LTM(Choi et al. [2011]).

Dataset	#vars	#train	#valid	#test	ACMN	LearnSPN	ID-SPN	MergeSPN	CNet	MCNet	ECNet	Mix.Trees	ChowLiu	WinMine	LTM
NLTCS	16	16181	2157	3236	**-6.00**	-6.11	-6.02	**-6.00**	-6.10	**-6.00**	**-6.00**	-6.01	-6.76	-6.025	-6.49
MSNBC	17	291326	38843	58265	**-6.04**	-6.11	**-6.04**	-6.10	-6.06	**-6.04**	-6.05	-6.07	-6.54	-6.041	-6.52
KDDCup2K	64	180092	19907	34955	-2.17	-2.18	-2.13	**-2.12**	-2.21	**-2.12**	-2.13	-2.13	-2.32	-2.189	-2.18
Plants	69	17412	2321	3482	-12.80	-12.98	-12.54	**-12.03**	-13.37	-12.74	-12.19	-12.95	-16.51	-12.65	-16.39
Audio	100	15000	2000	3000	-40.32	-40.50	-39.79	**-39.49**	-46.84	-39.73	-39.67	-40.08	-44.35	-40.50	-41.90
Jester	100	9000	1000	4116	-53.31	-53.48	-52.86	-52.47	-64.50	-52.57	**-52.44**	-53.08	-58.21	-51.07	-55.17
Netflix	100	15000	2000	3000	-57.22	-57.33	-56.36	**-55.84**	-69.74	-56.32	-56.13	-56.74	-60.25	-57.06	-58.53
Accidents	111	12758	1700	2551	-27.11	-30.04	-26.98	-29.32	-31.59	-29.96	-29.25	-29.63	-33.17	-26.32	-33.05
Retail	135	22041	2938	4408	-10.88	-11.04	-10.85	-10.82	-11.12	-10.82	**-10.78**	-10.83	-11.02	-10.87	-10.92
Pumsb-star	163	12262	1635	2452	-23.55	-24.78	-22.40	-23.67	-25.06	-24.18	-23.34	-23.71	-30.80	-21.72	-31.32
DNA	180	1600	400	1186	-80.03	-82.52	-81.21	-80.89	-109.79	-85.82	-80.66	-85.14	-87.70	-80.65	-87.60
Kosarek	190	33375	4450	6675	-10.84	-10.99	-10.60	-10.55	-11.53	-10.58	**-10.54**	-10.62	-11.52	-10.83	-10.87
MSWeb	294	29441	32750	5000	-9.77	-10.25	-9.73	-9.76	-10.20	-9.79	**-9.70**	-9.85	-10.35	-9.67	-10.21
Book	500	8700	1159	1739	-36.56	-35.89	-34.14	-34.25	-40.19	-33.96	**-33.78**	-34.63	-37.84	-36.41	-34.22
EachMovie	500	4524	1002	591	-55.80	-52.49	-51.51	**-50.72**	-60.22	-51.39	-51.14	-54.60	-64.79	-54.37	†
WebKB	839	2803	558	838	-159.13	-158.20	-151.84	**-150.04**	-171.95	-153.22	-150.10	-156.86	-164.89	-157.43	-156.84
Reuters-52	889	6532	1028	1540	-90.23	-85.07	-83.35	**-80.66**	-91.35	-86.11	-82.19	-85.90	-96.85	-87.56	-91.23
20Newsgrp.	910	11293	3764	3764	-161.13	-155.93	-151.47	**-150.80**	-176.56	-151.29	-151.75	-154.24	-164.99	-158.95	-156.77
BBC	1058	1670	225	330	-257.10	-250.69	-248.93	**-233.26**	-300.33	-250.58	-236.82	-261.84	-261.41	-257.86	-255.76
Ad	1556	2461	372	491	-16.53	-19.73	-19.00	-14.34	-16.31	-16.68	-14.36	-16.02	-16.67	-18.35	†

4. Sum-Product Graphical Models

This chapter describes Sum-Product Graphical Models, the architecture that constitutes the main contribution of this thesis. In Section 4.1 we provide a formal intrpduction of the model and a discussion of its properties. In Section 4.2 we discuss learning the parameters and structure of SPGMs.

4.1. Sum-Product Graphical Models

This section formally introduces *Sum-Product Graphical Models (SPGMs)* and discusses their properties. We provide an interpretation of the model as a mixture of tree GMs in Section 4.1.3, and as a high-level representation of SPNs in Section 4.1.4.

4.1.1. Definition

The first step in describing SPGMs is to define the type of nodes that appear in the underlying graph.

Definition 4.1.1. [Sum, Product and Variable Nodes] Let X and Z be disjoint sets of discrete variables, and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG.

- The basic nodes $s \in \mathcal{V}$ of a SPGM are called *SPGM Variable Node (Vnode)* and associated with a variable $X_s \in X$, They are graphically represented as a circle having X_s as label (Fig. 4.1.1, left).
- $s \in \mathcal{V}$ is called *Sum Node*, if it represents the corresponding operation indicated by the symbol $\oplus$. A Sum Node can be *Observed*, in which case it is associated to a variable $Z_s \in Z$ and represented by the symbol $\oplus Z_s$.
- $s \in \mathcal{V}$ is called *Product Node*, if it represents the corresponding operation indicated by the symbol $\otimes$.

In what follows, variables Z take the role of *context variables* according to Definition 2.5.1.

Definition 4.1.2. [Scope of a node] Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a rooted DAG with nodes as in Definition 4.1.1, and let $s \in \mathcal{V}$.

- The *scope* of s is the set of all variables associated to nodes in the sub-DAG rooted in s.

- The *X-scope* of s is the set of all variable associated to Vnodes in the sub-DAG rooted in s.
- The *Z-scope* of s is the set of variables associated to Observed Sum Nodes in the sub-DAG rooted in s.

Example 15. The scope of the Sum Node associated to Z_2 in Fig. 4.1.1a is $\{D, E, F, Z_2\}$, its Z-scope is $\{Z_2\}$, and its X-scope is $\{D, E, F\}$.

Finally, we define the set of "V-parents" of a Vnode s, which intuitively are the closest Vnode ancestors of s.

Definition 4.1.3. [Vparent] The *Vparent set* $vpa(s)$ of a Vnode s is the set of all $r \in \mathcal{V}$ such that r is a *Vnode*, and there is a directed path from r to s that does not include any other Vnode.

With the definitions above we can now define SPGMs.

Definition 4.1.4. [SPGM] A *Sum-Product Graphical Model* $\mathbb{S}(X, Z|\mathcal{G}, \{P_{st}\}, \{W_s\}, \{Q_s\})$ or more shortly, $\mathbb{S}(X, Z)$ or even $\mathbb{S}$, is a rooted DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where nodes can be Sum, Product or Vnodes as in Definition 4.1.1. The SPGM is governed by the following parameters:

1. Pairwise conditional probabilities $P_{st}(X_t|X_s)$ associated to each Vnode $t \in \mathcal{V}$ and each Vparent $s \in vpa(t)$.
2. Unary probabilities $P_s(X_s)$ associated to each Vnode $s \in \mathcal{V}$: $vpa(s) = \emptyset$.
3. Unary probabilities $W_s(k)$ for $k = \{1, 2, ..., |ch(s)|\}$ associated to each non-Observed Sum Node s, with value $W_s(i)$ associated to the edge between s and its i-th child (assuming any order has been fixed).
4. Unary probability $Q_s(Z_s)$ s.t. $\Delta(Z_s) = \{1, 2, ..., |ch(s)|\}$ and associated to each Observed Sum Node s, with value $Q_s(i)$ being associated to the edge between s and its i-th child.

In addition, each node $s \in \mathcal{V}$ must satisfy the following conditions:

1. If s is a *Vnode* (associated to variable X_s), then s has *at most* one child c, and X_s does not appear in the scope of c.
2. If s is a *Sum* Node, then s has *at least* one child, and the scopes of all children are the *same* set. If the Sum Node is Observed (hence associated to variable Z_s), then Z_s is not in the scope of any child.
3. If s is a *Product* Node, then s has *at least* one child, and the scopes of all children are *disjoint* sets.

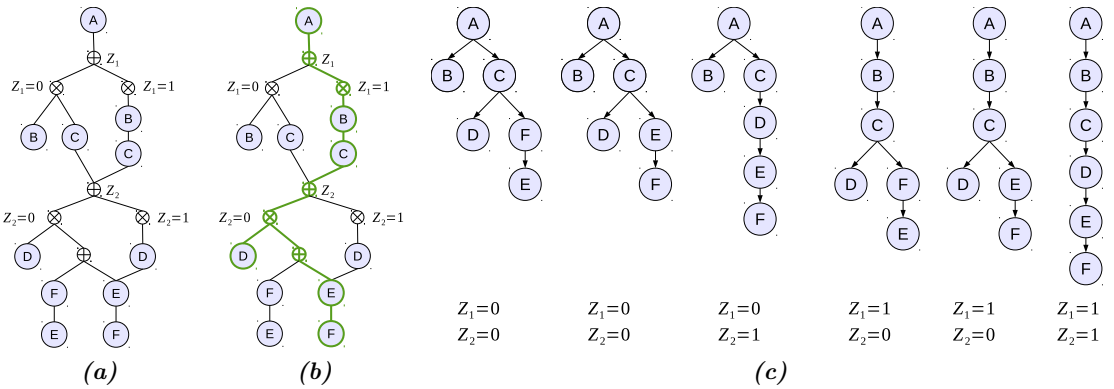


Figure 4.1.1. - Sum-Product Graphical Model Example. (a) A SPGM $\mathbb{S}(X, Z)$ with $X = \{A, B, C, D, E, F\}, Z = \{Z_1, Z_2\}$. (b) A subtree of $\mathbb{S}$ (in green). (c) All subtrees of $\mathbb{S}$ are represented as graphical models and corresponding context variables.

An example SPGM is shown in Fig. 4.1.1, left. Note that the closeness with both SPNs and GMs, to be further discussed later, can be already seen from the definition: the last three conditions in definition are closely related to SPN conditions (Definition 3.1.3), whereas the usage of pairwise and unary probabilities in 1.-4. above connects SPGMs to graphical models.

4.1.2. Message Passing in SPGMs

We now define a message passing protocol used to evaluate SPGMs, which conforms to the way how SPGMs represent conditional and contextual independence efficiently. The following definition refers to Definitions 4.1.3 and 4.1.4.

Definition 4.1.5. [Message passing in SPGMs] Let $s \in \mathcal{V}, t \in \mathcal{V}$ and let $ch(s)_k$ denote the k-th child of s in a given order. Node t sends a message $\mu_{t\to s;j}$ to each Vparent $s \in vpa(t)$ and for each parent state $j \in \Delta(X_s)$ according to the following rules:

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} [Z_t]_k Q_t(k) \mu_{ch(t)_k \to s;j} \quad t \text{ Sum Node, Observed} \tag{4.1.1a}$$

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} W_t(k) \mu_{ch(t)_k \to s;j} \quad t \text{ Sum Node, not-Observed} \tag{4.1.1b}$$

$$\mu_{t\to s;j} = \prod_{q\in ch(t)} \mu_{q\to s;j} \quad t \text{ Product Node} \tag{4.1.1c}$$

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}(k|j) [X_t]_k \mu_{ch(t)\to t;k} \quad t \text{ Vnode} \tag{4.1.1d}$$

If $vpa(s)$ is empty, top level messages are computed as:

$$\mu_{t\to root} = \sum_{k=1}^{|ch(t)|}[Z_t]_k Q_t(k)\mu_{ch(t)_k\to root} \quad t \text{ Sum Node, Observed} \tag{4.1.1e}$$

$$\mu_{t\to root} = \sum_{k=1}^{|ch(t)|} W_t(k)\mu_{ch(t)_k\to root;j} \quad t \text{ Sum Node, not-Observed} \tag{4.1.1f}$$

$$\mu_{t\to root} = \prod_{q\in ch(t)} \mu_{q\to root;j} \quad t \text{ Product Node} \tag{4.1.1g}$$

$$\mu_{t\to root} = \sum_{k\in\Delta(X_t)} P_s(k)\,[X_t]_k \mu_{ch(t)\to t;k} \quad t \text{ Vnode} \tag{4.1.1h}$$

Vnodes at the leaves send messages as in Eqns. (4.1.1d) and (4.1.1h) after substituting the incoming messages by the constant 1.

Note that that messages are only sent to Vnodes (or to a fictitious "root" for top level nodes), and no message is sent to Sum and Product Nodes. Notice further that Vnode messages resemble message passing in tree GMs (Eq. (2.3.6)), which is the base for our subsequent interpretation of SPGMs as graphical model.

Definition 4.1.6. [Evaluation of $\mathbb{S}(X,Z)$] Let $Y \subseteq X \cup Z$ denote evidence variables with assignment $y \in \Delta(Y)$. The evaluation of a SPGM $\mathbb{S}$ with assignment y, written as $\mathbb{S}(y)$, is obtained by setting the indicator variables accordingly (Definition 2.2.2), followed by evaluating messages for each node from the leaves to the root due to Definition 4.1.5, and then taking the value of the message produced by the root of $\mathbb{S}$.

Proposition 4.1.1. *The evaluation of a SPGM $\mathbb{S}$ has complexity $O(|\mathcal{V}|M|\Delta_{max}|^2)$, where M is the maximum number of Vparents for any node in $\mathbb{S}$, and $|\Delta_{max}| = \max\{\Delta(X_s)\colon s \in \mathcal{V}\}$ is the maximum domain size for any variable in X.*

Proof. Every message is evaluated exactly once. Each of the $|\mathcal{V}|$ nodes sends at most M messages (one for each Vparent), and each message has size $|\Delta_{max}|^2$ (one value per every state of sending and receiving node). □

4.1.3. Interpretation of SPGMs as Graphical Models

In this section, we consider and discuss SPGMs as probabilistic models. We show that SPGMs encode large mixtures of trees with shared subparts and provide a high-level representation of both conditional and contextual independence through D-separation.

Subtrees

We start by introducing subtrees of SPGMs and their properties.

Definition 4.1.7. [Subtrees of SPGMs]

Let $\mathbb{S}$ be a SPGM. A *subtree* $\tau(X,Z)$ (or more shortly τ) is a SPGM defined on a subtree of the DAG $\mathcal{G}$ underlying $\mathbb{S}$ (cf. Def. 4.1.4), that is recursively constructed based on the root of $\mathbb{S}$ and the following steps:

- If s is a *Vnode or a Product Node*, then include in τ all children of s and edges formed by s and its children. Continues this process for all included nodes.
- If s is a Sum Node, then include in τ only the k_s-th child and the corresponding connecting edge, where the choice of k_s is arbitrary. Continue this process for all included nodes.

We denote by $\mathcal{T}(\mathbb{S})$ the set of all subtrees of $\mathbb{S}$.

Example 16. One of the subtrees of the SPGM depicted in Fig. 4.1.1a is shown by Fig. 4.1.1b.

Definition 4.1.8. [Subtrees τ and indicator variable sets z_τ]
Let $\tau \in \mathcal{T}(\mathbb{S})$ be a subtree of $\mathbb{S}$. The symbol z_τ denotes the set of all indicator variables associated to Observed Sum Nodes and their corresponding state in the subtree. Specifically, if the k_s-th child of an Observed Sum Node s is included in the tree, then $[Z_s]_{k_s} \in z_\tau$.

Example 17. The set z_τ for the subtree in Fig. 4.1.1b is $\{[Z_1]_1, [Z_2]_0\}$.

Definition 4.1.9. [Context-compatible subtrees] Let $Y \subseteq Z$ be a subset of context variables with assignment $y \in \Delta(Y)$, and let $[y]$ denote the set of indicator variables corresponding to $Y = y$. The set of subtrees compatible with context $Y = y$, written as $\mathcal{T}(\mathbb{S}|y)$, is the set of all subtrees $\tau \in \mathcal{T}(\mathbb{S})$ such that $[y] \subseteq z_\tau$.

Example 18. The set of subtrees $\mathcal{T}(\mathbb{S}|Z_1 = 0, Z_2 = 0)$ for the SPGM in in Fig. 4.1.1 is composed by subtree τ, shown in Fig. 4.1.1b, and the subtree obtained by modifying τ through choosing the alternate child of the lowest sum node.

We now state properties of subtrees that are essential for the subsequent discussion.

Proposition 4.1.2. *Any subtree $\tau \in \mathcal{T}(\mathbb{S})$ is a tree SPGM.*

Proof. Only Product Nodes in $\tau \in \mathcal{T}(\mathbb{S})$ can have multiple children, since Vnodes have a single child by Definition 4.1.4, case 1, and Sum Nodes have a single child in τ by Definition 4.1.7. Children of Product Nodes have disjoint graphs by Definition 4.1.4, case 3. Therefore τ contains no cycles. A rooted graph with no cycles is a tree. □

Proposition 4.1.3. *The number $|\mathcal{T}(\mathbb{S})|$ of subtrees of $\mathbb{S}$ grows as $O(\exp(|\mathcal{E}|))$.*

Proof. See Appendix A.3.2. □

Proposition 4.1.4. *The scope of any subtree $\tau(X, Z) \in \mathcal{T}(\mathbb{S}(X, Z))$ is $\{X, Z\}$.*

Proof. A subtree is obtained with Definition 4.1.7 by iteratively choosing only one child of each sum node. However, each child of a sum node has the same scope, due to Definition 4.1.4, condition 6. Hence, taking only one child one obtains the same scope as taking all the children. □

$$P(X, Z) = \sum_{\tau \in \mathcal{T}(\mathbb{S}|Z)} \lambda_\tau P_\tau(X) \tag{4.1.2}$$

$$P(X) = \sum_{\tau \in \mathcal{T}(\mathbb{S})} \lambda_\tau P_\tau(X) \tag{4.1.3}$$

$$P_\tau(X) = \prod_{r \in V_\tau \,:\, vpa(r)=\emptyset} P_r(X_r) \prod_{s \in V_\tau, t : \in V_\tau \,:\, s \in vpa(t)} P_{s,t}(X_t|X_s) \tag{4.1.4}$$

$$\lambda_\tau = \prod_{s \in O_\tau} Q_s(k_{s,\tau}) \prod_{s \in U_\tau} W_s(k_{s,\tau}) \tag{4.1.5}$$

Table 4.1.1. - Probabilistic models related to a SPGM $\mathbb{S}(X, Z)$. Symbols V_τ, O_τ and U_τ denote respectively the set of Vnodes, Observed Sum Nodes and Unobserved Sum Nodes in a subtree $\tau \in \mathcal{T}(\mathbb{S})$. $k_{s,\tau}$ denotes the index of the child of s that is included in τ (see Definition 4.1.7). Evaluation of $\mathbb{S}$ (Definition 4.1.6) is equivalent to inference using the distribution (4.1.2), which is a mixture distribution (4.1.3) of tree graphical models (4.1.4), whose structure *depends on the context* as specified by the context variables Z of (4.1.2).

SPGMs as Mixtures of Subtrees

In this section, we show that SPGMs can be interpreted as mixtures of trees. Table 4.1.1 lists the notation and probabilistic (sub-)models relevant in this context.

As a first step, we show that inference in a subtree τ due to Definition 4.1.7 is equivalent to inference in a tree graphical model of the form (2.3.4), multiplied for a constant factor determined by the sum nodes in the subtree.

Proposition 4.1.5. *Let $\mathbb{S} = \mathbb{S}(X, Z)$ be a given SPGM, and let $\tau \in \mathcal{T}(\mathbb{S})$ be a subtree (Def. 4.1.7) with indicator variables z_τ (Def. 4.1.8). Then message passing in τ is equivalent to inference using the distribution*

$$\lambda_\tau P_\tau(X) \prod_{[Z_s]_j \in z_\tau} [Z_s]_j, \tag{4.1.6}$$

where $P_\tau(X)$ is a tree graphical model of the form (4.1.4), $\lambda_\tau > 0$ is a scalar term obtained by multiplying the weights of all sum nodes in τ given by (4.1.5), and $\prod_{[Z_s]_j \in z_\tau} [Z_s]_j$ is the product of all indicator variables in z_τ.

Proof. See Appendix A.3.1. □

The second step consists in noting that $\mathbb{S}$ can be written equivalently as the mixture of all its subtrees.

Proposition 4.1.6. *Evaluating a SPGM $\mathbb{S}(X, Z)$ is equivalent to evaluating a SPGM $\mathbb{S}'(X, Z) = \sum_{\tau \in \mathcal{T}(\mathbb{S})} \tau(X, Z)$.*

Proof. In Appendix A.3.3. □

We are now prepared to state the main result of this section.

Proposition 4.1.7. *Let $Y_x \subseteq X, Y_z \subseteq Z$ denote evidence variables with assignment $y_x \in \Delta(Y_x), y_z \in \Delta(Y_z)$, respectively, and denote by $x_{\setminus y} \in \Delta(X \setminus Y_x), z_{\setminus y} \in \Delta(Z \setminus Y_z)$ assignments to the remaining variables. Evaluating a SPGM $\mathbb{S} = \mathbb{S}(X, Z)$ with assignment (y_x, y_z) (Definitions 4.1.5 and 4.1.6) is equivalent to performing marginal inference with respect to the distribution* (4.1.2) *as follows:*

$$P(Y_x = y_x, Y_z = y_z) = \sum_{\substack{x_{\setminus y} \in \Delta(X \setminus Y_x) \\ z_{\setminus y} \in \Delta(Y \setminus Y_z)}} P\big((X \setminus Y_x) = x_{\setminus y}, (Y \setminus Y_z) = z_{\setminus y}, Y_x = y_x, Y_z = y_z\big). \tag{4.1.7}$$

Proof. Due to Propositions 4.1.5 and 4.1.6, the evaluation of $\mathbb{S}$ corresponds to performing message passing (Eq. 2.3.6) with the mixture distribution

$$\sum_{\tau \in \mathcal{T}(\mathbb{S})} \Big(\prod_{[Z_s]_j \in z_\tau} [Z_s]_j \Big) \lambda_\tau P_\tau(X).$$

We now note that the term $\big(\prod_{[Z_s]_j \in z_\tau} [Z_s]_j\big)$ attains the value 1 only for the subset of trees compatible with the assignment $Y_z = y_z$ and 0 otherwise (since some indicator in the product is 0), that is for subtrees in the set $\mathcal{T}(\mathbb{S}|Y_z = y_z)$ (Definition 4.1.9). Therefore, the sum can be rewritten as $\sum_{\tau \in \mathcal{T}(\mathbb{S}|Z)}$, and the indicator variables (with value 1) can be removed, which results in (4.1.2). The proof is concluded noting that computing message passing in a mixture of trees with assignment y_x, y_z corresponds to computing marginals $P(Y_x = y_x, Y_z = y_z)$ in the corresponding distribution (Section 2.3.2), hence Eq. (4.1.7) follows. □

Example 19. All subtrees of the SPGM $\mathbb{S}(X, Z)$ shown by Fig. 4.1.1a are shown as tree graphical models by Fig. 4.1.1c. The probabilistic model encoded by the SPGM is a mixture of these subtrees whose structure depends on the context variables Z.

