Graz University of Technology

PhD Thesis

Foundations of Sum-Product Networks for Probabilistic Modeling

Dipl.-Ing. Robert Peharz

DOCTORAL THESIS
to achieve the university degree of
Doktor der technischen Wissenschaften
submitted to

Graz University of Technology
Austria

Supervisor:
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf
Signal Processing and Speech Communication Laboratory
Graz University of Technology

Examination Committee:
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf
Signal Processing and Speech Communication Laboratory
Graz University of Technology

Assoc.Prof. Dr.habil. Manfred Jaeger
Department for Computer Science
Aalborg University

Graz, February 2015

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

date	(signature)

To Benjamin and Elisabeth :)

Abstract

Sum-product networks (SPNs) are a promising and novel type of probabilistic model, which has been receiving significant attention in recent years. There are, however, several open questions regarding the foundations of SPNs and also some misconceptions in literature. In this thesis we provide new theoretical insights and aim to establish a more coherent picture of the principles of SPNs.

As a first main contribution we show that, without loss of generality, SPN parameters can be assumed to be locally normalized, i.e. normalized for each individual sum node. Furthermore, we provide an algorithm which transforms unnormalized SPN-weights into locally normalized weights, without changing the represented distribution.

Our second main contribution is concerned with the two notions of consistency and decomposability. Consistency, together with completeness, is required to enable efficient inference in SPNs over random variables with finitely many states. Instead of consistency, usually the conceptually easier and stronger condition of decomposability is used. As our second main contribution we show that consistent SPNs can not represent distributions exponentially more compactly than decomposable SPNs. Furthermore, we point out that consistent SPNs are not amenable for Darwiche's differential approach to inference.

SPNs were originally defined over random variables with finitely many states, but can be easily generalized to SPNs over random variables with infinitely many states. As a third main contribution, we formally derive two inference mechanisms for generalized SPNs, which so far have been derived only for finite-state SPNs. These two inference scenarios are marginalization and the so-called evaluation of modified evidence.

Our fourth main contribution is to make the latent variable interpretation of sum nodes in SPNs more precise. We point out a weak point about this interpretation in literature and propose a remedy for it. Furthermore, using the latent variable interpretation, we concretize the inference task of finding the most probable explanation (MPE) in the corresponding latent variable model. We show that the MPE inference algorithm proposed in literature suffers from a phenomenon which we call low-depth bias. We propose a corrected algorithm and show that it indeed delivers an MPE solution. While an MPE solution can be efficiently found in the augmented latent variable model, we show that MPE inference in SPNs is generally NP-hard.

As another contribution, we point out that the EM algorithm for SPN-weights presented in literature is incorrect, and propose a corrected version.

Furthermore, we discuss some learning approaches for SPNs and a rather powerful sub-class of SPNs, called selective SPNs, for which maximum-likelihood parameters can be found in closed form.

Acknowledgements

I would like to thank my advisor Franz Pernkopf for putting his trust in me and guiding my academic development during the last 7 years. I would also like to thank my colleagues Sebastian Tschiatschek, Michi Wohlmayr, Bernhard Geiger, Georg Kapeller, Pejman Mowlaee, Matthias Zöhrer and Michael Stark for many nice discussions and the work we did together.

Many thanks to Pedro Domingos for giving an inspiring talk at the "Inferning" workshop at ICML 2012 in Edinburgh, and for hosting me during my internship at the University of Washington, Seattle. Also many thanks to my colleague Robert Gens for some nice and selective work we did.

Furthermore many thanks to my office mates, Christina, Thomas, Barbara, Manfred, Andreas, Martin, Yamilla, Elmar and Christian, and all other colleagues from SPSC, for having a nice time during the last 5 years and for providing an inspiring and scientific atmosphere.

Contents

Nomenclature

$\mathbb{1}(\cdot)$	Indicator function, page 31
Ω	Sample space, page 25
ω	Elementary event, page 25
A, B	Events, page 25
P	Probability measure, probability distribution, page 26
$P(A \mid B)$	Conditional probability, page 29
$\mathcal{A}$	Sigma algebra, page 27
$(\Omega, \mathcal{A}, P)$	Probability space, page 27
$(\Omega, \mathcal{A})$	Measurable space, page 28
$(\Omega, \mathcal{A}, \mu)$	Measure space, page 28
$\mathcal{F}_H$	Half-open hypercubes, page 28
$\sigma(\cdot)$	Generated sigma-algebra, page 28
$\mathcal{B}$	Borel sets, page 28
$\mathscr{L}$	Lebesque-Borel measure, page 29
X, Y, Z	Random Variables (RVs), page 30
x, y, z	Values of random variables X, Y and Z, respectively, page 30
$\mathbf{val}(X)$	Image of random variable X, i.e. set of assumed values, page 30
$\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$	Sets of random variables, page 32
$\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$	Elements from $\mathbf{val}(\mathbf{X})$, $\mathbf{val}(\mathbf{Y})$, $\mathbf{val}(\mathbf{Z})$, respectively, page 32
$\mathbf{x}[\mathbf{Y}]$, $\mathbf{x}[Y]$	Projection of $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ onto $\mathbf{Y} \subseteq \mathbf{X}$ and $Y \in \mathbf{X}$, respectively, page 32
$\mathbf{val}(\mathbf{X})$	$:= \times_{X \in \mathbf{X}} \mathbf{val}(X)$, page 32
P_X	Distribution (law) of random variable X, page 30
F_X	Cumulative distribution function of random variable X, page 30
p_X	Probability mass function (PMF) of discrete random variable X, page 31
	also: Probability density function (PDF) of continuous random variable X, page 31
$P_{\mathbf{X}}$	Joint probability distribution of random variables $\mathbf{X}$, page 32

$F_{\mathbf{X}}$	Joint cumulative distribution function of random variables $\mathbf{X}$, page 32
$p_{\mathbf{X}}$	Joint probability mass function of discrete random variables $\mathbf{X}$, page 32
	also: Joint probability density function of continuous random variables $\mathbf{X}$, page 33
	also: Mixed distribution function of random variables $\mathbf{X}$, page 35
p	Shorthand for (joint) PMF, PDF or mixed distribution function, page 35
$\mathcal{N}(x \mid \mu, \sigma)$	Gaussian PDF with mean μ and standard deviation σ, page 35
$\mathbb{E}[\cdot]$	Expected value, page 34
$p_{\mathbf{Y} \mid \mathbf{Z}}$	Conditional distribution for $\mathbf{Y}$, $\mathbf{Z}$, page 34
$\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}$	Conditional independence of $\mathbf{X}$, $\mathbf{Y}$ given $\mathbf{Z}$, page 35
$\mathcal{G}$	Graph, directed or undirected, page 36
$\mathbf{V}$	Vertices of a graph, page 36
$\mathbf{E}$	Edges of a graph, page 36
$(V_i - V_j)$	Undirected edge between vertices V_i and V_j, page 36
$(V_i \to V_j)$	Directed edge from vertex V_i to vertex V_j, page 36
$\mathbf{pa}_{\mathcal{G}}(V)$	Set of parents of vertex V in directed graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
$\mathbf{ch}_{\mathcal{G}}(V)$	Set of children of vertex V in directed graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
$\mathbf{nb}_{\mathcal{G}}(V)$	Set of neighbors of vertex V in undirected graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
Π	Path, trail, page 37
$\mathbf{anc}_{\mathcal{G}}(V)$	Set of ancestors of vertex V in acyclic directed graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
$\mathbf{desc}_{\mathcal{G}}(V)$	Set of descendants of vertex V in acyclic directed graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
$\mathbf{ndesc}_{\mathcal{G}}(V)$	Set of nondescendants of vertex V in acyclic directed graph $\mathcal{G}$; sub-script omitted when clear from context, page 37
$\mathbf{C}$	Clique, page 38
$\mathscr{B}$	Bayesian network (BN), page 42
$p_{\mathscr{B}}(\mathbf{X})$	Bayesian network distribution, page 42
$\boldsymbol{\theta}$, $\boldsymbol{\Theta}$	Parameters of probabilistic graphical models, page 43
$\theta^{X}_{x \mid \mathbf{x}_{\mathbf{pa}}}$	Parameter for Bayesian network over finite-state random variables, page 43
$\mathscr{M}$	Markov network, page 47