The propositions above entail the crucial result that the probabilistic model of a SPGM is a mixture of trees where the mixture size grows *exponentially* with the SPGM size (Definition 4.1.7), but in which the inference cost grows only *polynomially* (Proposition 4.1.1). Hence, *very large* mixtures models can then be modelled *tractably*. This property is obtained by modelling trees $\tau \in \mathcal{T}(\mathbb{S}|Z)$ by combining sets of *shared subtrees*, selected through context variables Z, and by computing inference in shared parts only once (cf. the example in Fig. 4.1.1).

Conditional and Contextual Independence

In this section we discuss conditional and contextual independence semantics in SPGMs, based on their interpretation as mixture model.

Definition 4.1.10. [Context-dependent paths] Consider variables $A \in X, B \in X$ and a context $z \in \Delta(\overline{Z})$ with $\overline{Z} \subseteq Z$.

- The set $\pi(A, B)$ is the set of all directed paths in $\mathbb{S}$ going from a Vnode with label A to a Vnode with label B.
- The set $\pi(A, B|\overline{Z} = z) \subseteq \pi(A, B)$ is the subset of paths in $\pi(A, B)$ in which all the indicator variables over $\overline{Z}$ (Definition 4.1.8) are in state z.

Proposition 4.1.8. *[*D-separation *in SPGMs] Consider a SPGM $\mathbb{S}(X, Z)$, variables $A, B, C \in X$ and a context $z \in \Delta(\overline{Z})$ with $\overline{Z} \subseteq Z$. The following properties hold for the probabilistic model $\mathbb{S}$ corresponding to Eq.* (4.1.2)*:*

1. *A and B are* independent *iff $\pi(A, B) = \emptyset$ and $\pi(B, A) = \emptyset$ (there is no directed path from A to B).*
2. *A and B are* conditionally independent *given C if all directed paths $\pi(A, B)$ and $\pi(B, A)$ contain C.*
3. *A and B are* contextually independent *given context $\overline{Z} = z$ iff $\pi(A, B|\overline{Z} = z) = \emptyset$ and $\pi(B, A|\overline{Z} = z) = \emptyset$.*
4. *A and B are* contextually and conditionally independent *given C and context $\overline{Z} = z$ iff all paths $\pi(A, B|\overline{Z} = z)$ and $\pi(B, A|\overline{Z} = z)$ contain C.*

Example 20. In Fig. 4.1.1, A and D are conditionally independent given C; A and C are conditionally independent given B and context $Z_1 = 1$.

Proof. In a mixture of trees, conditional independence of A, B given C holds *iff* for every tree in the mixture the path between A and B contains C (D-separation, see [Cowell et al., 2003]). If D-separation holds for all paths in $\pi(B, A|z)$ then it holds for all the subtrees compatible with $Z = z$. But $P(X, Z)$ is the mixture of all subtrees compatible with assignment z (Eq. (4.1.2)). Hence the result follows. □

The proposition above provides SPGMs with a high level representation of both contextual and conditional independence. This is obtained by using different variables sets X and Z for the two different roles. The set X appears in Vnodes and entails conditional independences due to D-separation, with close similarity to tree graphical models (Proposition 4.1.8). The set Z enable contextual independence through the selection of tree branches via sum nodes.

Note also that using the set of paths π allows to infer conditional and contextual independences without need to check all the individual subtrees, whose number can be exponentially larger than the cardinality of π.

Related Models

SPGMs are closely related to hierarchical mixtures of trees (HMT) [Jordan, 1994] and *generalize* them. Like SPGMs, HMTs allow a compact representation of mixtures of trees by using a hierarchy of "choice nodes" where different trees are selected at each branch (as sum nodes do in SPGMs). While in HMTs the choice nodes separate the graph into disjoint branches and thus an overall tree structure is induced, however, SPGMs enable to use a DAG structure where parts of the graph towards the leaves can appear in children of multiple sum nodes.

The probabilistic model encoded by SPGMs also has a close connection to Gates [Minka and Winn, 2009] and the similarly structured Mixed Markov Models (MMM) [Fridman, 2003]. Gates enable contextual independence in graphical models by including the possibility of activating/deactivating factors based on the state of some context variables. Regarding SPGMs, the inclusion/exclusion of factors in subtrees $\mathcal{T}(\mathbb{S}|Z)$ depending on values of Z can be seen as a gating unit that enables a full set of tree factors to be active, which suggests to identify a SPGM as a Gates model. On the other hand, SPGMs restrict inference to a family of models in which inference is tractable by construction, while inference in Gates generally is intractable. In addition, SPGMs allow an interpretation as mixture models, which is not the case for Gates and MMMs.

Finally, we remark that SPGM subtrees closely parallel the concept of SPN subnetworks, first described in Gens and Domingos [2012] and then formalized in Zhao et al. [2016b]. However, while subnetworks in SPNs represent simple factorizations of the leaf distributions (which can be represented as graphical models without edges), subtrees in SPGMs represent tree graphical models (which include edges).

4.1.4. Interpretation as SPN

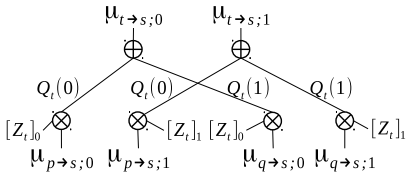
In this section we discuss SPGMs as a high level, fully general representation of SPNs as defined by Definition 3.1.3.

SPGMs encode SPNs

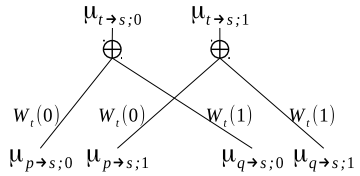
Proposition 4.1.9. *The message passing procedure in* $\mathbb{S}(X, Z)$ *encodes a SPN* $S(X, Z)$.

Proof. In Appendix A.3.4. □

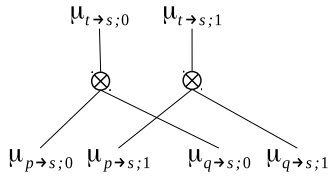
Note that in the encoded SPN, each SPGM message is represented by a set of sum nodes, which can be seen immediately from Fig. 4.1.2. Each sum node in the set represents the value of the message corresponding to a certain state of the output variable (namely, $\mu_{t\to s;j}$ for each j). This entails an increase in representation size (but not in inference cost) by a $|\Delta_{max}|^2$ factor. Note also that the role of nodes for implementing conditional and contextual independence is lost during the conversion to SPN, since both SPGM Vnode and SPGM sum node messages translate into a set of SPN sum nodes.



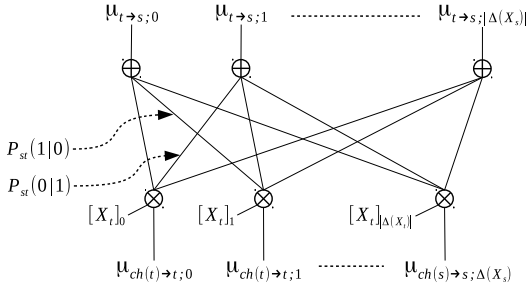
(a) Observed sum node (Eq. (4.1.1a)).



(b) Unobserved sum node (Eq. (4.1.1b)).



(c) Product node (Eq. (4.1.1c)).



(d) Vnode (Eq. (4.1.1d)).

Figure 4.1.2. - Representation of message passing equations as SPNs. For better visibility, sum and product nodes are assumed to have only two children p, q and with binary variables.

Proposition 4.1.10. *SPGMs are as* expressive *as SPNs, in the sense that if a distribution $P(X, Z)$ can be represented as a SPN with inference cost C, then it can also be represented as a SPGM with inference cost C and vice versa.*

Proof Sketch. Firstly, due to Proposition 4.1.9, SPNs are at least as expressive as SPGMs since they encode a SPN via message passing. Secondly, any SPN S can be transformed into an equivalent SPGM $\mathbb{S}$ by simply replacing the indicator variable $[A]_a$ in SPN leaves with Vnodes s associated to variable A and unary probability $P_s(A) = [A]_a$ (notice that pairwise probabilities do not appear). It is immediate to see that by doing so all the conditions of Definition 4.1.4 are satisfied, and evaluating $\mathbb{S}$ with message passing yields S. As a consequence, SPGMs are at least as expressive as SPNs. □

Discussion

Propositions 4.1.9 and 4.1.10, together with the connections to graphical models worked out above, enable an interpretation of a SPGM $\mathbb{S}$ as a high-level representation of the encoded SPN S. These generalizes what the introductory example demonstrated by comparing Fig. 1.1.4c with Fig. 1.1.4d.

The SPGM representation is more *compact* than SPNs because employing variable nodes as in graphical models enables to represent conditional independences through message passing. Passing from an SPGM to the SPN representation entails an increase in the model size due to the expansion of messages by a $N_{pa}|\Delta_{max}|^2$ factor (see Definition

4.1.5 and Fig. 4.1.2).[1]

The SPGM representation allows a high level representation of *conditional* independences through Vnodes and D-separation, and of *contextual* independences through the composition of subtrees due to context variables Z. In contrast, the roles of contextual and conditional independence in SPNs is hard to decipher (Fig. 1.1.4c) because there is no distinction between nodes created by messages sent from SPGM sum nodes (implementing contextual independence) and messages generated from SPGM Vnodes (implementing conditional independence), both of which are represented as a set of SPN sum and product nodes (see Fig. 4.1.2). In addition, there is no distinction between contextual variables Z and Vnode variables X.

Note that SPGMs are not more compact than SPNs in situations in which there are no conditional independences that can be expressed by Vnodes. However, we postulate that the co-occurrence of conditional and contextual independences creates relevant application scenarios (as shown in Section 3.4) and enables connections between SPNs and graphical models that can be exploited in future work.

Finally, the interpretation of SPGMs as SPN also allows to translate all methods and procedures available for SPNs to SPGMs. These include jointly computing the *marginals* of all variables by derivation [Darwiche, 2003], with time and memory linear cost in the number of edges in the SPN. In addition, Maximum a Posteriori queries can be computed simply by substituting the sums in Eqs. (4.1.1) by max operations. We leave the exploration of these aspects to future work, since they are not central for our present discussion.

4.2. Learning SPGMs

In this section, we exploit the relations between graphical models and SPNs embodied by SPGMs and present an algorithm for *learning the structure* of SPGMs.

4.2.1. Preliminaries

Structure learning denotes the problem of learning both the parameters of a probability distribution $P(X|\mathcal{G})$ and the structure of the underlying graph $\mathcal{G}$. As both the GM and the SPN represented by a given SPGM due to Sections 4.1.3 and 4.1.4 involve the same graph, the problem is well defined from both viewpoints.

Let $X = \{X_j\}_{j=1}^{M}$ be a set of M discrete variables. Consider a training set of N i.i.d samples $D = \{x^i\}_{i=1}^{N} \subset \Delta(X)$, used for learning. Formally, we aim to find the graph $\mathcal{G}^*$ governing the distribution $P(X)$ which maximizes the Log Likelihood

$$\mathcal{G}^*(X) = \arg\max_{\mathcal{G}} \mathrm{LL}(G) = \arg\max_{\mathcal{G}} \sum_{i=1}^{N} \ln P(x^i|\mathcal{G}) \tag{4.2.1}$$

[1] Note, however, that inference cost remains identical in $\mathbb{S}$ and S.

or the weighted Log Likelihood

$$\mathcal{G}^*(X) = \arg\max_{\mathcal{G}} \mathrm{WLL}(G, w) = \arg\max_{\mathcal{G}} \sum_{i=1}^{N} w_i \ln P(x^i|\mathcal{G}), \qquad w_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{4.2.2}$$

Learning Tree GMs. Learning the structure of GMs generally is NP-hard. For discrete tree GMs however the maximum likelihood solution T^* can be found with cost $O(M^2N)$ using the Chow-Liu algorithm (Chow and Liu [1968]).

Let $X_s, X_t \in X$ be discrete random variables ranging of assignments in the sets $\Delta(X_s), \Delta(X_t)$. Let $N_{s;j}$ and $N_{st;jk}$ respectively count the number of times X_s appears in state j and X_s, X_t appear jointly in state j, k in the training set D. Finally, define empirical probabilities $\overline{P}_s(j) = N_{s;j}/N$ and $\overline{P}_{st}(k|j) = N_{st;jk}/N_{s;j}$. The Chow-Liu algorithm comprises the following steps:

1. Compute the *mutual information* $\mathbb{I}_{st}$ between all variable pairs X_s, X_t,

$$\mathbb{I}_{st} = \sum_{j \in \Delta(X_s)} \sum_{k \in \Delta(X_t)} \overline{P}_{s,t}(j,k) \ln \frac{\overline{P}_{s,t}(j,k)}{\overline{P}_s(j)\overline{P}_t(k)}. \tag{4.2.3}$$

2. Create an undirected graph $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$ with adjacency matrix $\mathbb{I} = \{\mathbb{I}_{st}\}_{s,t \in \mathcal{V}}$ and compute the corresponding Maximum Spanning Tree $\overline{T}$.

3. Obtain the directed tree T^* by choosing an arbitrary node of $\overline{T}$ as root and using empirical probabilities $\overline{P}_s(X_s)$ and $\overline{P}_{st}(X_t|X_s)$ in place of corresponding terms in Eq. (2.3.3).

If the weighted Log Likelihood (4.2.2) is used as objective function, the algorithm remains the same. The only difference concerns the use of *weighted relative frequencies* for defining the empirical probabilities of (4.2.3): $\hat{N}_{s,j} = \frac{1}{\hat{N}_w} \sum_{i=1}^{N} \delta(x_s^i = j) w_i$ and $\hat{N}_{st,jk} = \frac{1}{\hat{N}_w} \sum_{i=1}^{N} \delta(x_s^i = j, x_t^i = k) w_i$, where $\hat{N}_w = \sum_{i=1}^{N} w_i$ and x_s^i denotes the state of variable X_s in sample x^i.

Learning Mixtures of Trees. We consider mixture models of the form

$$P(X) = \sum_{k=1}^{K} \lambda_k P_k(X|\theta^k),$$

with tree GMs $P_k(X|\theta^k)$, $k = 1, \ldots, K$, corresponding parameters $\{\theta^k\}_{k=1}^K$ and non-negative mixture coefficients $\{\lambda_k\}_{k=1}^K$ satisfying $\sum_{k=1}^K \lambda_k = 1$. While inference with mixture models is tractable as long as it is tractable with its individual mixture components, maximum likelihood generally is NP hard. A local optimum can be found with

Expectation-Maximization (EM) [Dempster et al., 1977], whose pseudocode we report in Algorithm 4.1 for convenience. The M-step (line 8) involves the weighted maximum likelihood problem and determines θ^k using the Chow-Liu algorithm described above.

It is well known that each EM iteration does not decrease the Log Likelihood, hence it approaches a local optimum.

Algorithm 4.1 EM for Mixture Models(P_k, λ_k, D)

Input: Initial model $P(X|\theta) = \sum_k \lambda_k P_k(X)$, training set D
Output: $P_k(X)$, λ_k locally maximizing $\sum_{i=1}^N \ln P(x^i|\theta)$

1: **repeat**
2: **for all** $k \in 1...K, i \in 1...N$ **do** // E-step
3: $\gamma_k(i) \leftarrow \frac{\lambda_k P_k(x_i)}{\sum_{k'} \lambda_{k'} P_{k'}(x_i)}$
4: $\Gamma_k \leftarrow \sum_{i=1}^N \gamma_k(i)$
5: $w_i^k \leftarrow \gamma_k(i)/\Gamma_k$
6: **for all** $k \in 1...K$ **do** // M-step
7: $\lambda_k \leftarrow \Gamma_k/N$
8: $\theta^k \leftarrow \arg\max_{\theta^k} \sum_{i=1}^N w_i^k \ln P_k(x^i|\theta^k)$
9: **until** convergence

Learning SPNs. Let $S(X)$ denote a SPN, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and graph with edge weights. Both *structure learning* (optimizing $\mathcal{G}$ and W) and *parameter learning* (optimizing W only) are NP-hard in SPNs [Darwiche, 2002]. Hence, only algorithms that seek a local optimum can be devised.

Parameter learning can be performed by directly applying the EM iteration for mixture models, while efficiently exploiting the interpretation of SPNs as a large mixture model with shared parts [Desana and Schnörr, 2016].

To describe EM for SPNs, which will be used in a later section, we need some additional notation. Consider a node $q \in \mathcal{V}$, and let S_q denote the sub-SPN having node q as root. If q is a Sum Node, then by Definition 3.1.3 a weight w_j^q is associated to each edge $(q, j) \in \mathcal{E}$. Note that evaluating $S(X = x)$ entails computing $S_q(X = x)$ for each node $q \in \mathcal{V}$ due to the recursive structure of SPNs. Hence $S(x)$ is function of $S_q(x)$. The derivative $\partial S\,(x)/\partial S_q(x)$ can be computed with a root-to-leaves pass requiring $O(|\mathcal{E}|)$ operations [Poon and Domingos, 2011].

With this notation, the EM algorithm for SPNs iterates the following steps:

E step. Compute for each Sum Node $q \in \mathcal{V}$ and each $j \in ch(q)$

$$\beta_j^q = w_j^q \sum_{n=1}^{N} S\,(x^n)^{-1} \frac{\partial S\,(x^n)}{\partial S_q} S_j\,(x_n)\,. \tag{4.2.4}$$

M step. Update weights for each Sum Node $q \in \mathcal{V}$ and each $j \in ch(q)$ by $w_j^q \leftarrow \beta_j^q / \sum_{(q,i)\in\mathcal{E}} \beta_i^q$, where $\leftarrow$ denotes assignment of a variable.

Since all the required quantities can be computed in $O(|\mathcal{E}|)$ operations, EM has a cost $O(|\mathcal{E}|)$ per iteration (the same as an SPN evaluation).

In some SPN applications, weights are shared among different edges (see e.g. Gens and Domingos [2012], Cheng et al. [2014] and Amer and Todorovic [2016]). Then the procedure still maximizes the likelihood locally. Let $\overline{V} \subseteq V$ be a subset of Sum Nodes with shared weights, in the sense that the set of weights $\{w_j^q\}_{j\in ch(q)}$ associated to incident edges $(q, j) \in \mathcal{E}$ is the same for each node $q \in \overline{V}$. The EM update of a shared weight w_j^q reads (cf. Desana and Schnörr [2016])

$$w_j^q \leftarrow \frac{\sum_{q\in\overline{V}} \beta_j^q}{\sum_i \sum_{q\in\overline{V}} \beta_i^q}, \qquad (q, j) \in \mathcal{E}. \tag{4.2.5}$$

Structure learning can be more conveniently done with SPNs than with graphical models, because tractability of inference is always guaranteed and hence not a limiting factor for learning the model's structure. Several greedy algorithms for structure learning were devised (see Section 5.2), which established SPNs as state of the art models for the estimation of probability distributions. We point out that most approaches employ a recursive procedure in which children of sum and Product Nodes are generated on *disjoint* subsets of the dataset, thus obtaining a *tree SPN*, while SPNs can be more generally defined on DAGs. Recently, [Rahman and Gogate, 2016b] discussed the limitations of using tree structured SPNs as opposed to DAGs, and addressed the problem of post-processing SPNs obtained with previous methods, by merging similar branches so as to obtain a DAG.

Our method proposed in Section 4.2.3 is the first one that directly estimates a DAG-structured SPN.

4.2.2. Parameter Learning in SPGMs

Parameters learning in a SPGM $\mathbb{S}$ can be done by interpreting $\mathbb{S}$ as a SPN encoded by message passing (Proposition 4.1.10) and directly using any available SPN parameter learning method (these include EM seen in Section 4.2.1 and others [Gens and Domingos, 2012]). Hence, we do not discuss this aspect further.

Note however that Sum Node messages (Definition 4.1.5) require weight sharing between the SPN Sum Nodes. EM for SPNs with weight tying is addressed in Section 4.2.1.

4.2.3. Structure Learning in SPGMs

Structure learning is an important aspect of tractable inference models, thus it is crucial to provide a structure learning for SPGMs. Furthermore, it is useful to provide a first

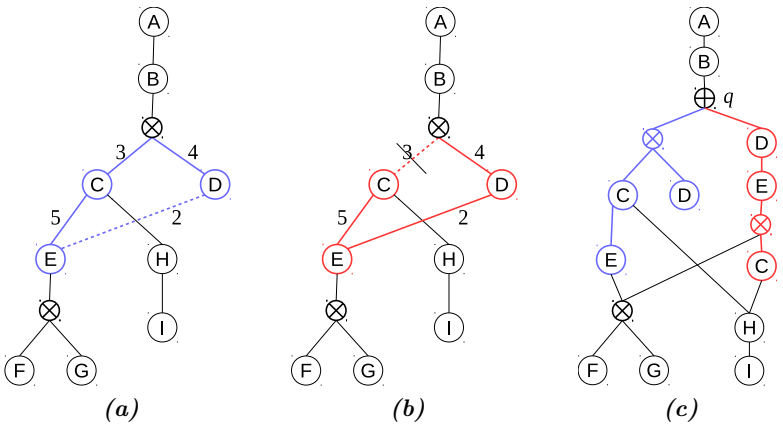


Figure 4.2.1. - Graphical representation of edge insertion. For simplicity, we start with a SPGM representing a single MST. Fig. 4.2.1a: Inserting (D, E) creates a cycle (blue). Fig. 4.2.1b: Removing the minimum edge in the cycle (except (D, E)) gives the MST containing (D, E). Fig. 4.2.1c: The red MST is inserted into $\mathbb{S}$ sharing the common parts.

example of how the new *connections* between GMs and SPNs can be exploited in practice.