Ψ	Clique factor/potential, page 47
$p_{\mathscr{M}}$	Markov network distribution, page 47
$\mathscr{Z}_{\mathscr{M}}$	Normalization constant or partition function of Markov network $\mathscr{M}$, page 47
$\mathcal{D}$	Collection of samples, page 51
$\mathcal{C}$	Model class, page 51
$\mathcal{L}$	Likelihood function, page 51
$\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$	Partial evidence for random variables X, Y, Z, respectively, page 60
$\mathcal{A}_X$	$:= \mathcal{B}(\mathbf{val}(X))$, i.e. Borel sets of $\mathbf{val}(X)$, page 60
$\mathcal{H}_{\mathbf{X}}$	Product sets of random variables $\mathbf{X}$, page 61
$\boldsymbol{\mathcal{X}}$, $\boldsymbol{\mathcal{Y}}$, $\boldsymbol{\mathcal{Z}}$	Partial evidence/elements of product sets for $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, respectively, page 61
$\boldsymbol{\mathcal{X}}[\mathbf{Y}]$, $\boldsymbol{\mathcal{X}}[Y]$	Projection of partial evidence onto $\mathbf{Y} \subseteq \mathbf{X}$ and $Y \in \mathbf{X}$, respectively, page 61
$\lambda_{X=x}$	Indicator variable (IV) for state x of a finite-state random variable X, page 61
$\boldsymbol{\lambda}$	Vector collecting all indicator variables, page 61
$f_{\mathscr{B}}$	Network polynomial of Bayesian network $\mathscr{B}$, page 62
Φ	Unnormalized distribution, page 62
f_{Φ}	Network polynomial of unnormalized distribution Φ, page 63
$\mathcal{S}$	Sum-product network (SPN), page 65
S	Sum node, page 65
P	Product node, page 65
N	Generic node in an SPN, page 65
C	Generic node in an SPN – child of other node, page 65
F	Generic node in an SPN – parent of other node, page 65
$w_{\mathsf{S},\mathsf{C}}$	Sum-weight associated with edge $(\mathsf{S} \to \mathsf{C})$ in an SPN, page 65
$\boldsymbol{w}$	Set of all sum-weights in an SPN, page 65
$\mathbf{S}(\mathcal{S})$	Set of all sum nodes in SPN $\mathcal{S}$, page 65
$\mathbf{P}(\mathcal{S})$	Set of all product nodes in SPN $\mathcal{S}$, page 65
$\boldsymbol{\lambda}(\mathcal{S})$	Set of all indicator variables in SPN $\mathcal{S}$, page 65
$\boldsymbol{\lambda}_X(\mathcal{S})$	Set of all indicator variables related to X in SPN $\mathcal{S}$, page 65
$A_{\mathcal{S}}$	Number of addition operations in SPN $\mathcal{S}$, page 66
$M_{\mathcal{S}}$	Number of multiplication operations in SPN $\mathcal{S}$, page 66
$\mathbf{sc}(\mathsf{N})$	Scope of node N, page 66
$p_{\mathcal{S}}$	SPN distribution, page 66

$\mathscr{Z}_{\mathcal{S}}$	Normalization constant of SPN $\mathcal{S}$, page 66
$\mathcal{S}_{\mathsf{N}}$	Sub-SPN rooted at node N, page 66
p_{N}	Distribution of sub-SPN $\mathcal{S}_{\mathsf{N}}$, page 66
$\mathsf{D}_{\mathbf{Y}}$	Distribution node, input distribution, page 77
f_p^e	Extended network polynomial, page 78
$\mathbf{D}^X$	Set of distribution nodes having X in their scope, page 80
$[\mathsf{D}]^X$	Index of D within $\mathbf{D}^X$, page 80
Z_X	Distribution selector of X, page 81
$\mathsf{P}^{X,[\mathsf{D}]^X}$	Gate, page 81
$\mathcal{S}^g$	Gated SPN, page 81
$\mathsf{C}_{\mathsf{S}}^k$	k^{th} child of sum node S, page 85
K_{S}	Number of children of sum node S, page 85
Z_{S}	Latent random variable associated with S, page 85
$\mathbf{Z}_{\mathbf{S}}$	Latent random variables associated with set of sum nodes $\mathbf{S}$, page 86
$\mathbf{anc}_{\mathbf{S}}(\mathsf{N})$	Sum ancestors of N, page 86
$\mathbf{desc}_{\mathbf{S}}(\mathsf{N})$	Sum descendants of N, page 86
$\mathbf{S}^c(\mathsf{S})$	Conditioning sums of S, page 86
$\bar{\boldsymbol{w}}$	Set of twin weights, page 86
$\bar{w}_{\mathsf{S},\mathsf{C}}$	Twin weight of sum-weight $w_{\mathsf{S},\mathsf{C}}$, page 86
$\mathbf{aug}(\mathcal{S})$	Augmented SPN of $\mathcal{S}$, page 86
$\mathsf{P}_{\mathsf{S}}^k$	k^{th} link of S, page 86
$\bar{\mathsf{S}}$	Twin sum node, page 86
$\mathcal{S}^{\mathbf{y}}$	Configured SPN, page 88
Y_{S}	Switching parent of Z_{S}, page 92
$\hat{\mathcal{S}}$	Max-product network of $\mathcal{S}$, page 94
$\hat{\mathsf{D}}$	Maximizing distribution node, page 94
$\hat{\mathsf{S}}$	Max node, page 94
$\hat{\mathsf{P}}$	Product node in a max-product network, page 94
$\mathbf{sup}(\mathsf{N})$	Support of N, page 109
$\mathbf{isup}(\mathsf{N})$	Inherent support of N, page 109
$I_X(\mathsf{N})$	Index set of distribution nodes with X in their scope and reachable by N, page 109
$\mathsf{S} \rightsquigarrow X$	S is regular selective with respect to X, page 110
$\mathcal{T}_{\mathbf{x}}$	Calculation tree, page 113

1 Introduction

In this thesis, we investigate sum-product networks (SPNs), a type of probabilistic model recently proposed in [79]. What is a probabilistic model and why is it useful? In a nutshell, a probabilistic model is a mathematical description of a probability distribution, allowing us to describe the behavior and interaction of random quantities, i.e. random variables (RVs). Using probabilities, we are able to reason and make decisions in uncertain or random domains. Probabilistic reasoning is sometimes treated with skepticism, so we present several arguments in the next section that probability theory is indeed a suitable tool for reasoning under uncertainty. In Section 1.2, we give a high-level motivation for using SPNs as probabilistic models, surpassing certain disadvantages of classical models regarding inference. Section 1.3 provides an overview of the contributions made in this thesis.

1.1 Motivation: Probability Theory for Reasoning under Uncertainty

In science and engineering, we strive for certainty, clarity and determinism. Deductive logic with syllogisms like

$$\frac{\begin{array}{l}\text{All men are mortal}\\ \text{Socrates is a man}\end{array}}{\text{Socrates is mortal}} \tag{1.1}$$

is appealing for rational minds due to its convincing clarity and seemingly eternal validity. The rules of deductive logic equip us with an inference machine which allows us to derive non-trivial results from our knowledge base, collected as a set of evidently true or accepted propositions, i.e. our axioms.

As another example, consider that we know the mass m and velocity v of a physical body. Classical mechanics tells us that the kinetic energy is given as

$$E = \frac{1}{2}mv^2. \tag{1.2}$$

Equations like this are the classical "working horse" of science: given m and v, (1.2) allows to infer E with certainty. It also allows to reason in multiple directions: given E and v, and knowing that mass is nonnegative, we can infer m, or given v, we can infer the ratio $\frac{E}{m}$. The

general theme in science is to express connections or dependencies of quantities of interest, and to perform reasoning using a certain calculus, in order to derive new, not self-evident insights.