We propose a structure learning algorithm based on the Chow-Liu algorithm for trees (Section 2.3.4). We start observing that edges with large Mutual Information can be excluded from the Chow-Liu tree, thus losing relevant correlations between variables (Fig. 1.3.1b, left). An approach to address this problem, inspired by [Meila and Jordan, 2000], is to use a mixture of spanning trees such that the k-best edges are included in at least one tree. We anticipate that the trees obtained in this way *share a large part* of their structure (Fig. 1.3.1c), hence the mixture can be implemented efficiently as a SPGM.

Algorithm Description. We describe next *LearnSPGM*, a procedure to learn structure and parameters of a SPGMs which locally maximizes the weighted Log Likelihood (4.2.2). We use the notation of Section 4.2.1.

The algorithm learns a SPGM $\mathbb{S}$ in three main steps (pseudocode in Algorithm 4.2). First, $\mathbb{S}$ is initialized to encode the Chow-Liu tree T^* (Fig. 4.2.1a) – that is, $\mathcal{T}(\mathbb{S})$ includes a single subtree (Definition 4.1.7) $\tau*$ corresponding to T^*. Then, we order each edge $(s, t) \in \overline{\mathcal{E}}$ which was not included in T^* by decreasing mutual information $\mathbb{I}_{st}$, collecting them in the ordered set Q. Finally, we insert each edge $(s, t) \in Q$ in $\mathbb{S}$ with the sub-procedure *InsertEdge* described below, until Log Likelihood convergence or a given maximum size of $\mathbb{S}$ is reached.

InsertEdge$(\mathbb{S}, T^*, (s, t))$ comprises three steps:

1. Compute the maximum spanning tree over $\overline{\mathcal{G}}$ which includes (s, t), denoted as T_{st}. Finding T_{st} can be done efficiently by first inserting edge (s, t) in T^*, which creates a cycle $\mathcal{C}$ (Fig. 4.2.1a), then removing the minimum edge in $\mathcal{C}$ *except* (s, t) (Fig. 4.2.1b). The potentials in T_{st} are set as empirical probabilities $\overline{P}_{st}$ according to

the Chow-Liu algorithm. Notice that trees T^* and T_{st} have identical structure up to $\mathcal{C}$ and can then be written as $T^* = T^1 \cup \mathcal{C}' \cup T^2$ and $T_{st} = T^1 \cup \mathcal{C}'' \cup T^2$, where $\mathcal{C}' = T^* \cap \mathcal{C}, \mathcal{C}'' = T_{st} \cap \mathcal{C}$.

2. Add T_{st} to the set $\mathcal{T}(\mathbb{S})$ by sharing the structure in common with T^* (T^1 and T^2 above). To do this, first identify the edge (s,t) s.t. $s \in T^1$ and $t \in \mathcal{C}'$ (e.g. (B,C) in Fig. 4.2.1b). Then, create a non-Observed Sum Node q, placing it between s and t, unless such node is already present due to previous iterations (see q in Fig. 4.2.1c).

   At this point, one of the child branches of q contains $\mathcal{C}' \cup T^2$. We now add a new child branch containing $\mathcal{C}''$ (Fig. 4.2.1c). Finally, we connect nodes in $\mathcal{C}''$ to their descendants in the shared section T^2. The insertion maintains $\mathbb{S}$ valid since the X and Z-scope of any node in $\mathbb{S}$ does not change. Furthermore, inserting $\mathcal{C}''$ in this way we add a subtree representing T_{st} in $\mathcal{T}(\mathbb{S})$, selected by choosing the child of s corresponding to $\mathcal{C}''$.

3. Update the weights of incoming edges of the Sum Node q by using Eq. 4.2.5 on the set of SPN nodes generated by q with Eq.(4.1.1a) during the conversion to SPN (see Proposition 4.1.9).

Convergence It is possible to prove that the Log Likelihood *does not decrease* at each insertion, and thus the initial Chow-Liu tree provides a *lower bound* for the Log Likelihood.

Proposition 4.2.1. *Each application of InsertEdge does not decrease the Log Likelihood of the SPGM (Eq.* (4.2.2)*).*

Proof. InsertEdge adds the branch $\mathcal{C}' \cup T^2$ to Sum Node q, hence it adds a new incident edge and a corresponding weight to q. We now note that computing weight values using Eq. (4.2.5) (step 3 in *InsertEdge* above) allows to find the optimal weights of edges incoming to q considering the other edges fixed, as shown e.g. in Desana and Schnörr [2016].[2] Since the new locally optimal solution includes the weight configuration of the previous iteration, which is simply obtained by setting the new edge weight to 0 and keeping the remaining weights, the Log Likelihood does not increase at each iteration.

Proposition 4.2.2. *The Chow Liu tree Log Likelihood is a lower bound for the Log Likelihood of a SPGM obtained with LearnSPGM.*

Proof. Follows immediately from Proposition 4.2.1, noting that the SPGM is initialized as the Chow Liu tree T^*.

[2] However, this is not the globally optimal solution when the other parameters are free.

Complexity. Steps 1 and 2 of *InsertEdge* are inexpensive, only requiring a number of operations linear in the number of edges of the Chow Liu tree T^*. The per iteration complexity of *InsertEdge* is dominated by step 3, in which the computation of weights through Eq. 4.2.5 requires evaluating the SPGM for the whole dataset. Although evaluation of a SPGM is efficient (see Proposition 4.1.1), this can still be too costly for large datasets. We found empirically that assigning weights proportionally to the mutual information of the inserted edge provides a reliable empirical alternative, which we use in experiments.

4.2.4. Learning Mixtures of SPGMs

LearnSPGM is apt at representing data belonging to a single cluster, since (similarly to Chow-Liu trees) the edge weights are computed from a single mutual information matrix estimated on the whole dataset. To model densities with natural clusters one can use mixtures of SPGMs trained with EM. We write a mixture of SPGMs in the form $\sum_{k=1}^{K} \lambda_k P_k(X|\theta^k)$, where each term $P_k(X|\theta^k)$ is the probability distribution encoded by a SPGM $\mathbb{S}_k$ (Eq. 4.1.3) governed by parameters $\theta^k = \{\mathcal{G}^k, \{P^k_{st}\}, \{W^k_s\}, \{Q^k_s\}\}$ (Definition 4.1.4).

EM can be adopted for any mixture model as long as the weighted maximum Log Likelihood in the M-step can be solved (alg. 4.1 line 8). In addition, Neal and Hinton [1998] show that EM converges as long as the M-step can be at least partially performed, namely if it possible to find parameters $\overline{\theta}_k$ such that (see Eq. 4.2.2)

$$\mathrm{WLL}(P_k(X|\overline{\theta}_k), w^k) \geq \mathrm{WLL}(P_k(X|\theta^k), w^k). \tag{4.2.6}$$

These observation suggest to use *LearnSPGM* to approximately solve the weighted maximum likelihood problem. However, while *LearnSPGM* ensures that the Chow-Liu tree lower bound always increases, the actual weighted Log Likelihood can decrease. To satisfy Eq. (4.2.6), we employ the simple shortcut of rejecting updates of component $\mathbb{S}_k$ when Eq. (4.2.6) is not satisfied for θ^k_{new} (Algorithm 4.3 line 9). Doing this, the following holds:

Proposition 4.2.3. *The Log Likelihood of the training set does not decrease at each iteration of EM for Mixtures of SPGMs (Alg. 4.3).*

Algorithm 4.2 *LearnSPGM* $(D = \{x^i\}, \{w^i\})$

Input: samples D, optional sample weights w
Output: SPGM $\mathbb{S}$ approx. maximizing $\sum_{i=1}^{N} w_i \ln P(x^i|\theta)$

$\mathbb{I} \leftarrow$ Mutual Information of D with weights w
$T^* \leftarrow$ Chow-Liu tree with connection matrix $\mathbb{I}$
$\mathbb{S} \leftarrow$SPGM representing T^*
$Q \leftarrow$ queue of edges $(s,t) \notin \mathcal{E}$ ordered by decreasing $\mathbb{I}_{st}$
repeat
    InsertEdge $(\mathbb{S}, T^*, Q.pop())$
    $AssignWeights$ to the modified Sum Node
until convergence **or** max size reached

Algorithm 4.3 EM for Mixtures of SPGMs$(\mathbb{S}_k, \lambda_k, D)$

Input: Initial model $P(X) = \sum_k \lambda_k \mathbb{S}_k(X)$, samples D
Output: Updated $\mathbb{S}_k(X)$, λ_k locally maximizing $\sum_{i=1}^{N} w_i \ln P(x^i|\theta)$

repeat
    for all $k \in 1...K, i \in 1...N$ **do** // E-step
        $\gamma_k(i) \leftarrow \frac{\lambda_k \mathbb{S}_k(x_i)}{\sum_{k'} \lambda_{k'} \mathbb{S}_{k'}(x_i)}$
        $\Gamma_k \leftarrow \sum_{i=1}^{N} \gamma_k(i)$
        $w_i^k \leftarrow \gamma_k(i)/\Gamma_k$
    for all $k \in 1...K$ **do** // M-step
        $\lambda_k \leftarrow \Gamma_k/N$
        $\overline{\mathbb{S}}_k \leftarrow LearnSPGM(D, w^k)$
        if WLL$(\overline{\mathbb{S}}_k, w^k) \geq$ WLL$(\mathbb{S}_k, w^k)$ **then**
            $\mathbb{S}_k \leftarrow \overline{\mathbb{S}}_k$
until convergence

5. Applications

The introduction of a new architecture and of new procedures, as was done in the previous chapters, creates the need to provide example applications and to empirically justify the need for the new representation.

This chapter presents several practical case studies that illustrate the techniques and architectures discussed in the previous sections of the thesis, with particular focus on SPGMs. The aim of these examples is to show various settings in which it is convenient to use jointly SPNs and graphical models, and to provide guidelines for applying these models in practical situations. We discuss the following applications:

- In Section 5.1, we apply Expectation-Maximization for SPNs with structured leaves (derived in Section 3.3.2) to a density estimation experiment. We perform a comparison between state-of-the-art parameter learning algorithms (which learn only the weights of internal SPN edges) and our EM derivation which allows to learn jointly edge parameters and the structure of the leaf models, using 20 real world datasets for density estimation. The results show competitive performances of our approach, despite using smaller models - suggesting that much of the SPN complexity can be condensed in learning structured leaf models.

- In Section 5.2, we apply the novel structure learning algorithm derived in Section 4.2 to a density estimation setting. We comparing our method with 7 other state-of-the-art density estimation models on 20 real world datasets. Results show that our approach better than all other models in 6 cases over 20 and has close performances otherwise. In addition, since these results are obtained using a novel approach and despite comparing against methods based on years of stratified research, this application is a promising direction of future research.

- In Section 5.3, we describe a joint application of GMs and SPNs. We build a SPN whose leaves are made by a graphical model for the segmentation of *healthy* human retina images presented in Rathke et al. [2014]. We can extend Rathke et al. [2014] to the to the case of *pathological* retina images, by using a SPN where leaves corresponds to graphical models with different priors, adapted to fit well *local* pathological deformations of the image. A *globally* optimal MAP inference procedure allows to select the best combination of leaves to produce the final segmentation, under mild assumptions. Empirical results show that this approach reaches state-of-the-art performances on real world datasets for retina segmentation.

- In Section 5.4, we present a preliminary experiment regarding the use of SPGMs for approximating an *intractable* graphical model $\mathcal{G}$. To this aim, we follow an intuitive idea and quantitatively evaluate if it works in practice. We do so by creating a SPGM $\mathbb{S}$ modeling a very large mixture of spanning trees taken from $\mathcal{G}$, with up to 8^{10} trees, and learning the parameters of $\mathbb{S}$ on data sampled from G. Empirical results indicate that the learned SPGM generalizes better than other state-of-the-art models, denoting potential for this approach in future work.

- In Section 5.5 we employ SPGMs to model a very rge mixture of Quadtrees whose structure can be selected dynamically with MAP inference(Crouse et al. [1998]). Starting from the observation that Quadtrees are tree graphical models, and that SPGMs are good at modeling large mixtures of trees, we discuss how a particular form of factors allows SPGMs to represent a very large mixture of Quadtrees with dynamic structure, alleviating the well known blocky effect that constitutes the major limitation of Quadtrees. We demonstrate our findings with a preliminary experiment on denoising pixel level labels. Quantitative results show potential of this approach in future developments.

5.1. Learning SPNs with Tree Graphical Model Leaves - Benchmark Evaluation

Parameter learning algorithms for SPNs typically focus on learning edge weights, but do not allow to learn the parameters governing the distribution at the leaves (see e.g. Poon and Domingos [2011], Zhao et al. [2016b] and Zhao et al. [2016a]). To address this problem, we provided in Section 3.3.2 a derivation of Expectation-Maximization (EM) for SPNs that allows to learn the parameters governing leaf distributions with complex structure. The aim of this section is to evaluate the possible gains of training *jointly* the edge weights and the structure of SPN leaf distributions with EM, as opposed to just training edge weights and using SPNs with simple indicator variables.

In order to provide a quantitative evaluation, we perform the following two experiments, using a set of 20 binary datasets that have been widely used as a density estimation benchmark for SPNs and related architectures, and whose structure is described in Table 5.1.1 (see Gens and Domingos [2013]).

Setup To showcase the connection with graphical models, the models we use for SPN leaf distributions are Tree Graphical Models, in which weighted maximum likelihood can be solved exactly (Section 3.3.4).

First, we need to define the structure of the SPN on which we want to test the parameter learning performances. In order to keep the focus on parameter learning rather than structure learning we chose to use the simplest structure learning algorithm (LearnSPN, Section 3.4.2), and augment it to use tree GM leaves by simply adding a fixed number of

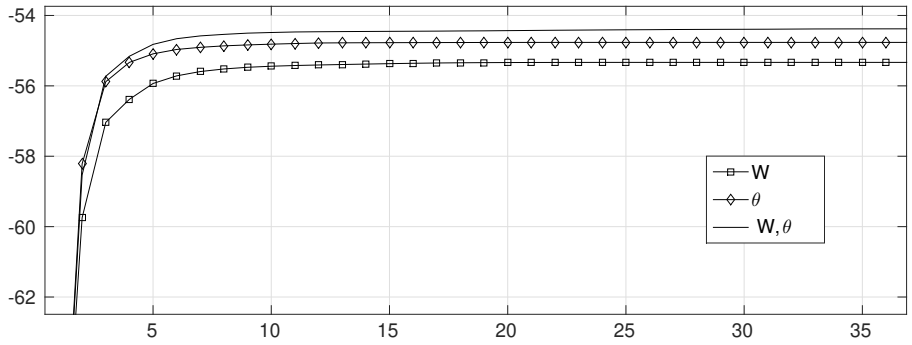


Figure 5.1.1. **-** Plot of training set Log Likelihood over iterations during EM training on the Jester dataset. Notice that the LL converges monotonically.

tree leaves to each generated sum node q. The tree leaves are initialized as a mixture of k trees learned over the data that was used to learn the subnetwork rooted in q (see Section 3.4.2 for details). To keep the models small and the structure simple, we limit the depth to a maximum integer value d. The number of trees k and the maximum depth d are hyperparameters of the algorithm. The algorithms in this section have been implemented in MATLAB.

Experiment 1**: Single Vs. Joint Parameters Training** In this experiment we compare learning using EM updates on edge weights W only and using updates on both weights and leaf nodes parameters θ (determining the *structure* and potentials of the tree leaves, see Section 3.3.2). The SPN structure is first learned from data with the modified LearnSPN algorithm described in the Setup section above. Then, parameters W and θ are randomly reinitialized, and EM on the training data is run until convergence of the training Log Likelihood.

In Table 5.1.1 we report training Log Likelihood when either EM updates on weights W only, on parameters θ only, or jointly on W and θ are used. Note: when training only W we also update the weights that define the tree potentials in each leaf, since they would be SPN weights if the tree leaves were expressed as SPNs. An example convergence plot of the three EM variants is shown in Fig. 5.1.1. The empirical results show that indeed EM updates for leaf distributions give better results than updates over the weight update only. While this is not surprising, since learning only the weights and not the tree structure leads to a strictly worse increase in Log Likelihood at each leaf update, the results indicate that learning the leaf structure during parameter training is a better way of investing computational resources than having a large number of edges and just tune the edge weights.

Experiment 2**: Comparison with State of the Art Parameter Learning Methods** In this experiment we test the generalization capabilities of learning tree leaves with EM

Table 5.1.1. - Dataset structure and training set Log Likelihood for Experiment 1 in Section 5.1. Columns W, θ and (W, θ) denote using EM on the respective parameters. Higher results (in bold) denote better density estimation performances.

Dataset	Nvars	|train|	|valid|	W	θ	W, θ
NLTCS	16	16181	2157	−6.16	−6.18	**−6.14**
MSNBC	17	291326	38843	−6.44	**−6.41**	**−6.41**
KDDCup2K	64	180092	19907	−2.41	−2.40	**−2.35**
Plants	69	17412	2321	−17.62	−14.01	**−13.78**
Audio	100	15000	2000	−42.34	−41.43	**−41.21**
Jester	100	9000	1000	−55.33	−54.77	**−54.38**
Netflix	100	15000	2000	−60.04	−58.47	**−58.33**
Accidents	111	12758	1700	−42.87	−32.37	**−31.87**
Retail	135	22041	2938	−10.76	**−10.67**	−10.69
Pumsb-star	163	12262	1635	−41.05	−26.69	**−26.45**
DNA	180	1600	400	−81.38	−80.68	**−79.95**
Kosarek	190	33375	4450	−11.12	−10.75	**−10.66**
MSWeb	294	29441	32750	−10.29	−9.70	**−9.69**
Book	500	8700	1159	−34.76	−34.68	**−33.67**
EachMovie	500	4524	1002	−59.05	−55.33	**−53.43**
WebKB	839	2803	558	−155.89	−148.97	**−147.90**
Reuters-52	889	6532	1028	−106.17	−92.93	**−92.61**
20Newsgrp.	910	11293	3764	−145.03	−143.89	**−143.22**
BBC	1058	1670	225	−261.69	−245.48	**−244.95**
Ad	1556	2461	327	−52.78	**−12.89**	−16.29

by comparing the test set Log Likelihood on the benchmark datasets against two state-of-the-art parameter learning methods. The SPN structure is learned used the simple structure learning algorithm described in the Setup section above. We perform a grid search for selecting the hyperparameters of the algorithm: the independence threshold used in Gens and Domingos [2013], with values $\lambda \in \{0.1, 0.01, 0.001\}$ (Section 3.4.2); the number of tree leaves attached to sum nodes, with values $k \in \{5, 20, 30\}$, and depth $d = \{2, 4, 6, \}$ (see Setup). Then, we fine tuned the initial SPN by training W, θ with EM, stopping as soon as validation Log Likelihood started to decrease.

Results. We compare against two state-of-the-art parameter learning methods: *Concave-Convex Procedure* (CCCP, Zhao et al. [2016b]) and *Collapsed Variational Inference* (CVI, Zhao et al. [2016a]) which employ LearnSPN for structure learning (like us, but without depth limit) then re-learn the edge parameters of the resulting SPN. The results of this experiment are shown in Table 5.1.2. To perform a fair comparison, we also plot the network size as the number of edges in the network (Table 5.1.2), and for each tree leaf node we also add to this count the number of edges which would be needed to represent the tree as a SPN. Our algorithm (column TreeSPN) *outperforms* CCCP and CVI in the majority of cases, despite the network size being *much smaller* (total number of edges is 5.41M vs. $27.1M$). These results indicates that it can be convenient to use computational resources for modelling SPNs with complex structured leaves, learned with EM, rather than in just increasing the number of SPN edges. This new aspect should be explored in future work.

Table 5.1.2. - Experimental results on Learning SPNs with EM. In the bottom row, we report the number of times one algorithm outperforms the others. Note that TreeSPN performs better (11 wins) than both CCCP (9 wins) and CVI (1 win) while using much smaller SPNs ($5.4M$ vs. $27M$ total edges).

	Test LL			**#edges**	
Dataset	TreeSPN	CCCP	CVI	TreeSPN	CCCP
NLTCS	**−6.01**	−6.03	−6.08	**2K**	$14K$
MSNBC	**−6.04**	−6.05	−6.29	**13K**	$55K$
KDDCup2K	−2.14	**−2.13**	−2.14	$50K$	**48K**
Plants	**−12.30**	−12.87	−12.86	**60K**	$133K$
Audio	**−39.76**	−40.02	−40.6	**93K**	$740K$
Jester	**−52.59**	−52.88	−53.84	**93K**	$314K$
Netflix	**−56.12**	−56.78	−57.96	**94K**	$162K$
Accidents	−29.86	**−27.70**	−29.55	**100K**	$205K$
Retail	−10.95	−10.92	**−10.91**	$116K$	**57K**
Pumsb-star	**−23.71**	−24.23	−25.93	**105K**	$140K$
DNA	**−79.90**	−84.92	−86.73	$167K$	**108K**
Kosarek	**−10.75**	−10.88	−10.70	**149K**	$203K$
MSWeb	−10.03	**−9.97**	−9.89	$186K$	**69K**
Book	**−34.68**	−35.01	−34.44	$434K$	**191K**
EachMovie	−55.42	**−52.56**	−52.63	**339K**	$523K$
WebKB	−167.8	**−157.5**	−161.5	**713K**	$1.44M$
Reuters-52	−91.69	**−84.63**	−85.45	**604K**	$2.21M$
20Newsgrp.	−156.8	**−153.2**	−155.6	**848K**	$14.6M$
BBC	−266.3	**−248.6**	−251.2	**881K**	$1.88M$
Ad	**−16.88**	−27.20	−19.00	**364K**	$4.13M$
#Wins/TotSize	**11**	8	1	**5.41**M	$27.1M$

5.2. Evaluating LearnSPGM on Benchmark Datasets for Density Estimation

This section aims at evaluating the performances of our structure learning algorithm for SPGMs (LearnSPGM, cf. Section 4.2 for details) on real world datasets.

We evaluate LearnSPGM on 20 real world datasets for density estimation described in Poon and Domingos [2011]. The number of variables ranges from 16 to 1556 and the number of training examples from $1.6K$ to $291K$ (Table 5.1.1). All variables are binary. We compare against several well cited state-of-the-art methods for density estimation, referred to with the following abbreviations: MCNets (Mixtures of CutsetNets, Rahman et al. [2014]); ECNet (Ensembles of CutsetNets, Rahman and Gogate [2016a]); MergeSPN Rahman and Gogate [2016b]; ID-SPN Rooshenas and Lowd [2014]; SPN Gens and Domingos [2013]; ACMN Lowd and Domingos [2012], MT (Mixtures of Trees Meila and Jordan [2000]); LTM (Latent Tree Models, Choi et al. [2011]). These methods report the current best results on the considered datasets.

Methodology. We found empirically that the best results were obtained using a two phase procedure: first we run EM updates with LearnSPGM on both structure and parameters until validation log-likelihood convergence, then we fine-tune using EM for SPNs on parameters only until convergence. We fix the following hyperparameters by grid search

Table 5.2.1. - Test set Log Likelihood comparison for the experiment in Section 5.2. In the bottom row, we report the number of times one algorithm performs better or on par with the others.

Dataset	SPGM	ECNet	MergeSPN	MCNet	ID-SPN	ACMN	SPN	MT	LTM
NLTCS	**-5.99**	-6.00	-6.00	-6.00	-6.02	-6.00	-6.11	-6.01	-6.49
MSNBC	**-6.03**	-6.05	-6.10	-6.04	-6.04	-6.04	-6.11	-6.07	-6.52
KDDCup2K	-2.13	-2.13	**-2.12**	**-2.12**	-2.13	-2.17	-2.18	-2.13	-2.18
Plants	-12.71	-12.19	**-12.03**	-12.74	-12.54	-12.80	-12.98	-12.95	-16.39
Audio	-39.90	-39.67	**-39.49**	-39.73	-39.79	-40.32	-40.50	-40.08	-41.90
Jester	-52.83	**-52.44**	-52.47	-52.57	-52.86	-53.31	-53.48	-53.08	-55.17
Netflix	-56.42	-56.13	**-55.84**	-56.32	-56.36	-57.22	-57.33	-56.74	-58.53
Accidents	**-26.89**	-29.25	-29.32	-29.96	-26.98	-27.11	-30.04	-29.63	-33.05
Retail	-10.83	**-10.78**	-10.82	-10.82	-10.85	-10.88	-11.04	-10.83	-10.92
Pumsb-star	**-22.15**	-23.34	-23.67	-24.18	-22.40	-23.55	-24.78	-23.71	-31.32
DNA	**-79.88**	-80.66	-80.89	-85.82	-81.21	-80.03	-82.52	-85.14	-87.60
Kosarek	-10.57	**-10.54**	-10.55	-10.58	-10.60	-10.84	-10.99	-10.62	-10.87
MSWeb	-9.81	**-9.70**	-9.76	-9.79	-9.73	-9.77	-10.25	-9.85	-10.21
Book	-34.18	**-33.78**	-34.25	-33.96	-34.14	-36.56	-35.89	-34.63	-34.22
EachMovie	-54.08	-51.14	**-50.72**	-51.39	-51.51	-55.80	-52.49	-54.60	†
WebKB	-154.55	-150.10	**-150.04**	-153.22	-151.84	-159.13	-158.20	-156.86	-156.84
Reuters-52	-85.24	-82.19	**-80.66**	-86.11	-83.35	-90.23	-85.07	-85.90	-91.23
20Newsgrp.	-153.69	-151.75	**-150.80**	-151.29	-151.47	-161.13	-155.93	-154.24	-156.77
BBC	-255.22	-236.82	**-233.26**	-250.58	-248.93	-257.10	-250.69	-261.84	-255.76
Ad	**-14.30**	-14.36	-14.34	-16.68	-19.00	-16.53	-19.73	-16.02	†
#Wins	6	5	9	1	0	0	0	0	0

on validation set: maximum number of edge insertions $\{10, 20, 60, 120, 400, 1000, 5000\}$, mixture size $\{5, 8, 10, 20, 100, 200, 400\}$, uniform prior $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-9}\}$ (used for mutual information and sum weights). LearnSPGM was implemented in C++ and is available at the following URL: `https://github.com/ocarinamat/SumProductGraphMod`. The average learning time per dataset is 42 minutes on an Intel Core i5-4570 CPU with 16 GB RAM. Inference takes up to one minute on the largest dataset.

Results. Test set log-Likelihood results are shown in Table 5.2.1. Our methods performs best between all compared models in 6 *datasets over* 20 (for comparison, ECNets are best in 5 cases, MergeSPN in 9, MCnets in 1). Notice that in doing so we obtain the current *best available* results on these 6 datasets.

Interestingly, LearnSPGM compares well against a well established literature despite being a radically *novel* approach, since ECNet, MergeSPN, MCNet, ID-SPN, ACMN, SPN all create a search tree by finding independences in data recursively (see Gens and Domingos [2013]). LearnSPGM is simpler than methods with similar performances: for instance ECnets use boosting and bagging procedures for ensemble learning, evolving over CNets that use EM, and MergeSPN post-processes the SPN in Gens and Domingos [2013]. Our model can be improved by including these techniques, such as using ensemble learning as in ECNets rather than EM as in MCnets. Finally, notice that SPGMs - that are large mixtures of trees - *always* outperform standard mixture of trees. This confirms that sharing tree structure *helps preventing overfitting*, which is critical in these models.

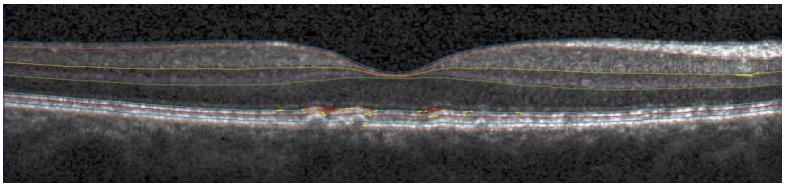


Figure 5.3.1. - Segmentation of a non pathological human retina in horizontal layers, obtained with the baseline graphical model discussed in Rathke et al. [2014].

5.3. SPNs with GM leaves for Locally Adaptive Priors in Human Retina Images

In this section we describe a joint application of SPNs and Graphical Models, which illustrates the potential of using the two families of models together. We create a SPN that employs Graphical Models for the segmentation of medical images of the human retina (Rathke et al. [2014]) as leaf distribution. By using this SPN, we can model a large set of graphical models adopted to fit local regions, whose selection depending on the context can be optimally performed through a MAP inference procedure. Empirical results show that taking the MAP estimates obtained with this approach reaches state-of-the-art performances on real world datasets for retina segmentation.

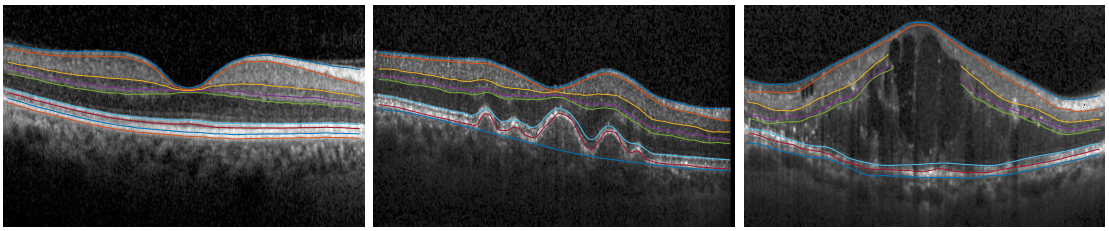
5.3.1. Setup

In the field of medical imaging, segmenting retinal tissue deformed by pathologies is a crucial but challenging task. Segmentation approaches are often constructed with a certain pathology in mind and may require a large set of labeled pathological scans, and therefore are tailored to that particular pathology.

We present an approach that can be easily transfered to new pathologies, as it is designed with no particular pathology in mind and requires no pathological ground truth. The approach is based on a graphical model presented in Rathke et al. [2014] trained for healthy scans, which is modified *locally* by adding pathology-specific shape modifications. We use the framework of SPNs to enable local modification of the model and to find the best combination of modified and unmodified local models that *globally* yield the best segmentation. The approach further allows to localize and quantify the pathology. We demonstrate the flexibility and the robustness of our approach, by presenting results for three different pathologies: diabetic macular edema (DME), age-related macular degeneration (AMD) and non-proliferative diabetic retinopathy.

5.3.2. Baseline Probabilistic Graphical Model

The model that we develop upon is a probabilistic model of segmentation of images of the human retina, discussed in Rathke et al. [2014] and summarized here. They model an OCT scan $y \in \mathbb{R}^{M \times N}$ (M rows and N A-Scans) and its segmentations b and



(a) Mild Diabetic RP *(b)* Intermediate AMD *(c)* Advanced DME

Figure 5.3.2. - Three different pathologies of different difficulty segmented by our SPN model. This is solely done by adding new shape information to a graphical model for healthy scans.

c respectively. Here $c \in \mathbb{N}^{K \cdot N}$ (K boundaries) denotes the discretized version of the continuous boundary vector $b \in \mathbb{R}^{K \cdot N}$, which is the connection between the discrete pixel domain of y and the continuous boundary domain of b. The graphical model is given by

$$p(y, c, b) = p(y|c)p(c|b)p(b). \tag{5.3.1}$$

We will briefly discuss each component, and refer to Rathke et al. [2014] for more details. **Appearance $p(y|c)$.** Appearance of boundaries and layers is modeled via *local* class-specific Gaussian densities: The probability of pixel $y_{i,j}$ belonging to a class $x_{i,j} \in \{l_1, \ldots, l_n, t_1, \ldots, t_{n-1}\}$ (see Fig. 5.3.6 (a)) is modeled as Gaussian,

$$p(y|c) = \prod_{i=1}^{M} \prod_{j=1}^{N} p(y_{i,j}|c), \qquad p(y_{i,j}|c) = \mathcal{N}(\tilde{y}_{i,j}|\mu_{x_{i,j}}, \Sigma_{x_{i,j}}), \tag{5.3.2}$$

where the class-label $x_{i,j}$ is determined by the boundary configuration c and $\tilde{y}_{i,j}$ is a patch of size around pixel $y_{i,j}$.

Shape $p(b)$. The *global* shape prior captures typical variations of cell layer boundaries. The shape vector b is determined by a linear Gaussian model

$$b = Ws + \mu + \epsilon, \quad s \sim \mathcal{N}(0, I), \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{5.3.3}$$

The matrix $W \in \mathbb{R}^{K \cdot N \times m}$ maps the low-dimensional vector $s \in \mathbb{R}^m$ onto b. Each column of W denotes a certain shape variation that gets added to the mean shape μ. Given n training segmentations $X \in \mathrm{R}^{n \times N \cdot K}$, W is obtained by the first m eigenvectors of $\mathrm{cov}(X)$ weighted by the corresponding eigenvectors, and μ simply is $\overline{X}$. The marginal distribution of b can then be shown to be

$$p(b) = \mathcal{N}(b; \mu, \Sigma = WW^T + \sigma^2 I). \tag{5.3.4}$$

MRF Regularization $p(c|b)$. Shape and appearance are combined via a Markov random field over the discrete variable c. It is composed of column-wise chain models

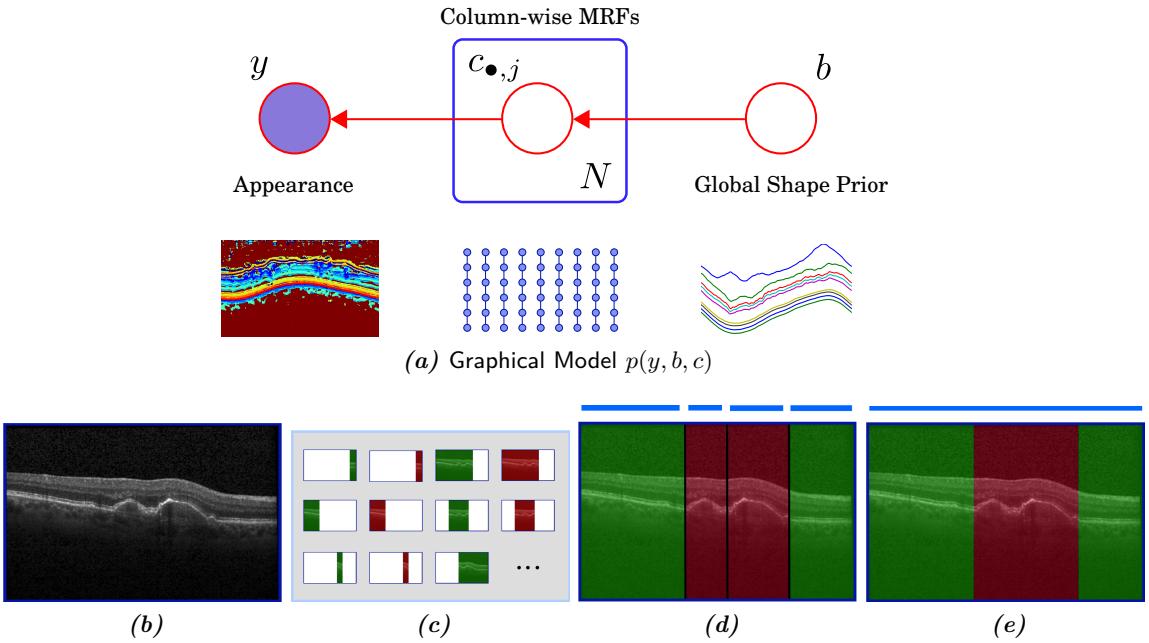


Figure 5.3.3. - (a) The baseline graphical model used at the leaves of the SPN, combining shape and texture components by using a Markov Random Field (b,c,d,e) Our workflow: Given a new OCT scan (b), we segment many local candidate models (c) (Sec. 5.3.2) being either modified (red = ill) or unmodified (green = healthy), where each candidate model corresponds to one leaf of the SPN. We then find the globally optimal combination by running MAP inference in the SPN (d). Finally we merge the local models into a global one to obtain a smooth segmentation (e).

that allow for parallel inference (with more details to be found in Rathke et al. [2014])

$$p(c|b) = \prod_{j=1}^{N} p(c_j|b), \qquad p(c_j|b) = p(c_{1,j}|b) \prod_{k=2}^{K} p(c_{k,j}|c_{k-1,j}, b). \tag{5.3.5}$$

Inference. Rathke et al. [2014] propose a variational scheme: Design a tractable graphical model $q(c, b)$ by adding conditional independences to $p(c, b|y)$, and then infer the full distribution $q(b, c)$ by minimizing the Kullback-Leibler (KL) divergence to $p(c, b|y)$. They decouple the discrete and continuous model components, $q(c, b) = q_c(c)q_b(b)$ while keeping the remaining structure intact: That is $q_c(c)$ are column-wise MRFs as in (5.3.5) and $q(b) = \mathcal{N}(b; \bar{\mu}, \bar{\Sigma})$. Infering $q(c, b)$ then corresponds to minimizing the following non-convex optimization problem

$$\min_{q_c, \bar{\mu}, \bar{\Sigma}} \mathrm{KL}(q(c, b) \| p(c, b|y)) = \int_b \sum_c q(c, b) \log \frac{q(c, b)}{p(c, b|y)}. \tag{5.3.6}$$

Plugging in the definitions of $q(c, b)$ and $p(c, b|y)$ one can find explicit update equations for the parameters of q_c and q_b. Of interest for this work is the update step for $\bar{\mu}$, which

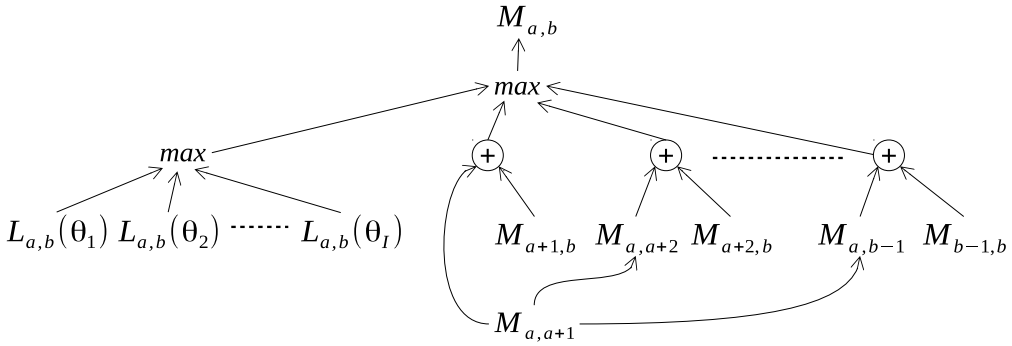


Figure 5.3.4. - A recursive step of Eq. 5.3.10 represented as a SPN in its MAP form (Section 3.1.2). Notice that lower level operations are *reused* by several upper level nodes.

is of the form

$$A(\bar{\mu} - \mu) = \mathbb{E}_{q_c}[c] - \mu \quad \Longrightarrow \quad A\bar{\mu} = (A - I)\mu + \mathbb{E}_{q_c}[c], \tag{5.3.7}$$

which links the mean of q_c ($\mathbb{E}_{q_c}[c]$) to the mean of q_b ($\bar{\mu}$) via the linear mapping A determined from Σ. We will revert to this equation at the end of the next Section, see Eq. (5.3.13).

Optimization. $q_c(c)$ is initialized by setting terms $p(b_{k,1}|b_{\setminus j}) = 1$ and solving the MRF in (5.3.5) taking into account appearance (5.3.2). After that one solves for $\bar{\mu}$ and $\bar{\Sigma}$ while keeping q_c fixed and vise versa until convergence.

5.3.3. Locally Adaptive Priors by SPNs

The model described above, when trained on healthy data, is not sufficiently flexible to adapt to unseen pathologies with large deformations. We address this problem by finding a global optimal combination of locally modified submodels using the principle of maximum-likelihood and dynamic programming.

We assume that models of pathological structure are *translation invariant*, *local* and *approximately independent*. Independence and locality allow to factorize the full distribution $p(y, b, c)$ into local distributions, an assumption necessary for SPNs. Translation invariance implies that the pathology can appear at any horizontal position in the image.

Recall that W in (5.3.3) contains typical shape variations of healthy retina layers. We adapt the graphical model towards an illness, by adding translation-invariant pathology-specific modes to W:

$$\theta_{a,b}^{\text{ill}} := \begin{pmatrix} W_{a,b} & W_{a,b}^{ill} \end{pmatrix}, \qquad \theta_{a,b}^{\text{healthy}} := W_{a,b} \tag{5.3.8}$$

Here subscript a, b denotes the pruning of W to entries lying inside columns a and b. Let

$L_{a,b}(\theta^k_{a,b})$ be the Log Likelihood of the segmentation for region a, b:

$$L_{a,b}(\theta^l_{a,b}) := \log q\left(c_{a,b}, b_{a,b}|y_{a,b}, \theta^l_{a,b}\right), \qquad l \in \{\text{healthy}, \text{ill}\}. \tag{5.3.9}$$

Not let $X = \{x_1, x_2, ..., x_K, N\}$ denote the division into $K+1$ regions and $\theta = \left\{\theta^{l_1}_{1,x_1}, \theta^{l_2}_{x_1,x_2}, ..., \theta^{l_K}_{x_K,N}\right\}$ denote a corresponding set of shape modifications. We want to find the combination of submodels with maximal total Log Likelihood

$$\max_K \max_X \max_\theta \; L_{1,N}(\theta) = L_{1,x_1}(\theta^{l_1}_{x_1,x_2}) + \ldots + L_{x_K,N}(\theta^{l_K}_{x_K,N}) \tag{5.3.10}$$

The global optimal of this combinatorial problem can be found with *dynamic programming*. Let $M_{a,b}$ be the optimal selection of X and θ in region $[a, b]$. It can be computed recursively as:

$$M_{a,b} = \max\left(\max_{x\in(a,b)}\left(M_{a,x} + M_{x,b}\right), \max_{l_i\in\{\text{ill,healthy}\}} L_{a,b}(\theta^{l_i}_{a,b})\right), \tag{5.3.11}$$

which is the maximum between the single best model over area $[a, b]$ and the optimal factorization in two adjacent areas. To compute $M_{a,b}$ for regions of width w, we need quantities $M_{a,x}, M_{x,b}$ for all regions of width $< w$. Given $M_{a,x}$ and $M_{x,b}$, the complexity is dominated by the evaluations of $L_{a,b}(\theta^{l_i}_{a,b})$.

Assuming a minimal width w_{min}, this suggests an iterative algorithm: first, compute $M_{a,b}$ for regions width w_{min}. Then, recursively compute Eq. (5.3.11) for regions of increasingly higher width. We can reduce complexity even further, by increasing and shifting windows by some fixed step size $s > 1$. Due to the nature of dynamic programming, many terms $M_{a,b}$ get reused during the optimization. To favor more compact subregions, we add a regularization to (5.3.10) to punish models of small size. This algorithm implements globally optimal MAP-inference in a SPN, whose structure is defined by Eq. 5.3.10 and is shown in Fig. 5.3.4.

Combining Local Models. In this section we describe a post processing approach used to smooth the MAP estimates produces by inference in the SPN.

Since there are no interdependencies between submodels found in (5.3.10), due to the product factorization between the leaves, there may occur jumps in the segmentation. To obtain a *smooth* solution for the full B-Scan, we combine them into a modified version of the full graphical model $p(y, b, c)$.