What if we are unable to find these clear, deterministic connections like in (1.1) or (1.2), i.e. when our knowledge is afflicted with *uncertainty*? And what do we actually mean with uncertainty? A philosophical treatment of this question is out of the scope of this thesis, but we want to illustrate some possible sources of uncertainty:

1. The quantities of interest are *truly random*. According to the Copenhagen interpretation, the only "true" source of randomness in universe stems from quantum mechanics, possibly amplified by chaotic effects.

2. It is technically infeasible to consider all possible quantities which interact with the process of interest. For example, the physical process of roulette is far too detailed to be predicted with absolute accuracy.

3. It is technically possible to assess all quantities of interest with certainty, but we are not allowed to gather all information we need. For example, we could always make optimal decisions when playing Texas hold'em, would we know the cards of the other players.

4. We are confronted with objects whose properties are not clearly defined. For example a fig tree can be grown to a tree, to a shrub, and somehow in between.

We see that the causes of uncertainty can be diverse. The first example refers to true randomness (whether it exists or not is a subtle philosophical question), uncertainty in 2. and 3. stems from a somehow incomplete information state of the observer, and uncertainty in 4. appears to be somehow inherent to the object itself.

How can we deal with uncertainty? Humans can evidently deal with uncertain situations in an intuitive way. But what is a formal way to represent uncertainty and to perform reasoning under uncertainty? In other words and loosely speaking, how do we generalize deterministic modeling tools like (1.1) or (1.2) to uncertain situations? Furthermore, arguing from the perspective of *machine learning* and *artificial intelligence*, how do we perform reasoning under uncertainty in an automated fashion?

In this thesis we follow the prevailing approach using *probabilistic reasoning*, advocated and firmly established in the AI community by Pearl's textbook *Probabilistic Reasoning in Intelligent Systems* [70]. Pearl gives qualitative analogies between probability theory and the intuitive notion of *belief* or *plausibility*, reinforcing the use of probabilities as a tool for reasoning under *uncertain knowledge*, i.e. lack of information. It should be noted that *uncertainty* is not identical to *uncertain knowledge*. In the example about the fig tree above, uncertainty does actually not stem from an incomplete information state, but rather from the object itself, or from our perception of the world, i.e. the way we think about trees, shrubs, etc. Pearl's qualitative analogies between probability theory and intuitive reasoning patterns are:

- *Likelihood:* Probabilities naturally reflect statements like "A is more likely as B".

- *Conditioning:* Conditional probabilities of the form $P(A \mid B) = \frac{P(A,B)}{P(B)}$ are a core concept of probability theory. The conditional probability $P(A \mid B)$ naturally reflects statements like "how likely is A, given we know B (and nothing else)?". Furthermore, as argued by Pearl, conditional probabilities reflect *non-monotonic reasoning*: Say A denotes "Tim flies" and B "Tim is a bird". Since a bird is likely to fly, we would assess $P(A \mid B)$ as high. However, learning additionally C, "Tim is sick", we can at the same time assess $P(A \mid B, C)$ as low, without contradicting the former statement. Some of the early proposed reasoning systems are incapable of non-monotonic reasoning [70]. Conditional probabilities also obey the rule of the *hypothetical middle*:

> If two diametrically opposed assumptions impart two different degrees of belief onto a proposition Q, then the unconditional degree of belief merited by Q, should be somewhere between the two.

In terms of probability theory, the rule of the hypothetical middle is naturally verified by a special case of the law of total probability: Given $P(A \mid B) < P(A \mid \bar{B})$ we have $P(A \mid B) \leq P(A) = P(A \mid B)\, P(B) + P(A \mid \bar{B})\, P(\bar{B}) \leq P(A \mid \bar{B})$, since $P(B) + P(\bar{B}) = 1$, $P(B), P(\bar{B}) \geq 0$.