First note that to decouple any two regions $\mathcal{R}_1 = [1, b]$ and $\mathcal{R}_2 = [b+1, N]$ in the graphical model, we have to set all entries $\Sigma_{i,j}$ with $i \in \mathcal{R}_1$ and $j \in \mathcal{R}_2$ and vice versa to zero. These two regions now become inferred completely independent. Now, to adapt the MAP estimate(5.3.10) accordingly, we set

$$\tilde{\Sigma}_{\mathcal{R}_k} = \theta^l_{\mathcal{R}_k}(\theta^l_{\mathcal{R}_k})^T + \sigma^2 I_{\mathcal{R}_k}, \qquad k = 1, \ldots K, \tag{5.3.12}$$

and set all other entries to zero. Running the full graphical model with $\tilde{\Sigma}$ would yield

Table 5.3.1. - Unsigned error for all tested datasets in μm (1px = 3.87 μm). Surface numbers (1-9) correspond to Fig. 5.3.6 (a).

Dataset	Method	Avg.	1	2	3	4	5	6	8	9
RP	Tian et al. Tian et al. [2016]	4.48	**3.70**	4.49	3.84	–	5.75	–	–	4.63
	Our method	**4.08**	4.39	**4.15**	3.84	–	**4.65**	–	–	**3.37**
AMD	Our method	4.90	2.87	–	–	–	–	–	6.06	5.77
DME 1-5	Chiu et al. Chiu et al. [2015]	7.82	6.59	8.38	9.04	**11.02**	**11.01**	4.84	5.74	5.91
	Karri et al. Karri et al. [2016]	9.54	**4.47**	11.77	11.12	17.54	16.74	4.99	**5.35**	**4.30**
	Our method	**7.71**	4.66	**6.78**	**8.87**	11.02	13.60	**4.61**	7.06	5.11
DME 6-10	Chiu et al. Chiu et al. [2015]	5.81	5.01	6.37	7.46	7.34	7.74	3.88	4.34	4.32
	Karri et al. Karri et al. [2016]	5.14	3.64	5.95	6.48	**6.64**	8.00	**3.09**	**4.12**	3.17
	Our method	**5.11**	**3.62**	**4.87**	**5.92**	7.50	**7.69**	3.29	4.83	**3.16**

exactly the MAP estimate from above.

To smooth this segmentation, while staying as closely as possible to the original estimate, we modify the inference of $\bar{\mu}$: We replace (5.3.7) by the constrained least-squares problem:

$$\min_x \|\tilde{A}\bar{\mu} - (A - I)\mu - \mathbb{E}_{q_c}[c]\|^2, \qquad \text{subject to} \quad B\bar{\mu} \leq \delta \not\Vdash \tag{5.3.13}$$

Each row in the constraint matrix B selects two neighboring entries in $\bar{\mu}$ belonging to different regions $\mathcal{R}_k$ and restricts their distance to be less than δ. Solving the full graphical model with $\tilde{\Sigma}$ then yields a smooth segmentation, as the SPN output in Fig. 5.3.6 (b) and and the smoothed segmentation in (c) demonstrate.

5.3.4. Results

We demonstrate the flexibility of our approach by segmenting three different pathologies, ranging from minor deformations to severe distortions of the retina structure. We will use the same graphical model for all pathologies, only adapting the pathological shape modes we add.

- **Diabetic Retinopathy.** The dataset of Tian et al. (Tian et al. [2016]) contains 10 subjects with mild non-proliferative diabetic retinopathy (RP). Only small deformations occurred, so we simply used the healthy graphical model without the SPN framework. Unfortunately the information about the positions of the B-Scans inside the volume was missing. Since we trained individual 2-D shape priors, we require that information. So we tested all shape-priors for each scan and did two evaluations, giving an upper and lower bound on the true error: a) Use the largest likelihood of our model to pick a region and b) choose the region with the smallest error. We averaged both estimates to obtain the results in Table 5.3.1.

- **Age-related Macular Degeneration.** The AMD dataset is an in-house dataset

with 8 Spectralis volumes and labels for surfaces 1, 8 and 9 for all 19 B-Scans. The examples consisted of early and intermediate AMD. We added one mode with cosines for surfaces $6-9$, simulating the effect of those layers being pushed up by a (roughly) circular-shaped fluid deposit underneath. We included Bruch's membrane (surface 9) for performance reasons, even though it is supposed to lie below the fluid region. To obtain a correct segmentation for this surface, we evaluated the conditional mean $\mu_{a|b}$ of Eq. 5.3.4, where subscript b denotes the part of the segmentation that was identified as healthy. This worked well in most cases, but can fail if there are too many pathological parts. Future work will address this issue.

- **Diabetic Maculuar Edema.** The dataset of Chiu et al. (Chiu et al. [2015]) consists of 10 Spectralis volumes with 11 labeled B-Scans per volume. While volumes $6-10$ are mild and intermediate cases, volumes $1-5$ constitute advanced DME cases, with disappearing layers (Fig. 5.3.2 (c)) and advanced texture artifacts due to highly reflective regions characteristic for DME (Fig. 5.3.6 (c)). As these artifacts do not occur in healthy scans, our texture models failed in these regions. We dealt with this problem by additionally using smaller patches of size 7×7 and 3×3 (besides the standard 15×15 patches), which reduced sensitivity to these artifacts. To deal with the disruptive layers, we drop the segmentation in very low intensity regions when the total width exceeds a certain threshold. Karri et al. (Karri et al. [2016]) also tested their approach on this dataset, but only published results for volumes $6-10$, using the first 5 volumes for training. Using their published code (`https://github.com/ultrai/Chap_1`), we reversed training and test set to obtain results for volumes $1-5$. We also could reproduce their results.

Results are displayed in the 2 lower blocks of Table 5.3.1 and in Fig. 5.3.6 (d)-(f). In general, volumes $6-10$ yield lower errors for all approaches as expected. Furthermore, Karri's and our approach outperform Chiu et al. The picture changes for volumes $1-5$. Now Chiu's and our approach perform on par, beating the one of Karri et al. This is caused by the lack of shape regularization in their approach, inferring surfaces without taking others into account.

Pathology hinting. Finally, Fig. 5.3.5 demonstrates another benefit of using a shape prior. The red marked surfaces showcase the added pathological modes $W^{ill}_{a,b}$, indicating where the healthy shape prior had to be modified to fit the pathological scan.

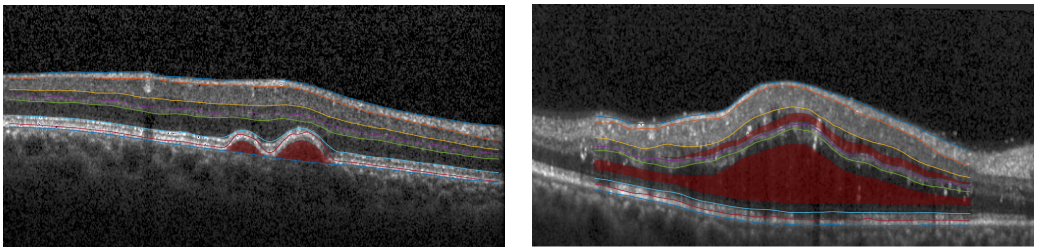
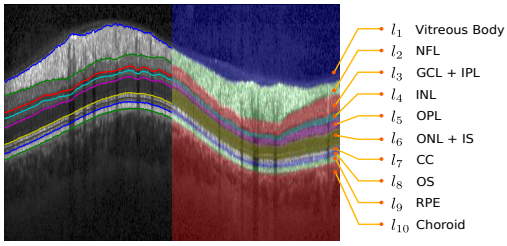
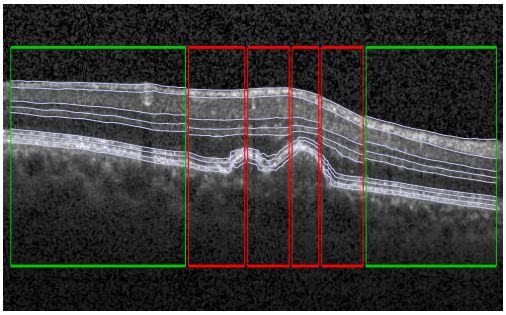


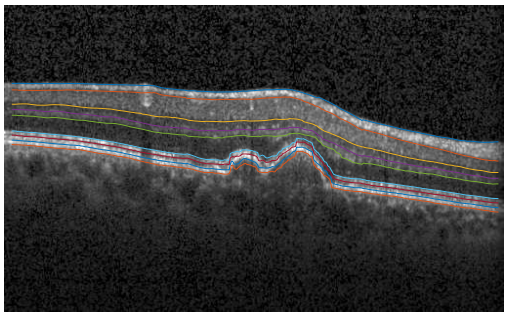
Figure 5.3.5. - Estimates of the fluid regions due to the selected pathological modes.



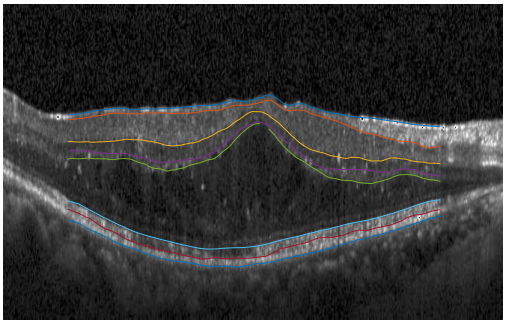
(a) Segmented retina layers.



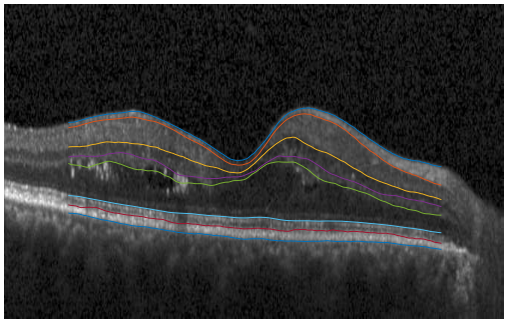
(b) SPN output



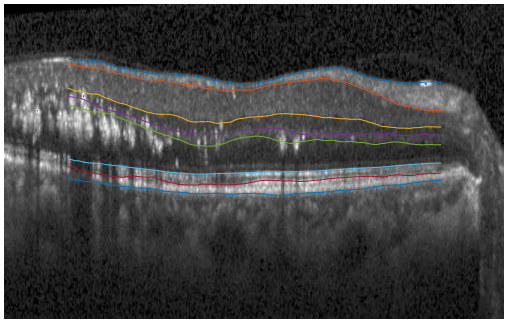
(c) Smoothed estimate



(d) Large fluid deposit



(e) Several small deposits



(f) Hyperreflecive loci

Figure 5.3.6. - (a) The names of the segmented retina layers. Surfaces 1-9 lie in between layers $l_1, \ldots, l_{10}$. Used with permission from Rathke et al. [2014]. (b) and (c) Combining and smoothing a SPN estimate. Note that Bruch's membrane (surface 9) gets fitted in a post-processing step, described in the results section. (d)-(f) Example segmentations from the DME dataset.

5.4. Very Large Mixture of Spanning Trees for Density Estimation in Layered Distributions

In this section we present a *preliminary* experiment regarding the use of SPGMs for approximating an intractable graphical model $\mathcal{G}$. To perform a preliminary investigation on this property, we limit our discussion here to quantitatively evaluate if such an approach works in practice in a small academical setting, leaving a full fladged discussion to future work.

We start with the observation that mixtures of spanning trees have been used extensively to approximate intractable graphs (see e.g. Meila and Jordan [2000], Bach and Jordan [2001], Pletscher et al. [2009]). In these models, the approximation quality typically increases with the number of components in the mixture, hence the ability of SPGMs to model very large mixture of trees suggests that SPGMs might be apt for this approach (as already seen in Section 5.2).

Hence, the procedure that we employ is twofold. First, we find a mixture of spanning trees over $\mathcal{G}$ such that many parts between the trees are shared, in such a way as to implement this mixture efficiently with a SPGM $\mathbb{S}$. Secondly, we learn the parameters governing $\mathbb{S}$ by maximizing the Log Likelihood of a set of samples taken from $\mathcal{G}$. This can be done in case $\mathcal{G}$ is a directed graphical model, for which samples can be obtained efficiently with Ancestor Sampling even if inference is infeasible (Pearl [2000]).

However, finding a set of spanning trees with shared parts, which can then be efficiently represented as SPGM, is not a simple problem. We leave a full fledge discussion of this approach for future work, and in this preliminary application we consider a class of models for which such mixture can be easily found.

Layered Distributions. A large mixture of spanning trees can be obtained with a simple heuristic in the case of *layered distributions*. We define a layered distribution as a directed GM composed by successive layers of variables, where variables in one layer connect only to variables in the next one. Variable X_k^l denotes the k-th variable at layer l (Fig. 5.4.1a). This class of distributions is relevant in applications, since it includes Factorial Hidden Markov Models, Multiscale Quadtrees (Wainwright and Jordan [2008]) and deep belief networks (Hinton and Osindero [2006]). Inference cost in Layered Distributions is worst case exponential in the layer size and it is therefore intractable. However, samples can be obtained efficiently with ancestral sampling.

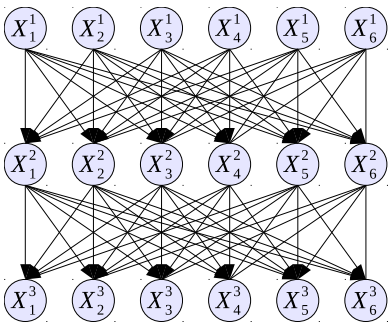
A spanning tree can be taken from a layered distribution by allowing a single "active variable" to have children at each layer (Fig. 5.4.1b). It is easy to see that if two trees T_1 and T_2 taken in this way differ only by the choice of one active variable, then their structure is largely shared - this is due to the fact that parts of the trees corresponding to the same active variables are identical.

The mixture of many spanning trees with this structure can be modeled compactly with the SPGM shown in Fig. 5.4.2: notice that any subtree in this model corresponds

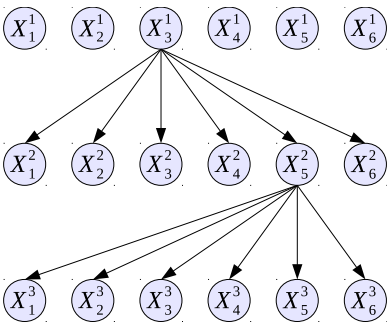
to a tree in the form of Fig. 5.4.1, right, hence the SPGM encodes the mixture of all such trees (Proposition 4.1.7).

In addition, we can also allow more than one variable to be active at the same time, i.e. allow more than one variable in the same layer to have children (Fig. 5.4.1c). This can be done by creating a clique by merging the state of all active variables in a single node: e.g., if two variables A and B are active at a certain layer, then we create a node associated to a variable $\{A, B\}$ with values in $\Delta(A) \times \Delta(B)$, which merges the individual variables (Fig. 5.4.1d).

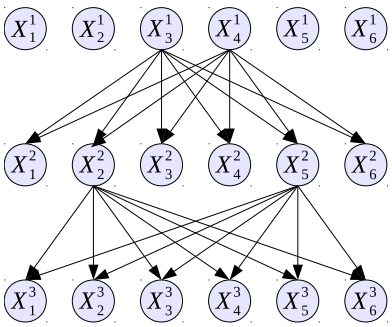
The resulting SPGM efficiently encodes a *very large* mixture of trees with shared parts. Let the model contain L layers, and let there be C choices of active variables at each layer. Then it is immediate to see that the number of subtrees, each of which corresponds to a spanning tree, grows as C^L due to the combinatorial number of choices of active variables at each layer. However, due to Proposition 4.1.1 inference in the SPGM has just $O(LC^2)$ cost in memory and time. This exponential reduction in inference cost is made possible by exploiting the fact that many parts of the trees are shared.



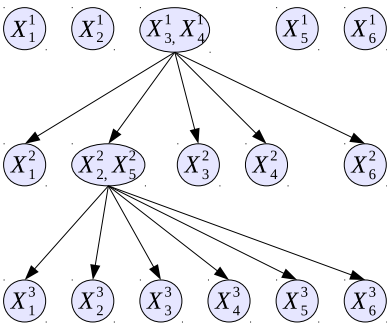
(a) A subsection of a layered directed GM.



(b) A subsection of a spanning tree of (a).



(c) Allowing two active variables per layer.



(d) The model in (c) represented as a tree.

Figure 5.4.1. - A mixture of spanning trees with shared subparts obtained from a layered directed graphical model, as discussed in Section 5.4.

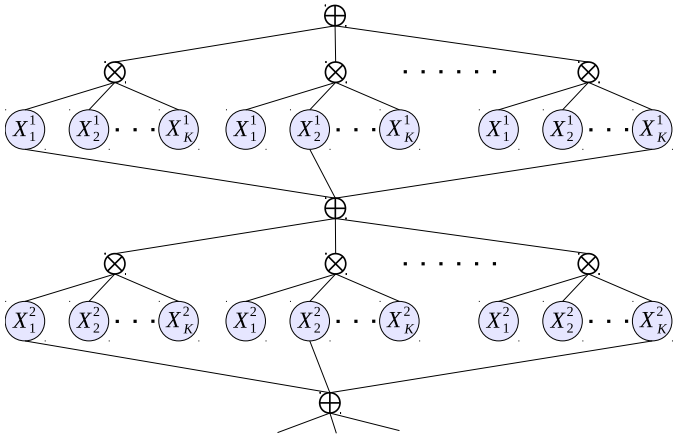


Figure 5.4.2. - First two layers of an SPGM encoding a mixture of spanning trees in a layered model with K variables per layer. X_k^l denotes the k-th variable at layer l.

Empirical Evaluation. We tested the SPGM on a layered mixture model with 10 layers, each containing 6 binary variables. As described above, we created a SPGM encoding a mixture of spanning trees over this model, whose parameters are learned by taking a set samples from the layered distribution, dividing them into training and test set and maximizing the training Log Likelihood via EM (Section 3.3.2).

We report Log Likelihood results obtained by SPGMs and several well established methods for density estimation in Table 5.4.1. We test SPGM models using a different number of choices of active variables per layer (i.e. the number of sum node children), which result in an increasingly large number of subtrees in the resulting mixture model. Choosing from 1 to 8 possible active variables per layer, the mixture size ranges from 1 to 8^{10}. We also rest different numbers of active variables at each layer (1,2 and 4). We first compared against methods based on trees, namely the optimal spanning trees (Chow and Liu [1968], see Section 2.3.4) and mixture of trees trained with EM (Meila and Jordan [2000], see Section 2.4). We report separate results depending on the number of trees in the mixture. Then, we compared against two state-of-the-art density estimation methods for SPNs: ACMNs (Section 3.4.1) and ID-SPN (Section 3.4.2).

From the quantitative results it is evident that SPGMs *widely* outperform all competing methods in terms of test set LL. In particular, they do not seem to suffer from the overfitting problem that plagues mixtures of tree even for moderately large mixture sizes. In addition, very large mixtures with up to 8^{10} tree components can be modeled tractably: learning time was about 5 minutes in a non-optimized MATLAB implementation. We hypothesize that this is due to the strong regularization imposed by sharing the structure, and hence the parameters, between the trees in the mixture. The results of this preliminary experiment show that using SPGMs as approximation of an intractable graph with known structure is a very interesting research direction, to be explored in future work.

Table 5.4.1. - Log Likelihood values for the experiment in Section 5.4, for different models (rows) and for different number of trees in the mixture (columns, only defined when the model is a mixture of trees). Unavailable result are due either to excessive run time (for mixture of trees with more than 2^{10} components) or because the algorithm is not applicable for the given number of trees (for SPGMs). Notice that SPGMs obtain by far the best test set LL scores.

model/#trees	1	10	20	40	2^{10}	4^{10}	6^{10}	8^{10}
Chow-Liu tree, Train	−27.7							
Chow-Liu tree, Test	−30.2							
Mix. Tree, Train		−20.6	−17.5	−14.2				
Mix. Tree, Test		−23.4	−23.7	−24.2				
SPGM, 1 act. var. per layer, Train	−34.1				−24.7	−18.4	−16.3	
SPGM, 1 act. var. per layer, Test	−34.5				−25.7	−20.0	−17.8	
SPGM, 4 act. vars. per layer, Train	−31.5				−21.2	−16.5	−15.4	−14.8
SPGM, 4 act. vars. per layer, Test	−32.1				−22.6	−18.1	−17.1	−16.5
SPGM, 6 act. vars. per layer, Train	−20.7				−14.9	−13.6	**−13.3**	−13.8
SPGM, 6 act. vars. per layer, Test	−22.1				−16.9	−15.6	**−15.1**	−15.9
ACMN, Train	−21.7							
ACMN, Test	−23.9							
IDSPN, Train	−19.7							
IDSPN, Test	−21.6							

5.5. SPGMs for Quadtree Images Models

In this section we show an example application of SPGMs to model a large mixture of Quadtrees for image denoising (Laferté et al. [2000]). Starting from the observation that Quadtrees are tree graphical models, and that SPGMs are good at modeling large mixtures of trees, we discuss how a particular form of factors allows SPGMs to represent a very large mixture of Quadtrees. Empirical results in a preliminary experiment denote potential for future real world applications.

Quadtrees Quadtree models (see Laferté et al. [2000]) provide a hierarchical representation of $2D$ images as a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ where each node $s \in \mathcal{V}$ is associated to a square area of the image denoted R_s, as follows: leaf nodes corresponds to pixels, and nodes at higher levels are iteratively composed by connecting four adjacent nodes at lower levels (Fig. 5.5.1). A directed tree graphical model is defined on a Quadtree by associating each node $s \in \mathcal{V}$ to a variable X_s with domain $\Delta(X_s)$, and assigning probability $P_r(X_r)$ to the root node $r \in \mathcal{V}$ and pairwise conditional probabilities $P_{st}(X_t|X_s)$ to each $(s,t) \in \mathcal{E}$.

Quadtrees have found several applications in image processing, due to the flexibility in choosing the potentials of the graphical model: for instance, they have been used to model relations between wavelet coefficients at multiple scales (Crouse et al. [1998]) and to perform segmentation (Feng et al. [2002]) and denoising (Beaulieu and Goldberg [1989]). We consider here Quadtrees for denoising of image labels. In this case, each variable X_s takes values in the discrete domain and each value represents a label in the scene (see e.g. images in Fig. 5.5.5). The pairwise potentials $P_{st}(X_t|X_s)$ in the quadtree

can assume several forms, but they are generally *attractive* with respect to the parent variable, i.e. they assign a higher probability when variable is in the same state as the child variable. One possible form is the Potts model, which assign a probability $p > 0$ when parent and child nodes are in the same state, and $q < p$ if they are in different state. Denoising is performed by taking the Maximum a Posteriori (MAP) state after performing inference on the Quadtree given the current observed pixels.

Quadtrees have been used for image modeling since their hierarchical structure naturally leads to a coarse to fine representations of the image at successive levels. Furthermore, they provide a natural mechanism for all regions in the image to have some influence over each other and thus exert global consistency, and they have potential for using the segmentations given at higher levels in image coding applications.

However, Quadtrees have a crucial limitation in the fact that the generated image models assume a "*blocky*" aspect, due to the fixed subdivision in square regions induced by the tree structure. To solve the problem two main approaches have been proposed:

- Grid structured Markov Random Fields (Pearl [2000]) represent images through a grid structured undirected graphical model where each node in the grid represents a pixel, thus do not suffer of blockiness. However, in contrast to Quadtrees inference in these models is generally is intractable and the representation is not hierarchical. The issue of dealing with a hierarchical representation for MRFs is considered e.g. in He et al. [2004].

- Dynamic Tree Models (Adams and Williams [2003]) aim to make the Quadtree structure more flexible by allowing each node to select an arbitrary parent between nodes in the previous level of the tree (in contrast, in Quadtrees each node has a fixed parent). The selection is based on a distribution on the set of all possible choice of parents. Inference in the joint distribution (involving the selection of tree edges) is intractable, but once the MAP tree has been found (with approximate inference) it becomes tractable.

We propose here an alternative approach: we use a SPGM to model a large mixture of Quadtrees, and choose the one whose structure best fits the image given the observed data. In this way the blockiness problem is, hopefully, alleviated by the flexible selection of edges.

The SPGM model for a mixture of Quadtrees A Quadtree SPGM is a SPGM that encodes a mixtures of Quadtrees with flexible subdivision of images in subregions. We provide here an description of how the SPGM Quadtree model is obtained.

Let us assume that the image covers a squared area, for simplicity. First, we create the root node r as a Vnode associated to the full image area. Below the root node we create a Sum Node, whose children are Product Nodes representing the splits of the parent's area as in Fig. 5.5.2a. The sum node encodes the "choice" of the split type, and the product nodes encode the actual split (notice that only one child of the Sum Node is

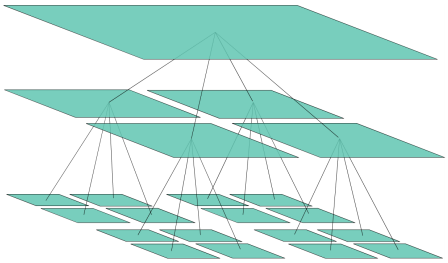


Figure 5.5.1. - Basic Quadtree structure, recursively square image regions into sub-areas up to single pixels.

included in a subtree, by Definition 4.1.7). Then, the procedure is repeated recursively for each Vnode corresponding to child areas: squared areas are subdivided with the rule of Fig. 5.5.2a, and rectangular areas with the rule of Fig. 5.5.2b, right. This procedure is repeated until the pixel level is reached.

Most importantly, we remark that this subdivision does not lead to an exponential explosion of the number of children, since many of the internal Vnodes can be *shared* between several higher level subdivisions. In fact, it is only necessary to create Vnodes corresponding to areas of size $w \times w$, $w \times 2w$ and $2w \times w$, for $w = \{1, 2, 4, 8, \dots, W\}$where W is the size of the top level region. With this structure it is then possible to model tractably a very large mixture of Quadtrees (exponential in the Quadtree depth), due to the fact that nodes in lower layers are shared between multiple trees.

The ability of SPGMs to reuse internal nodes in several Quadtrees is be seen by considering the example Quadtree SPGM shown in Fig. for the 1D case of a 4×1 pixel image: notice that several nodes have multiple parents and thus are shared by multiple subtrees (see Section 4.1.3).

Intuitively, each subtree in the SPGM can be seen as an extension of the single Quadtree model, where each node rather than being split in four fixed parts can be split in *four different ways* depicted in Fig. 5.5.2a. A full tree is obtained by recursively performing this split at lower regions, until regions covering a single pixel are reached. One of such subtrees is shown in Fig. 5.5.4.

Encoding mixtures of Quadtrees in a SPGM in this way has two main advantages:

- It allows to model tractably very large mixtures of trees (exponential in the depth of the SPGM, but still with polynomial inference cost due to Proposition 4.1.1).

- It alleviates the problem of blocky representations, since MAP inference can be performed exactly and thus the location of the borders can be chosen in order not to split continuous parts of the images. Notice that the rougher splits at higher level nodes corresponds to lower image resolutions, which agrees with the Laplacian Pyramid-like interpretation of Quadtrees in image models.

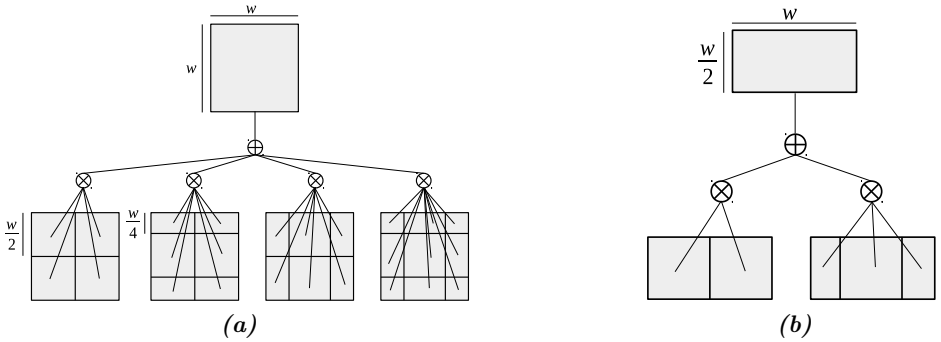


Figure 5.5.2. - (a) A node covering a square region of the image (top) has as children the mixture of 4 possible splits of the region (below). Note that the sum node act as a mixture, and that this can be seen as the mixture of 4 trees, one for each child of the sum node. The standard Quadtree model is obtained when only the leftmost split is selected. (b) Subdivision for a rectangular region.

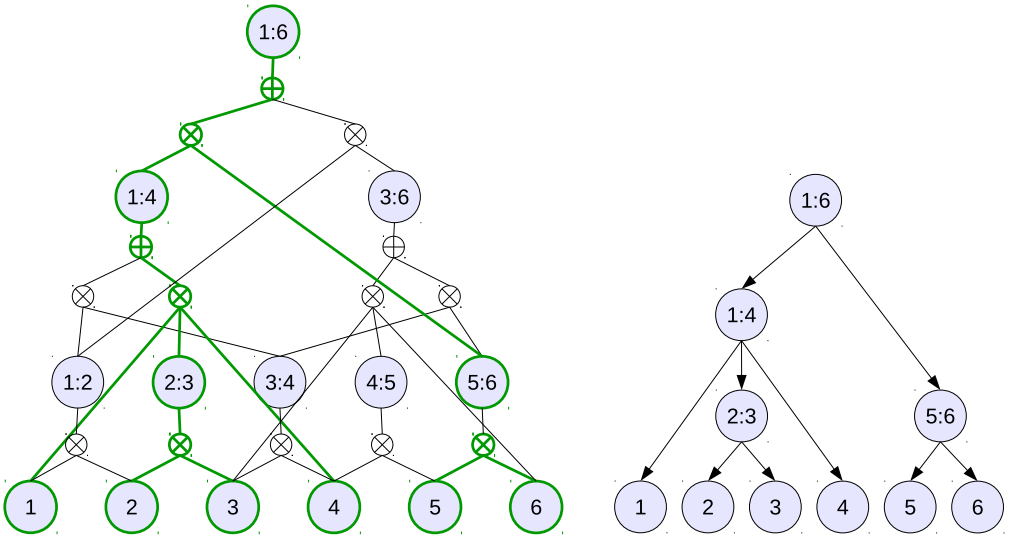


Figure 5.5.3. - An example SPGM $\mathbb{S}$ encoding a mixture of Quadtrees in 1D. Left: Quadtree SPGM over a 6×1 rectangle, recursively divided according to Fig. 5.5.2a, but in 1D . Labels $a : b$ denote variables associated to the area from a to b inclusive. In green, a subtree of $\mathbb{S}$. Right: the subtree in green represented as a directed GM. Each subtree in the model assumes this form, and $\mathbb{S}$ encodes the mixture of all such subtrees.

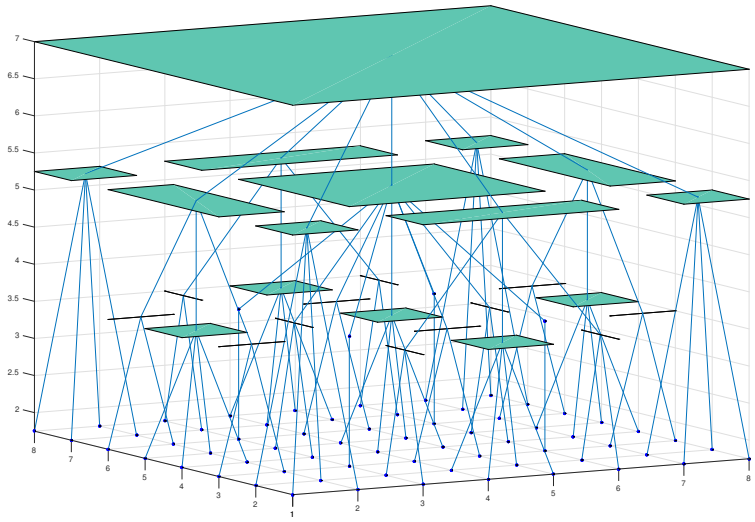


Figure 5.5.4. - 3D representation of one of the subtrees of the SPGM encoding a mixture of Quadtrees over an 8×8 pixel image. Notice that the selection of region borders can change locally. A Quadtree SPGM represents the mixture of all possible subtrees of this kind.

A Simple Denoising Experiment In this preliminary experiment we test the performances of SPGM Quadtrees against standard Quadtrees and against grid Markov Random Fields (MRFs). To this aim, we used the labeled images in the Sowerby dataset described in Adams and Williams [2003], which contains 105 labeled images of size $96x64$ and 7 labels. This dataset contains a set of road scenes and has been widely used in literature on semantic segmentation.

In our simple experiment we add random noise to the dataset labels and try to recover the original labels by performing MAP inference on standard Quadtrees, SPGM Quadtrees and on MRFs (in which inference is computed by Tree Reweighted Message Passing by using the OpenGM toolkit (Andres et al. [2012])). We created the structure of a SPGM $\mathbb{S}$ with the algorithm described above, and used simple Potts-like potentials with probability of maintaining the same state 0.7.

Accuracy of the reconstruction if 88.4% for standard Quadtrees, 90.2% for SPGM quadtrees and 90.6% for Potts models. Notice that the results improve over standard quadtrees, although they are worse than MRF models, which are not restricted to rectangular sub-areas of the image and are therefore mode apt for this application. In addition, SPGM Quadtrees allow to use a hierarchical structure, a characteristic which differentiates them from planar MRFs and is exploited in the next experiment. Graphical results of the denoising procedure are shown in Fig. 5.5.5.

Comparison with Dynamic Trees for Image Compression

Similarly to SPGMs, Dynamic Trees (Adams and Williams [2003]) alleviate the problem of fixed region borders by selecting the optimal tree (via approximate inference) and then performing denoising on the selected tree (see the discussion above). In contrast to SPGMs, inference in Dynamic Trees cannot be computed exactly. However, they allow more flexibility than SPGMs in selecting the tree structure, since a node at level l can be connected to *any* parent in level $l-1$.

First, in order to highlight the ability of SPGMs to select dynamically the best tree, we show the map tree obtained over 1d binary signals in figure 5.5.6 in comparison to Quadtrees and Dynamic Trees. Notice that SPGMs and Dynamic Trees can both adapt their structure to the input signal, in contrast to standard Quadtrees.

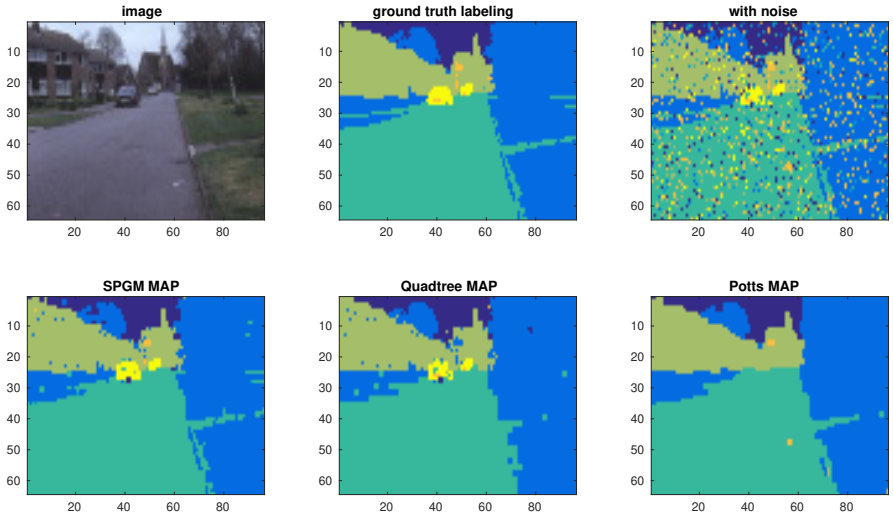
Then, we compare the two models in practice by reproducing the experiments performed in Adams and Williams [2003]. The Dynamic Tree, Quadtrees and also SPGMs are generative, probabilistic models of label images. We can evaluate the quality of the learned models by calculating the Log Likelihood on the test set of 43 images used by Adams and Williams [2003], which can then be used to obtain the average coding cost in *bits per pixel* (bpp) of an image X with N pixels, given by:

$$\text{Coding Cost} = -\frac{\log_2 P(X)}{N},$$

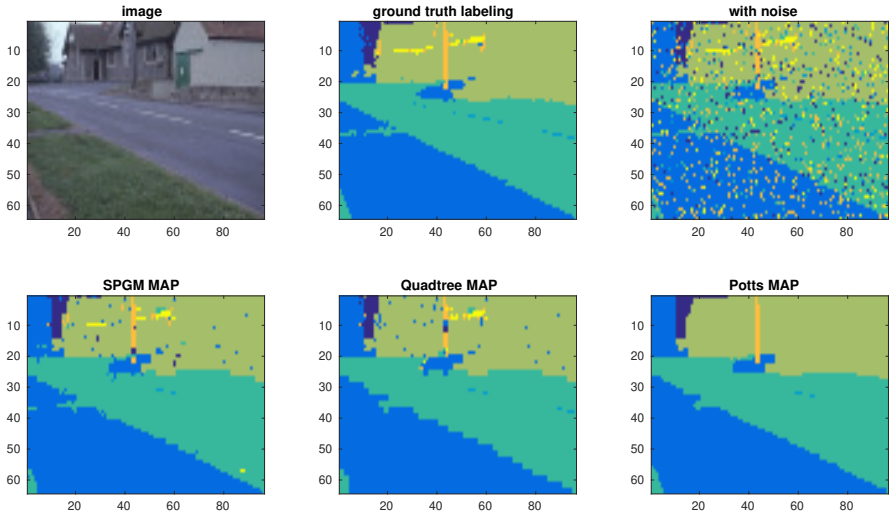
where P is the probability function of the model. Lower coding cost denotes better performances. The results shown in Table 5.5.1 show that our model is in this sense better than Dynamic Trees. Hence, this preliminary application indicates that SPGMs for mixture of Quadtree represent, once again, an interesting direction of research.

Model		Coding Cost (bits per pixel)
Mean Field	Fixed Tree	0.8588
	Dyn. Tree - CPT only	0.4089
	Full Dyn. Tree	0.3805
Exact EM	Fixed Tree	0.3421
JPEG-LS		0.3810
SPGM		**0.3103**

Table 5.5.1. - Coding Cost of the test set in the test set of 43 images from the Sowerby dataset described in Adams and Williams [2003]. Lower value indicates better performance.



(a)



(b)

Figure 5.5.5. **-** Denoising using SPGMs, Quadtrees, and a Markov Random Fields with Potts potentials, performed on two images from the Sowerby dataset (Adams and Williams [2003]). Note that the images denoised with SPGMs suffer less from blockiness than Quadtrees (but more than the Markov Random Fields, which does not assume rectangular image regions). In addition, SPGMs preserve more details than both competing models - see e.g. the car in (a).

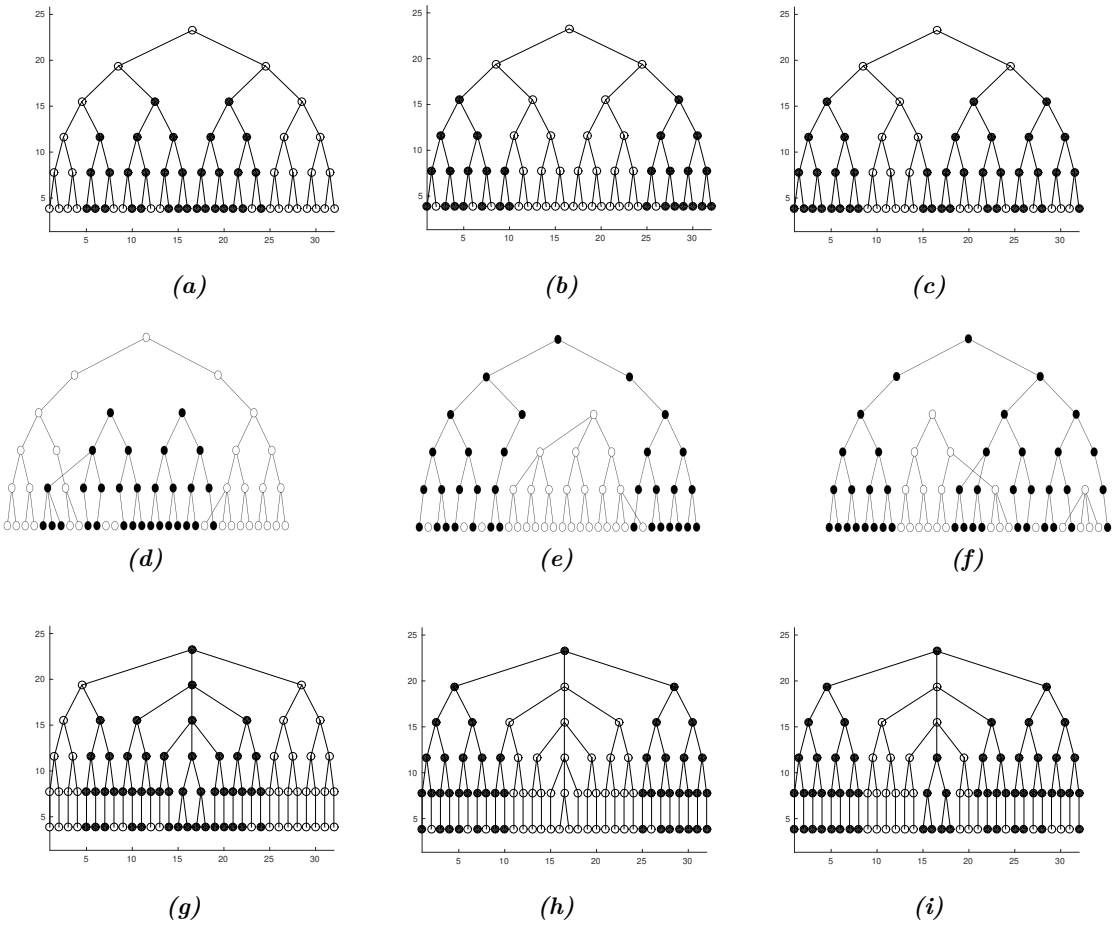


Figure 5.5.6. - Comparison of MAP tree obtained with Quadtrees (row 1), dynamic trees (row 2) and SPGMs (row 3) on three data samples (left to right). Note that the trees obtained in row 1 (Quadtrees) have fixed structure, while the ones in 2 and 3 have flexible structure that can adapt to the data. Figures in row 2 are taken from corresponding examples in Adams and Williams [2003].

6. Conclusions and Future Work

6.1. Conclusions

In this thesis we investigated and developed connections between the field of probabilistic Graphical Models (GMs) and Sum-Product Networks (SPNs).

First, we investigated SPNs that employ GMs as leaf models, obtaining a derivation of EM that allows to learn jointly the SPN parameters and the GM leaves ones.

We introduced Sum-Product Graphical Model (SPGM), a new architecture bridging Sum-Product Networks (SPNs) and Graphical Models (GMs) by inheriting the expressivity of SPNs and the high level probabilistic semantics of GMs. The *new connections* between the two fields were exploited in a structure learning algorithm extending the Chow-Liu tree approach.

Finally, we presented several applications of the new tools and architectures, with a particular focus on SPGMs. We found that the structure learning algorithm LearnSPGM is competitive with the state of the art methods in density estimation despite using a novel approach and being the first algorithm to directly obtain DAG structured SPNs. Furthermore, we showed some application settings of SPGMs that denote promise of such architecture in real world scenarios that are typically out of the domain of SPNs.

6.2. Future Work

As a concluding note to this thesis, we present several extensions to the framework of Sum-Product Graphical Models that should be subject of future work.

6.3. Generalizing SPGMs

The definition of SPGMs can be generalized in several ways to adapt it to different application settings:

- An undirected representation of SPGMs can be obtained straightforwardly by replacing conditionals probabilities in Definition 4.1.4 with non-normalized potentials. Namely, terms $P_s(X_s)$ and $P_{st}(X_t|X_s)$ in the SPGM definition can be replaced by factors $\varphi_s(X_s)$ and $\varphi_{st}(X_s, X_t)$, whose elements are constrained to be non negative but they are not constrained to be normalized. Since message passing in directed and undirected *tree* graphical models assumes the same shape, all the subsequent propositions on SPGMs (that are based on message passing for trees) still maintain

validity.
In contrast to the directed GM case, however, there is need to normalize the distribution by dividing for the partition function. The partition function can be efficiently computed with an evaluation of the SPGM with no observed variables, due to Proposition 4.1.7.