- *Relevance:* Probability theory reflect statements like "A and B are irrelevant to each other, given context C" by the notion of *conditional independence*, i.e. $P(A \mid B, C) = P(A \mid C) \Leftrightarrow P(B \mid A, C) = P(B \mid C)$.
- *Causation:* Humans clearly rely on the concept of direct causation in their reasoning. Causation is a highly controversial topic and hard to formalize [91]. Nevertheless, the directional character of conditional probabilities is capable to represent causal semantics. The factorization $P(A, B, C) = P(A \mid B)\, P(B \mid C)\, P(C)$ can be interpreted such that B is a direct cause of A, rendering C irrelevant when B is known. Furthermore, probabilities reflect a common sense reasoning pattern called *explaining away*: When an effect C can yield from two unlikely causes A and B, and we observe A, then B becomes less likely, although A and B might a priori be unrelated.

Another qualitative argument for probability theory is the so-called *Dutch book argument* [29]. This argument states that for an agent not reasoning according to the laws of probability, one always can find a set of bets yielding a long-term loss situation for this agent.

Furthermore, there have been efforts to show that probability theory is de facto inevitable when constructing a theory which generalizes deductive logic to *degrees of plausibility*. Deductive logic can be interpreted to have two degrees of plausibility, namely 1 (certainly true) and 0 (certainly false). To represent continuous degrees of plausibility, we want to use any value from $[0, 1]$, or more generally any real number. The seminal work in this area was delivered by Cox [25] and is thus often called *Cox's theorem*. The desiderata for a theory on degrees of plausibility, as formulated by Jaynes [48], are:

1. Degrees of plausibility are represented by real numbers.
2. Qualitative correspondence with common sense: when B becomes more plausible when updating $C \to C'$ and A remains equally plausible, $A \wedge B$ can not become less plausible.
3. If a conclusion can be reasoned out in more than one way, then every possible way must lead to the same results (consistency).

It can be shown [3, 25, 48] that any theory fulfilling these desiderata must have an equivalent to the Bayes rule (product rule):

$$P(A, B \mid C) = P(A \mid B, C)\, P(B \mid C) = P(B \mid A, C)\, P(A \mid C). \tag{1.3}$$

Furthermore, such theory must fulfill the sum rule, stated without loss of generality that

$$P(A \mid C) = 1, \text{when } A \text{ is certain given } C, \text{ and} \tag{1.4}$$
$$P(A \mid C) + P(\bar{A} \mid C) = 1. \tag{1.5}$$

Several point of criticism about Cox's theorem should be mentioned. As noted in [5, 43, 67], the work in [25, 48] is not completely rigorous in defining the underlying assumptions. Thus, the refinement of Cox's theorem seems to be an active process and is not commonly accepted.

Several authors argue that the first requirement, that degrees of plausibility are represented by a *single* number, is a severe restriction. The two most notable examples using *two-dimensional*

representation of belief are the *belief-function theories* [87] and the *possibility theory* [32]. One might further ask if we need to require *all* real numbers to measure degrees of plausibility in our theory. As argued by van Horn [98], this requirement can be relaxed, but we have to require that the set of numbers used to represent plausibility must be a dense set. Using sets with holes would produce counter-intuitive results.

A further criticism about Cox's theorem is concerned about the second desiderata. The argument states that qualitative correspondence with common sense can finally be not captured in satisfactory and universal manner. Common sense might include further reasonable desiderata, which could possibly rule out any formal theory. For further reading, we refer to [98] which gives an accessible overview of the discourse about Cox's theorem and its refinements.

A general point of critique about probability theory as reasoning tool under uncertainty addresses the inherent Aristotelian view of logic: a proposition A can either be true or false, and nothing else. Also probability theory, extending deductive logic with degrees of plausibility, follows this principle: The quantity $P(\omega \in A)$ reflects our degree of plausibility for the event that ω is contained in set A. Although we might not be certain about this event, we take for sure that $\omega \in A \vee \omega \notin A$, or in terms of probability $P(\omega \in A \cup A^c) = 1$. The theory about *fuzzy logic* and *fuzzy sets* considers degrees of set-membership rather than taking set-membership for granted. This leads to semantic and syntactic differences to probability theory. The example about whether a fig tree is a shrub or a tree is a typical example where fuzziness is more appropriate than probability. Statements like "this plant is with probability 0.5 a tree" make little sense when we are allowed to examine the plant and gather all information we desire. In this case uncertainty is not caused by lack of information but deterministically inherent to the object or to our world perception. Fuzziness on the other hand allows to make statements like "this plant is a shrub and a tree to the same extend". A comparison between probability theory and fuzziness is provided in [53].