- Continuous variables can be used at *leaf nodes*, but not at the internal nodes. This can be done because summation can be substituted with integration at the leaf messages while at the same time maintaining a discrete number of states in the leaf messages (cf. Definition 4.1.5) . Thus, leaf messages can be passed to internal nodes exactly sas in discrete SPGMs. This allows to apply SPGMs to situations where the leaf variables should be continuous, which arise often e.g. in the field of image processing.

- The expressive power of SPGMs can be extended by modeling mixtures of Junction Trees rather than modeling mixtures of trees. In this way, each Junction Tree corresponds to inference in a graphical model with cycles, but with tractable treewidth (Section 2.3.3).
  This can be done by performing three changes: firstly, each Vnode $t \in \mathcal{V}$ should be associated to a set of variables X_t, like nodes in Junction Trees, as opposed to a single variable like in SPGMs. Secondly, the running intersection property must be enforced between s and each $s \in vpa(t)$ (cf. 2.3.3). Finally, the Vnode message must be modified to match the message sent between nodes in Junction Trees (Eq. 2.3.7).
  With these changes, it is possible to show that SPGMs encode a mixture of Junction Trees exactly in the same way as usual SPGMs encode a mixture of trees.

6.4. Tree-Reweighted Message Passing with SPGMs

An interesting aspect of tractable probabilistic models is to approximate inference in intractable models, and a particularly relevant application of this concept lies in the approximation of intractable graphical models by a *convex combination of trees*. This line of research was first introduced for the maximization of the energy of intractable GMs in Wainwright et al. [2002] (Tree-Reweighted Message Passing). Then, Wainwright et al. [2003] extended the approach to the approximation of the log partition function, which allows to compute approximate marginals. Later, Kolmogorov [2006] showed a convergent procedure called Sequential that is guaranteed to increase a lower bound of the MAP energy iteratively. Finally, Meltzer et al. [2009] unified these approaches by showing that the convergent sequential procedure described in Kolmogorov [2006] also allows to find an approximation of the log partition function by simply substituting summations with max. This algorithm provides an efficient, sequential convergent approximation for the partition function.

At a high level, Tree-Reweighted Message Passing aims to approximate a certain probability distribution $P(X)$ governed by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a mixture model $\prod_{k=1}^{K} \rho_k T_k(X)$, where $\{\rho_k \geq 0\}$ and $\{T_k\}_{k=1}^{K}$are tree graphical models. It requires taking each edges $(s,t) \in \mathcal{G}$ in a given order and re-estimating its associated pairwise probability P_{st} with the following rules:

1. Reparametrization Step: compute the marginal probability $T_k(X_s, X_t)$ for each tree $\{T_k\}_{k=1}^{K}$ in the mixture model (which requires passing messages in the tree).
2. Averaging Step: Substitute P_{st} with the weighted average $\overline{P}_{st} = \sum_{k=1}^{K} \rho_k T_k(X_s, X_t)$.

By iterating the procedure above and choosing a particular order of edges, Kolmogorov [2006] showed that the algorithm converges to a local minimum of the Kullback-Leibler divergence with respect to the original distribution.

A crucial step to make this procedure tractable is to use a mixture in which several sections of the trees are shared, in such a way that messages can be computed only once and parameters can be updated simultaneously for a large number of trees in the mixture. Kolmogorov [2006] obtains this by using a mixture model in which each element is a chain graph (i.e. each node is connected to at most two nodes), and in which nodes in all chains are consistent with a given ordering: since messages are in common between several chains, inference in the mixture can be greatly simplified. Furthermore, the marginals can be computed with only *local operations* due to the message structure. The authors note that using *longer chains* and a *large number* of mixture components empirically helps to obtain better maxima and faster convergence.

The last observations make using SPGMs for Tree-Reweighted Message Passing an appealing direction of future research, due to their ability to compactly model very large mixtures of trees by naturally sharing subsections of the trees and reusing the related messages. In particular, a preliminary analysis of this application showed that steps 1 and 2 above can be performed with *local operations* in SPGMs, similarly to what happens in mixture of chain graphs in Kolmogorov [2006]. Thus, Tree-Reweighted Message Passing can potentially be applied to the very large, hierarchical mixture of trees encoded by SPGMs. This opens an interesting research area in learning SPGM to approximate an intractable graphical model, which should be subject of future work.

A. Appendix - Proof Details

A.1. Proof of Proposition 3.1.8

Proof. Consider the mixture implementation of a SPN (Eq. 3.1.6), and let us compute the marginal w.r.t. a variable $V \in X$ by summing out the remaining variables $X \backslash V$:

$$\sum_{x \in \Delta(X \backslash V)} S(X) = \sum_{\sigma \in \Sigma(S)} \lambda_\sigma \sum_{x \in \Delta(X \backslash V)} P_\sigma(X).$$

Now, let us note that each element P_σ is by Eq. 3.1.8 the product of some subset of the leaf distributions $L_c \subseteq \mathcal{L}(S)$, in the form $P_\sigma = \prod_{i \in L_\sigma} \varphi_i(X_i)$. Since the scopes are disjoint (see Eq. 3.1.8 again), one and only one of these leaves includes V in its scope. Calling this leaf φ_{v_σ}, we can separate the product $P_\sigma = \prod_{i \in L_\sigma} \varphi_i(X_i)$ as $P_\sigma = \varphi_{v_\sigma}(X_{v_\sigma}) \prod_{i \in L_\sigma \backslash v_\sigma} \varphi_i(X_i)$. Then, moving the sum $\sum_{x \in \Delta(X \backslash V)}$ inside the product we obtain:

$$\begin{aligned}
\sum_{x \in \Delta(X \backslash V)} S(X) &= \sum_{\sigma \in \Sigma(S)} \lambda_\sigma \sum_{x \in \Delta(X \backslash V)} \left(\varphi_{v_\sigma}(X_{v_\sigma}) \prod_{i \in L_\sigma \backslash v_\sigma} \varphi_i(X_i) \right) \\
&= \sum_{\sigma \in \Sigma(S)} \lambda_\sigma \left(\left(\sum_{x_{v_\sigma} \in \Delta(X_{v_\sigma} \backslash V)} \varphi_{v_\sigma}(X_{v_\sigma}) \right) \prod_{i \in L_\sigma \backslash v_\sigma} \underbrace{\sum_{x_i \in \Delta(X_i)} \varphi_i(x_i)}_{1} \right) \\
&= \sum_{\sigma \in \Sigma(S)} \lambda_\sigma \left(\underbrace{\sum_{x_{v_\sigma} \in \Delta(X_{v_\sigma} \backslash V)} \varphi_{v_\sigma}(X_{v_\sigma})}_{} \right).
\end{aligned}$$

Now, the inner sum assumes the same value for any two subtrees σ', σ'' whenever $\varphi_{v_{\sigma'}} = \varphi_{v_{\sigma''}}$- that is, when the leaf node is in common. Hence we can rearrange the sum as ranging over leaves rather than over all subnetworks, as follows:

$$\sum_{x \in \Delta(X \backslash V)} S(X) = \sum_{k \subseteq \mathcal{L}(S) : V \in k} \left(\left(\sum_{\sigma \in \Sigma(S) : k \in \sigma} \lambda_\sigma \right) \left(\sum_{x_k \in \Delta(X_k \backslash V)} \varphi_k(V, x_k) \right) \right).$$

Here $\sum_{k\subseteq\mathcal{L}(S):V\in k}$ denotes the sum over all leaves that contain V and $\sum_{\sigma\in\Sigma(S):\varphi_k(x)\in\sigma}$ denotes the sum over all subnetworks that include leaf k. Finally, using Lemma 3.1.7 the left part can be identified as the partial derivative of $S(x)$ w.r.t. $S_k(x)=\varphi_k(x)$, which concludes the proof. □

A.2. EM for SPNs

Preliminars. Let $S(X)$ be a SPN, and consider a subnetwork $\sigma_c \in \Sigma(S)$ which includes the edge (q,i) (fig. 3.1.3, right). We use an index $c=\{1,2,...,C\}$ to enumerate elements of $\Sigma(S)$ for convenience, since we will later enumerate over subnetworks.

Remembering that σ_c is a tree, we divide σ_c in three disjoint subgraphs: the edge (q,i), the tree $\sigma^{d(i)}_{h(c)}$ corresponding to "descendants" of i, and the remaining tree $\sigma^{a(q)}_{g(c)}$. Notice that $g(c)$ could be the same for two different subnetworks $\sigma_1,\sigma_2\in\Sigma(S)$, meaning that the subtree $\sigma^{a(q)}_{g(c)}$ is in common (similarly for $\sigma^{d(i)}_{h(c)}$).

We now observe that the coefficient λ_c and component P_c corresponding to σ_c (Table 3.1.1) factorize in terms corresponding to $\sigma^{a(q)}_{g(c)}$ and to $\sigma^{d(i)}_{h(c)}$ as follows: $\lambda_c = w^q_i \lambda^{d(i)}_{h(c)} \lambda^{a(q)}_{g(c)}$ and $P_c = P^{d(i)}_{h(c)} P^{a(q)}_{g(c)}$, where $\lambda^{d(i)}_{h(c)} = \prod_{(m,n)\in\mathcal{L}\left(\sigma^{d(i)}_{h(c)}\right)} w^m_n$, $P^{d(i)}_{h(c)} = \prod_{l\in\mathcal{L}\left(\sigma^{d(i)}_{h(c)}\right)} \varphi_l$ and similarly for $a(q)$. With this notation, for *each* subnetwork σ_c including (q,i) we write:

$$\lambda_c P_c = w^q_i \left(\lambda^{a(q)}_{g(c)} P^{a(q)}_{g(c)}\right)\left(\lambda^{d(i)}_{h(c)} P^{d(i)}_{h(c)}\right). \tag{A.2.1}$$

Let us now consider the sum over *all* the subnetworks σ_c of S that include (q,i). The sum can be rewritten as two nested sums, the external one over all terms $\sigma^{a(q)}_g$ (red part, fig. 3.1.3) and the internal one over all subnets $\sigma^{d(i)}_h$ (blue part, fig. 3.1.3). This is intuitively easy to grasp: we can think of the sum over all trees σ_c as first keeping the subtree $\sigma^{a(q)}_g$ fixed and varying all possible subtrees $\sigma^{d(i)}_h$ below i (inner sum), then iterating this for choice of $\sigma^{a(q)}_g$ (outer sum). Exploiting the factorization A.2.1 we obtain the following formulation:

$$\sum_{c:(q,i)\in\mathcal{E}(\sigma_c)} \lambda_c P_c = w^q_i \sum_{g=1}^{C_{a(q)}} \lambda^{a(q)}_g P^{a(q)}_g \sum_{h=1}^{C_{d(i)}} \lambda^{d(i)}_h P^{d(i)}_h. \tag{A.2.2}$$

where $C_{d(i)}$ and $C_{a(q)}$ denote the total number of different trees $\sigma^{d(i)}_h$ and $\sigma^{a(q)}_g$ in S.

Lemma A.2.1. $\frac{\partial S(X)}{\partial S_q} = \sum_{g=1}^{C_{a(q)}} \lambda^{a(q)}_g P^{a(q)}_g$.

Proof. First let us separate the sum in Eq. 3.1.6 in two sums, one over subnetworks including q and one over subnetworks not including q: $S(X) = \sum_{k:q\in\sigma_k} \lambda_k P_k + \sum_{l:q\notin\sigma_l} \lambda_l P_l$. The second sum does not involve S_q so for $\frac{\partial S(X)}{\partial S_q}$ it is a constant $\hat{k}$. Then, $S = \sum_{k:q\in\sigma_k} \lambda_k P_k + \hat{k}$. As in Eq. A.2.2, we divide the sum $\sum_{k:q\in\sigma_k}(\cdot)$ in two nested sums

acting over disjoint terms: $S = \left(\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)}\right) \left(\sum_{h=1}^{C_{d(q)}} \lambda_h^{d(q)} P_h^{d(q)}\right) + \hat{k}$. We now notice that $\sum_{k=1}^{C_{d(q)}} \lambda_k^{d(q)} P_k^{d(q)} = S_q$ by Eq. 3.1.6, since $\lambda_k^{d(q)} P_k^{d(q)}$ refer to the subtree of σ_c rooted in i and the sum is taken over all such subtrees. Therefore: $S = \left(\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)}\right) S_q + \hat{k}$. Taking the partial derivative leads to the result. □

A.2.1. Proof of Proposition 3.1.7

We start by writing the sum on the left-hand side of Eq. 3.1.11 as in Eq. A.2.2. Now, first we notice that $\sum_{k=1}^{C_{d(i)}} \lambda_k^{d(i)} P_k^{d(i)}$ equals $S_i(X)$ by Eq. 3.1.6, since $\lambda_k^{d(i)} P_k^{d(i)}$ refer to the subtree of σ_c rooted in i and the sum is taken over all such subtrees. Second, $\sum_{g=1}^{C_{a(q)}} \lambda_g^{a(q)} P_g^{a(q)} = \frac{\partial S(X)}{\partial S_q}$ for Lemma A.2.1. Substituting in A.2.2 we get the result. □

A.2.2. EM step on W

Starting from eq. 3.3.1 and collecting terms not depending on W in a constant, we obtain:

$$Q(W) = \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \ln \lambda_c(W) + \text{const} = \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{(q,i)\in\mathcal{E}(\sigma_c)} \ln w_i^q + \text{const}.$$

We now drop the constant and move out $\sum_{(q,i)\in\mathcal{E}(\sigma_c)}$ by introducing a delta $\delta_{(q,i),c}$ which equals 1 if $(q,i) \in \mathcal{E}(\sigma_c)$ and 0 otherwise and summing over *all* edges $\mathcal{E}(S)$:

$$\begin{aligned} Q(W) &= \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{(q,i)\in\mathcal{E}(S)} \ln w_i^q \delta_{(q,i),c} = \sum_{(q,i)\in\mathcal{E}(S)} \sum_{n=1}^{N} \frac{\sum_{c=1}^{C} \lambda_c P_c(x_n) \delta_{(q,i),c}}{S(x_n)} \ln w_i^q \\ &= \sum_{(q,i)\in\mathcal{E}(S)} \sum_{n=1}^{N} \frac{\sum_{c:(q,i)\in\mathcal{E}(\sigma_c)} \lambda_c P_c(x_n)}{S(x_n)} \ln w_i^q. \end{aligned}$$

Applying Lemma 3.1.7 we get: $Q(W) = \sum_{(q,i)\in\mathcal{E}(S)} \left(\sum_{n=1}^{N} \frac{w_{i,old}^q \frac{\partial S(x_n)}{\partial S_q} S_i(x_n)}{S(x_n)}\right) \ln w_i^q$, and defining $\beta_i^q = w_{i,old}^q \sum_{n=1}^{N} S^{-1}(x_n) \frac{\partial S(x_n)}{\partial S_q} S_i(x_n)$ we write $Q(W) = \sum_{q\in\mathcal{N}(S)} \sum_{i\in ch(q)} \beta_i^q \ln w_i^q$. □

A.2.3. EM step on θ

Starting from Eq. 3.3.1, as in A.2.2 we expand $\ln P_c$ as a sum of logarithms and obtain: $Q(\theta) = \sum_{n=1}^{N} \sum_{c=1}^{C} \frac{\lambda_c P_c(x_n)}{S(x_n)} \sum_{l\in\mathcal{L}(\sigma_c)} \ln \varphi_l(x_n|\theta_l) + const$. Introducing $\delta_{l,c}$ which equals 1 if $l \in \mathcal{L}(\sigma_c)$ and 0 otherwise, dropping the constant and performing the sum $\sum_{l\in\mathcal{L}(S)}$

over all leaves in S we get:

$$Q(\theta) = \sum_{n=1}^{N}\sum_{c=1}^{C}\frac{\lambda_c P_c(x_n)}{S(x_n)}\sum_{l\in\mathcal{L}(S)}\ln\varphi_l(x_n|\theta_l)\,\delta_{l,c} = \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\frac{\sum_{c=1}^{C}\lambda_c P_c(x_n)}{S(x_n)}\ln\varphi_l(x_n|\theta_l)\,\delta_{l,c}$$
$$= \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\frac{\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c(x_n)}{S(x_n)}\ln\varphi_l(x_n|\theta_l) = \sum_{l\in\mathcal{L}(S)}\sum_{n=1}^{N}\alpha_{ln}\ln\varphi_l(x_n|\theta_l).$$

Where $\alpha_{ln} = S(x_n)^{-1}\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c(x_n)$. To compute α_{ln} we notice that the term P_c in this sum always contains a factor φ_l (corresponding to a weight in Eq. 3.1.7), and $\varphi_l = S_l$ by def. 3.1.3. Then, writing $P_{c\setminus l} = \left(\prod_{k\in\mathcal{L}(\sigma_c)\setminus l}\varphi_k\right)$ we obtain: $\alpha_{ln} = S(x_n)^{-1}S_l\left(\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l}(x_n)\right)$. Finally, since $S = \sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_c + \sum_{k:l\notin\mathcal{L}(\sigma_k)}\lambda_k P_k = S_l\sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l} + \hat{k}$ (where $\hat{k}$ does not depend on S_l), taking the derivative we get $\frac{\partial S}{\partial S_l} = \sum_{c:l\in\mathcal{L}(\sigma_c)}\lambda_c P_{c\setminus l}$. Substituting we get: $\alpha_{ln} = S(x_n)^{-1}\frac{\partial S(X)}{\partial S_l}S_l(x_n)$. □

A.3. Sum-Product Graphical Models

A.3.1. Proof of Proposition 4.1.2.

Consider a subtree $\tau \in \mathcal{T}(\mathbb{S})$ as in Definition 4.1.7.

1. Let us first consider messages generated by sum nodes s. Considering that s has only one child $ch(s)$ in τ (for Definition 4.1.7), corresponding to indicator variable $[Z_s]_{k_s}$, and applying Eqs. 4.1.1, the messages for observed and unobserved sum nodes are as follows:

$$\mu_{st;j} = [Z_s]_{k_s}Q_s(k_s)\mu_{ch(s),t;j}, \quad \text{Observed Sum Node,} \tag{A.3.1}$$
$$\mu_{st;j} = W_s(k_s)\mu_{ch(s)_k,t;j}, \quad \text{Unobserved Sum Node.} \tag{A.3.2}$$

   Hence sum messages contribute only by introducing a multiplicative term $[Z_s]_{k_s}Q_s(k_s)$ or $W_s(k_s)$. Now, all variables in the set Z appear in τ (Proposition 4.1.4). Hence, the sum nodes together contribute with the following multiplicative term:

$$\prod_{s\in O(\tau)}Q_s(k_{s,\tau})\prod_{s\in U(\tau)}W_s(k_{s,\tau})\prod_{[Z_s]_{k_s}\in z_\tau}[Z_s]_{k_s} = \lambda_\tau\prod_{[Z_s]_{k_s}\in z_\tau}[Z_s]_{k_s}. \tag{A.3.3}$$

   From this it follows that message passing in τ is equivalent to message passing in a SPGM $\overline{\tau}$ obtained by *discarding* all the sum nodes from τ, followed by multiplying the resulting messages for Eq. A.3.3.

2. Consecutive Product Nodes in $\overline{\tau}$ can be merged by adding the respective children to the parent Product Node. In addition, between each sequence Vnode-Vnode in $\overline{\tau}$ we can insert a product node with a single child. Thus, we can take $\overline{\tau}$ as

containing only Vnodes and Product Nodes, such that the children of Product Nodes are Vnodes. Putting together Eqs. 4.1.1c and 4.1.1d, the message passed by each Vnode t to $s \in vpa(t)$ is:

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)\left[X_t\right]_k \prod_{q\in ch(ch(c))} \mu_{q\to t;j}. \tag{A.3.4}$$

Notice that the input messages are generated from the grandchildren of t, that is the children of the Product Node child of t. This corresponds to the message passed by variables in a tree graphical model obtained by removing the Product Nodes from $\overline{\tau}$ and attaching the children of Product Nodes (here, elements $q \in ch\left(ch(c)\right)$) as children of their parent Vnode (here, t), which can be seen by noticing the equivalence to Eq. (2.3.6). This tree GM can be immediately identified as $P_\tau(X)$. Note also that if the root of $\overline{\tau}$ is a Product Node, then $P_\tau(X)$ represents a forest of trees, one for each child of the root Product Node.

3. The proof is concluded reintroducing the multiplicative factor in Eq. A.3.3 discarded when passing from τ to $\overline{\tau}$.

□

A.3.2. Proof of Proposition 4.1.3.

The proposition can be proven by inspection, considering a SPGM $\mathbb{S}$ built by stacking units as follows: Sum Node s_1 (the root of $\mathbb{S}$) is associated to observed variable Z_1 and has as children the set of Vnodes $V_1 = \{v_{1,1}, v_{1,2}, \ldots, v_{1,M}\}$. Each node in V_1 has as child the same Sum Node s_2. In turn, s_2 has the same structure of s_1, having Vnode children $V_1 = \{v_{2,1}, v_{2,2}, \ldots, v_{2,M}\}$, and so forth for Sum Nodes $s_3, s_4, \ldots, s_K$. The number of edges in $\mathbb{S}$ is $2MK$. On the other hand, a different subnetwork can be obtained for each choice of active children at each sum node. There is a combinatorial number of such choices, and the number of different subnetworks is K^M. □

A.3.3. Proof of Proposition 4.1.5.