In summary, there are strong qualitative as well as theoretical arguments for probability theory as suitable tool for reasoning under uncertainty, or more precisely under uncertain knowledge. Additionally, since the establishment of probabilistic models in the AI community by Pearl [70], we can observe increasing confidence in the theory and a bulk of positive results in practice. After having pointed out to alternative theories and to points of criticism, we want to finish this short motivation and cheerfully accept as a working hypothesis, that probability theory is an adequate tool for our purposes, or as Laplace puts it:

> *"Probability theory is nothing but common sense reduced to calculation."*

1.2 Why Sum-Product Networks?

An interesting point of probabilistic reasoning is that it can be automated, i.e. used in artificial intelligence or machine learning. To this end, we need mathematical representations of probability distributions, i.e. a probabilistic model, which can be interpreted by machines. The prevailing probabilistic models are Bayesian networks (BNs), also called belief networks or directed graphical models, and Markov networks (MNs), also called Markov random fields, or undirected graphical models. Both types of model have in common that each RV correspond to a node in a graph, where edges correspond to direct probabilistic dependencies. Due to their graphical representation, they are called *probabilistic graphical models* (PGMs). In [52], it is proposed to consider PGMs under the following three aspects:

1. **Representation**, i.e. which kind of distributions can a PGM represent?
2. **Learning**, i.e. how can a PGM be learned from data?
3. **Inference**, i.e. how can a PGM be applied to answer probabilistic queries?

Both BNs and MNs are appealing due to their representational semantics and interpretability. The underlying graph defines a set of *conditional independence* assertions, which characterizes the represented distribution. Furthermore, these conditional independencies are key for efficiently learning PGMs and performing inference within them. BNs and MNs, which we call *classical PGMs* in this thesis, are reviewed in Chapter 3.

What is the problem of BNs and MNs, calling for another type of probabilistic model, like SPNs? In fact, it is the aspect of inference: for classical PGMs there exist multi-purpose inference tools, where the junction tree algorithm is a well-known example. The computational effort of these inference tools scales unproportional to the apparent complexity of the underlying graph of the model. In fact, a slight modification of the graph can render exact inference intractable. A common remedy is to use approximate inference methods. However, this approach renders probabilistic modeling into "kind of black art", since approximate inference often behaves unpredictable and sometimes lacks of theoretical guarantees. Problems about inference often carry over to learning PGMs, when inference is used as a sub-routine of learning. A further point of criticism about inference in PGMs is that coming up with an inference machine is sometimes tedious work: we are not ready to go and use a specified PGM, i.e. immediately perform inference.

SPNs follow a different approach: they represent probability distributions and a corresponding *exact inference machine* for the represented distribution at the same time. An immediate interpretation of SPNs is that of a potentially deep, feedforward neural network with two kinds of neurons: sum neurons with weighted inputs and product neurons. The inputs are numeric values which are functions of the states of the modeled RVs. A corresponding probability distribution is defined by normalizing the SPN output over all possible states of the RVs. Therefore, the representational semantics of SPNs are somewhat more concealed than in BNs and MNs and harder to be interpreted by humans. However, if our main purpose is *automated* probabilistic reasoning, this concern is secondary. When SPNs fulfill certain structural constraints called completeness and decomposability (or consistency), then many inference scenarios in the represented distribution can be stated as simple network evaluations, stated as feedforward/upwards passes and backpropagation/backwards passes as known from neural network literature. Thus, the cost for these inference scenarios is *linear* in the network size, i.e. unlike as in BNs and MNs their inference cost is directly related to their representation. In that way, SPNs naturally incorporate a safeguard for inference: As long as we can represent an SPN, i.e. store it on a computer, we will probably also be able to perform inference. In a more principled approach this leads to *inference-aware learning*, i.e. incorporate inference cost already during learning. Although SPNs are naturally restricted to models with tractable inference, they come with considerable model power. SPNs naturally incorporate finer grained types of conditional independencies, e.g. context-specific independencies [10,19] and other advanced concepts, cf. Section 6.4.1. SPNs are thus a powerful modeling language for probability distributions and can immediately be used for inference, since they readily provide us with a general-purpose and conceptually simple inference machine.