First, consider a SPGM $\mathbb{S}(X, Z)$ defined over $\mathcal{G} = \mathcal{V}, \mathcal{E}$. Let us take a node $t \in \mathcal{V}$ and let $\mathbb{S}_t(X^t, Z^t)$ denote the sub-SPGM rooted in t. Suppose that $\mathbb{S}_t$ satisfies the proposition and hence it can be written as:

$$\mathbb{S}_t = \sum_{\tau_t\in\mathcal{T}(\mathbb{S}_t)} \tau_t. \tag{A.3.5}$$

Let us consider messages sent from node t to a Vparent $s \in vpa(t)$ (Definition 4.1.3), and note that if form (A.3.5) is satisfied then messages take the form

$$\mu_{t\to s;j} = \sum_{\tau_t\in\mathcal{T}(\mathbb{S}_t)} \mu_{root(\tau_t)\to s;j}, \tag{A.3.6}$$

where $root(\tau_t)$) denotes the message sent from the root of subtree τ_t to s. Vice versa, if form (A.3.6) is satisfied then $\mathbb{S}_t$ can be written as Eq. (A.3.5). This extends also to the case $vpa(t) = \emptyset$, in which messages are sent to a fictitious *root* node (Definition 4.1.5).

The proposition can now be proved by induction. First, the base case: sub-SPGMs rooted at Vnodes leaves trivially assume form (A.3.5), and hence also the form (A.3.6). Then, the inductive step: Lemma (A.3.1) can be applied recursively for all nodes from the leaves to the root. Hence, $\mathbb{S}$ assumes the form of Eq. (A.3.6) and thus also of (A.3.5). □

Lemma A.3.1. *Consider a node $t \in \mathcal{V}$. Suppose that the messages sent from the children of node $t \in \mathcal{V}$ are in the form* (A.3.6)*. Then, each message sent from s also assumes the form A.3.6.*

Let us suppose, for simplicity, t has only two children p and q - the extension to the general case is straightforward. We distinguish the three cases of t being a Sum Node, a Product Node or a Vnode.

- Suppose t is an Observed Sum Node with weights $[Q_t(0), Q_t(1)]$ and indicator variables $[Z_t]_0$ and $[Z_t]_1$ associated to each child. By Eq. (A.3.2), the children send messages to their Vparent s, and since input message are in the form (A.3.6), t sends the following message:

$$\mu_{t\to s;j} = Q_t(0)[Z_t]_0 \sum_{\tau_p \in \mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p)\to s;j} + Q_t(1)[Z_t]_1 \sum_{\tau_q \in \mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_q)\to s;j}.$$

  This is again in the form A.3.6. This can be seen because each term in the sum is in the form $Q_t(0)[Z_t]_0\mathcal{T}(\mathbb{S}_p)$ which corresponds to message passed from the subtree $\tau_t \in \mathcal{T}(\mathbb{S}_t)$ obtained choosing child p (in this case). It is easy to see that terms corresponding to each subtrees of t are present.

- If t is an Unobserved Sum Node the discussion is identical to the point above.

- If t is a Product Node, then the children send messages to their Vparent s, and since input message are in the form (A.3.6), t sends the following message:

$$\begin{aligned}\mu_{t\to s;j} &= \Big(\sum_{\tau_p \in \mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p)\to s;j}\Big)\Big(\sum_{\tau_q \in \mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_q)\to s;j}\Big) \\ &= \sum_{(\tau_p,\tau_q)\in \mathcal{T}(\mathbb{S}_p)\times\mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_p)\to s;j}\mu_{root(\tau_q)\to s;j}.\end{aligned}$$

  Now, $\mu_{root(\tau_p)\to s;j}\mu_{root(\tau_q)\to s;j}$ can be seen as the message generated by a particular subtrees of t, and thus node $\mu_{t\to s;j}$ is in the form (A.3.6).

- If t is a Vnode, then it has a *single child* p by definition 4.1.4, sending messages to t.

Thus, the input message is in the form (A.3.6), and t sends the following message:

$$\begin{aligned}\mu_{t\to s;j} &= \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)\left[X_t\right]_k \sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p)\to t;k} \\ &= \sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)} \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)\left[X_t\right]_k \mu_{root(\tau_p)\to t;k}.\end{aligned}$$

Let us analyze a term of the sum for each fixed τ_p. Such term corresponds to the root message of a SPGM $\overline{\mathbb{S}}$ obtained taking τ_p and adding the Vnode t as parent of p (due to Eq. (4.1.1d)). But $\overline{\mathbb{S}}$ obtained in this form is a subtree of t by Definition 4.1.7. Therefore, taking the sum $\sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)}$ corresponds to summing over messages sent by all subnetworks of t, in the form (A.3.6). □

A.3.4. Proof of Proposition 4.1.9

The proposition can be proven by induction showing that if input messages represent valid SPNs, then also the output SPGM messages are SPNs (see Fig. 4.1.2). Formally, it is sufficient to notice that the hypothesis of Lemma A.3.2 below is trivially true for leaf messages (inductive hypothesis), hence the lemma can be inductively applied for all nodes up to the root. □

Lemma A.3.2. *Consider nodes $t \in \mathcal{V}$ and $s \in vpa(t)$, and let X^t and Z^t respectively denote the X scope and Z scope of node $t \in \mathcal{V}$. If the message $\mu_{t\to s;j}$ encodes a SPN $S_{t;j}(X^t, Z^t)$, then the message $\mu_{s\to r;i}$ sent from node s to any node $r \in vpa(s)$ also implements a SPN $S_{r;i}(X^s, Z^s)$. This also holds when $vpa(s) = \emptyset$, where r is simply replaced by the fictitious root node (Definition 4.1.5.*

Proof. Consider the message passing Eqs. 4.1.1, referring to Fig. 4.1.2 for visualization. We distinguish the case of node t being a Sum, Product and Vnode.

- Sum Nodes (Observed and non-Observed). First, we note that every child of a Sum Node has the same scope X^s, Z^s (for Definition 4.1.4 condition 2). Employing the hypothesis, the generated message is:

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} [Z_t]_k Q_t(k) S_{t;j}\left(X^t, Z^t\right), \quad t \text{ is a Sum Node, Observed,} \tag{A.3.7}$$

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} W_t(k) S_{t;j}\left(X^t, Z^t\right), \quad t \text{ is a Sum Node, not-Observed} \tag{A.3.8}$$

  It is straightforward to see that both equations represent valid SPNs: the sum $\sum_{k=1}^{|ch(t)|}$ becomes the root SPN Sum Node with non-negative weights $Q_t(k)$ and $W_t(k)$ respectively, and its children are SPNs having the same scope (in the form $[Z_t]_k \otimes S_j\left(X^t, Z^t\right)$ for Eq. A.3.7 and in the form $S_j\left(X^t, Z^t\right)$ for Eq. A.3.8). Note that $Z^t \cap Z_t = \emptyset$ for Definition 4.1.4 condition 2, therefore condition 2 in Definition 3.1.3 is satisfied.

- Product Nodes. Applying the hypothesis to input messages, Eq. 4.1.1c becomes:

$$\mu_{t\to s;j} = \prod_{q\in ch(t)} S_{q;j}\left(X^q, Z^q\right).$$

  This represents a valid SPN with a Product Node as root since the children node's scopes are disjoint (for Definition 4.1.4 condition 3), and thus condition 1 in Definition 3.1.3 is satisfied.

- Vnodes. Applying the hypothesis to input messages, Eq. 4.1.1d becomes:

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)[X_t]_k S_{ch(t);k}\left(X^{ch(t)}, Z^{ch(t)}\right).$$

  This represents a valid SPN with a Sum Node $\sum_{k\in\Delta(X_t)}$ as root. To see this, first note that terms $P_{s,t}\left(k|j\right)$ can be interpreted as weights. The Sum Node is connected to children SPNs in the form $[X_t]_k \otimes S_{ch(t);k}\left(X^{ch(t)}, Z^{ch(t)}\right)$, which are valid SPNs since $X^i \cap X_s = \emptyset$ and thus condition 1 in Definition 3.1.3 is satisfied. In addition all child SPNs have the same scope for Definition 4.1.4 condition 2, hence condition 2 in Definition 3.1.3 is satisfied for the Sum Node.

□

Bibliography

Nicholas J. Adams and Christopher K.I. Williams. Dynamic trees for image modelling. *Image and Vision Computing*, 21(10):865 – 877, 2003. ISSN 0262-8856. doi: http://dx.doi.org/10.1016/S0262-8856(03)00073-8. URL http://www.sciencedirect.com/science/article/pii/S0262885603000738. 5.5, 5.5, 5.5, 5.5.1, 5.5.5, 5.5.6

Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the Structure of Sum-Product Networks via an SVD-based Algorithm. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pages 32–41, 2015. (document), 3.4.4, 3.5

S.M. Aji and R.J. McEliece. The generalized distributive law. *IEEE Trans. Information Theory*, 46(2):325–343, 2000. 2.3.3

Mohamed Amer and Sinisa Todorovic. Sum Product Networks for Activity Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI 2015)*, 2015. 1, 1.2.1, 1.4

Mohamed R. Amer and Sinisa Todorovic. Sum Product Networks for Activity Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(4):800–813, 2016. doi: 10.1109/TPAMI.2015.2465955. URL http://dx.doi.org/10.1109/TPAMI.2015.2465955. 3.5, 4.2.1

B. Andres, Beier T., and J. H. Kappes. OpenGM: A C++ Library for Discrete Graphical Models. *ArXiv e-prints*, 2012. 5.5

Robert Ash and Catherine Doleans-Dade. *Probability and Measure Theory*. Academic Press, 2000. 2.1, 2.1.2

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Value Elimination: Bayesian Inference via Backtracking Search. *CoRR*, abs/1212.2452, 2012. 1.4.1, 1.4

Francis R. Bach and Michael I. Jordan. Thin Junction Trees. In *Advances in Neural Information Processing Systems 14*, pages 569–576. MIT Press, 2001. 5.4

J. M. Beaulieu and M. Goldberg. Hierarchy in picture segmentation: a stepwise optimization approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):150–163, Feb 1989. ISSN 0162-8828. doi: 10.1109/34.16711. 5.5

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-Specific Independence in Bayesian Networks. pages 115–123, 1996. 1.1, 1.4.1, 1.4, 2.5

Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Chai. Language Modeling with Sum-Product Networks. *Annual Conference of the International Speech Communication Association 15 (INTERSPEECH 2014)*, 2014. 1, 3.3.3, 3.5, 4.2.1

David Maxwell Chickering. *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer-Verlag, January 1996. URL https://www.microsoft.com/en-us/research/publication/learning-bayesian-networks-is-np-complete/. 2.3.4

David Maxwell Chickering, David Heckerman, and Christopher Meek. A Bayesian Approach to Learning Bayesian Networks with Local Structure. *CoRR*, abs/1302.1528, 2013. 1.4.1, 1.4

Stephanie J Chiu, Michael J Allingham, Priyatham S Mettu, Scott W Cousins, Joseph A Izatt, and Sina Farsiu. Kernel regression based segmentation of optical coherence tomography images with diabetic macular edema. *Biomed. Opt. Express*, 6(4):1172–1194, 2015. 5.3.1, 5.3.4

Myung Jin Choi, Vincent Y. F. Tan, Animashree Anandkumar, and Alan S. Willsky. Learning Latent Tree Graphical Models. *J. Mach. Learn. Res.*, 12:1771–1812, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021056. 3.4.1, 5.2

C. I. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968. 1, 1.3, 1.5.2, 2.3.4, 3.3.4, 3.4.1, 4.2.1, 5.4

Microsoft Corporation. Winmine toolkit. URL https://www.microsoft.com/en-us/download/details.aspx?id=52333. 3.4.1

R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 2003. 1.2.2, 2, 2.3.1, 4.1.3

Matthew Crouse, Robert Nowak, and Richard Baraniuk. Wavelet-Based Statistical Signal Processing Using Hidden Markov Models, 1998. 5, 5.5

Madeleine Cule, Richard Samworth, and Michael Stewart. Maximum likelihood estimation of a multi-dimensional log-concave density. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(5):545–607, 2010. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2010.00753.x. URL http://dx.doi.org/10.1111/j.1467-9868.2010.00753.x. 3.3.5

A. Darwiche. A Differential Approach to Inference in Bayesian Networks. *J. ACM*, 50 (3):280–305, 2003. 1.1.1, 3.2.3, 4.1.4

Adnan Darwiche. A Logical Approach to Factoring Belief Networks. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *KR*,

pages 409–420. Morgan Kaufmann, 2002. ISBN 1-55860-554-1. 1, 1.1, 1.4.1, 1.4, 3.2.2, 4.2.1

Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2 - 3):73 – 106, 2007. ISSN 0004-3702. 1, 1.1, 1.4.1, 1.4, 3.2.2

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. doi: 10.2307/2984875. 4.2.1

Aaron Dennis and Dan Ventura. Greedy Structure Search for Sum-product Networks. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 932–938. AAAI Press, 2015. ISBN 978-1-57735-738-4. 3.1.1

Mattia Desana and Christoph Schnörr. Expectation Maximization for Sum-Product Networks as Exponential Family Mixture Models. *CoRR*, abs/1604.07243, 2016. URL http://arxiv.org/abs/1604.07243. 3.1.1, 4.2.1, 4.2.1, 4.2.3

R. Diestel. *Graph Theory*. Springer, 3rd edition, 2006. 1.1, 2.3.1

Xiaojuan Feng, C. K. I. Williams, and S. N. Felderhof. Combining belief networks and neural networks for scene segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):467–483, Apr 2002. ISSN 0162-8828. doi: 10.1109/34.993555. 5.5

Arthur Fridman. Mixed Markov models. *PNAS*, page 100, 2003. ISSN 8092-8096. 1.4.1, 1.4, 4.1.3

Stuart Geman and Chii-Ruey Hwang. Nonparametric maximum likelihood estimation by the method of sieves. *The Annals of Statistics*, 10:401–414, 1982. 3.3.5

Robert Gens and Pedro Domingos. Discriminative Learning of Sum-Product Networks. In *NIPS*, pages 3248–3256, 2012. 3.1.1, 3.3.1, 3.3.1, 3.5, 4.1.3, 4.2.1, 4.2.2

Robert Gens and Pedro Domingos. Learning the Structure of Sum-Product Networks. In *ICML (3)*, pages 873–880, 2013. (document), 1.1, 1.3, 1.4, 3.4, 3.4, 1, 3.4.2, 3.4.2, 3.5, 3.4.1, 5.1, 5.1, 5.2

Vibhav Gogate, William Webb, and Pedro Domingos. Learning Efficient Markov Networks. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 748–756. Curran Associates, Inc., 2010. 1.4.1, 1.4

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. *Measuring Statistical Dependence with Hilbert-Schmidt Norms*, pages 63–77. Springer Berlin

Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31696-1. doi: 10.1007/11564089_7. URL http://dx.doi.org/10.1007/11564089_7. 3.4.4

Xuming He, Richard S. Zemel, and Miguel Á. Carreira-Perpiñán. Multiscale Conditional Random Fields for Image Labeling. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 695–703, Washington, DC, USA, 2004. IEEE Computer Society. URL http://dl.acm.org/citation.cfm?id=1896300.1896400. 5.5

Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006. 5.4

Michael I. Jordan. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994. 1.4.1, 1.4, 1.4.1, 4.1.3

SPK Karri, Debjani Chakraborthi, and Jyotirmoy Chatterjee. Learning layer-specific edges for segmenting retinal layers with large deformations. *Biomed. Opt. Express*, 7 (7):2888–2901, 2016. 5.3.1, 5.3.4

V. Kolmogorov and R. Zabih. What Energy Functions Can Be Minimized via Graph Cuts? *IEEE Trans. Patt. Anal. Mach. Intell.*, 26(2):147–159, 2004. 1.1.1, 1.1

Vladimir Kolmogorov. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, October 2006. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.200. URL http://dx.doi.org/10.1109/TPAMI.2006.200. 6.4, 6.4

Jean-Marc Laferté, Patrick Pérez, and Fabrice Heitz. Discrete markov image modeling and inference on the quadtree. *IEEE Trans. Image Processing*, 9(3):390–404, 2000. doi: 10.1109/83.826777. URL http://dx.doi.org/10.1109/83.826777. 1.6, 5.5, 5.5

Daniel Lowd and Pedro Domingos. Learning Arithmetic Circuits. *CoRR*, abs/1206.3271, 2012. (document), 3.4.1, 3.5, 3.4.1, 5.2

George Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving (5th Edition)*. Pearson Addison Wesley, 2004. ISBN 0321263189. 3.2.1

David Mcallester, Michael Collins, and Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI 04)*, pages 382–391, 2004. 1, 1.4.1, 1.4

Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000. 1.4.1, 1.3, 1.4, 3.3.4, 3.4.1, 4.2.3, 5.2, 5.4, 5.4

Mazen Melibari, Pascal Poupart, and Prashant Doshi. Dynamic Sum Product Networks for Tractable Inference on Sequence Data. *CoRR*, abs/1511.04412, 2015. URL http://arxiv.org/abs/1511.04412. 3.5

Talya Meltzer, Amir Globerson, and Yair Weiss. Convergent Message Passing Algorithms: A Unifying View. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 393–401, Arlington, Virginia, United States, 2009. AUAI Press. ISBN 978-0-9749039-5-8. URL http://dl.acm.org/citation.cfm?id=1795114.1795160. 6.4

Tom Minka and John Winn. Gates. In *Advances in Neural Information Processing Systems 21*, 2009. 1.4.1, 1.4, 4.1.3

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029. 2.1.3, 2.4.1, 2.4.3, 3.2.3, 3.3.1, 3.3.3, 3.3.4

Radford Neal and Geoffrey E. Hinton. A View Of The Em Algorithm That Justifies Incremental, Sparse, And Other Variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998. 3.3.5, 3.4.2, 4.2.4

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000. ISBN 0-521-77362-8. 1, 2.3.1, 3.1.1, 3.1.2, 5.4, 5.5

Robert Peharz. Foundations of Sum-Product Networks for Probabilistic Modeling. (PhD thesis). *Researchgate:273000973*, 2015. 3, 7, 3.1.3, 3.1.3

Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling Speech with Sum-Product Networks: Application to Bandwidth Extension. In *ICASSP*, pages 3699 – 3703, 2014. 3.5

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M. Domingos. On the Latent Variable Interpretation in Sum-Product Networks. *CoRR*, abs/1601.06180, 2016. URL http://arxiv.org/abs/1601.06180. 1.1, 1.4, 1.5.1, 3.3.2, 3.3.3

Patrick Pletscher, Cheng Soon Ong, and Joachim M. Buhmann. Spanning Tree Approximations for Conditional Random Fields. In David A. Van Dyk and Max Welling, editors, *AISTATS*, volume 5 of *JMLR Proceedings*, pages 408–415. JMLR.org, 2009. 5.4

David L. Poole and Nevin Lianwen Zhang. Exploiting Contextual Independence In Probabilistic Inference. *CoRR*, abs/1106.4864, 2011. 1.4.1

Hoifung Poon and Pedro Domingos. Sum-Product Networks: A New Deep Architecture. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 337–346, 2011. 1, 1.1, 1.4.1, 3.1.1, 3.2.2, 3.3.1, 3.3.2, 3.5, 4.2.1, 5.1, 5.2

Tahrima Rahman and Vibhav Gogate. Learning Ensembles of Cutset Networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3301–3307, 2016a. URL http://www.

aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12503. 1, 1.1, 1.3, 3.4, 3.4.5, 3.5, 3.4.1, 5.2

Tahrima Rahman and Vibhav Gogate. Merging Strategies for Sum-Product Networks: From Trees to Graphs. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*, 2016b. URL http://auai.org/uai2016/proceedings/papers/71.pdf. (document), 1, 1.1, 1.3, 3.4, 3.4.6, 3.4.6, 3.5, 3.4.1, 4.2.1, 5.2

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*, pages 630–645, 2014. doi: 10.1007/978-3-662-44851-9_40. URL http://dx.doi.org/10.1007/978-3-662-44851-9_40. (document), 1.2.1, 1.4.1, 1.4, 3.2.2, 3.4.5, 3.5, 3.4.1, 5.2

Fabian Rathke, Stefan Schmidt, and Christoph Schnörr. Probabilistic intra-retinal layer segmentation in 3-D OCT images using global shape regularization. *Med. Image Anal.*, 18(5):781–794, 2014. 1.5.1, 5, 5.3.1, 5.3, 5.3.1, 5.3.2, 5.3.2, 5.3.2, 5.3.2, 5.3.6

Amirmohammad Rooshenas and Daniel Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 710–718. JMLR Workshop and Conference Proceedings, 2014. (document), 1, 1.1, 1.3, 1.4, 3.4, 3.4.3, 3.4.1, 5.2

Jing Tian, Boglarka Varga, Erika Tatrai, Palya Fanni, Gabor Mark Somfai, William E. Smiddy, and Delia Cabrera Debuc. Performance evaluation of automated segmentation software on optical coherence tomography volume data. *J. Biophotonics*, 9(5):478–489, 2016. ISSN 1864-0648. doi: 10.1002/jbio.201500239. URL http://dx.doi.org/10.1002/jbio.201500239. 5.3.1, 5.3.4

M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 49(5):1120–1146, May 2003. ISSN 0018-9448. doi: 10.1109/TIT.2003.810642. 6.4

Martin Wainwright, Tommi Jaakkola, and Alan Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. *IEEE Transactions on Information Theory*, 51:3697–3717, 2002. 1.6, 6.4

Martin J. Wainwright and Michael I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, January 2008. ISSN 1935-8237. doi: 10.1561/2200000001. 1, 3.3.5, 5.4

Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning Fine-Grained Image Similarity with Deep Ranking. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1386–1393, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.180. URL http://dx.doi.org/10.1109/CVPR.2014.180. 3.5

Han Zhao, Mazen Melibari, and Pascal Poupart. On the Relationship between Sum-Product Networks and Bayesian Networks. *CoRR*, abs/1501.01239, 2015. 1.4.1

Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed Variational Inference for Sum-Product Networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1310–1318, 2016a. URL http://jmlr.org/proceedings/papers/v48/zhaoa16.html. 5.1, 5.1

Han Zhao, Pascal Poupart, and G. Gordon. A Unified Approach for Learning the Parameters of Sum-Product Networks. *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS 2016)*, 2016b. 1.4, 3.1.1, 3.1.1, 3.1.1, 3.1.1, 3.3.2, 3.3.3, 3.4.3, 4.1.3, 5.1, 5.1