1.3 Contributions and Organization of the Thesis

In recent years, SPNs have attracted much attention and several learning algorithms for SPNs have been proposed [31,39,40,72,73,79,81,84]. Furthermore, SPNs have been applied to various tasks like computer vision [4], classification [39] and speech/language modeling [14,73].

However, several theoretical and practical aspects of SPNs are not well understood. When studying the current literature on SPNs, one becomes aware of several open questions. The main contribution of this thesis is to deepen the theoretical understanding of SPNs and also to

clear some irritating misconceptions. In particular, in this thesis we treat the following points:[1]

1. In [79], two conditions on the SPN structure were given to guarantee tractable inference: *completeness* and *consistency*. As alternative to consistency, the stronger condition of *decomposability* was given. Decomposability can be understood as an independence assumption of a sub-SPN, while consistency is harder to grasp. In Section 4.3.1 we illustrate the notion of consistency in an alternative and probably clearer way.

2. One inference scenario which can efficiently be performed in SPNs is to *evaluate modified evidence* using the so-called differential approach [27]. So far, it has been tacitly assumed that the differential approach can be applied in consistent SPNs. In Section 4.3.2 we point out that this assumption is incorrect. However, we show that the differential approach *can* be applied in *decomposable* SPNs.

3. It was several times remarked in literature that *locally normalized* SPNs, i.e. SPNs whose weights are normalized for each sum node, have a normalization constant of 1. Furthermore, it was often stated that the sum-weights of locally normalized SPNs have immediate probabilistic semantics, see also point 7. below. However, so far it has not been clear whether this is a restriction of the model capabilities. We show in Section 4.3.3, Theorem 4.2, that this is not the case: any complete and consistent (or decomposable) SPN can be transformed into a locally normalized SPN with the same structure and representing the same distribution.

4. As mentioned above, decomposability implies consistency, but not vice versa. Either of the two conditions, together with completeness, enable efficient marginalization of the SPN distribution. An interesting question is if we can model distributions significantly (i.e. exponentially) more compact when using consistent SPNs instead of decomposable SPNs. We show in Section 4.3.4, Theorem 4.3, that this is not the case and that any complete and consistent SPN can be transformed into a complete and decomposable SPN with a worst-case polynomial grow in network size and required arithmetic operations.

5. In [79], SPNs were defined over RVs with finitely many states. In the same paper, it was mentioned that SPNs can be generalized to continuous RVs by using arbitrary distributions over the RVs as leaves of the SPN. This view was also adopted in follow-up work. However, inference mechanisms corresponding to the finite-state case were never derived for this generalized notion of SPNs. In Section 4.4 we derive these inference scenarios for generalized SPNs.

6. In the seminal work [79], a latent RV interpretation of sum nodes was introduced, i.e. interpreting each sum node as the result of a marginalized latent RVs. This was justified by explicitly incorporating these latent RVs in the model, i.e. by augmenting the model by the latent RVs. However, the proposed method to incorporate the latent RVs leads to irritating results:

   - The resulting model can be rendered *in-complete*, loosing its marginalization guarantees and thus strictly speaking the latent RV interpretation of sum nodes.
   - The resulting model fails to fully specify the joint distribution of model RVs and latent RVs.

   In Chapter 5, we give a remedy for these issues by proposing a modified augmentation method.

[1] This summary of contributions already requires some knowledge about classical PGMs and SPNs, and can be skipped by readers using this thesis as a tutorial text.

7. It was several times remarked in literature that the sum-weights have the interpretation of conditional probabilities, cf. also point 3. above. It was never made precise what are the events on which we condition here. Furthermore, given that a sum node corresponds to a latent RV, it must be possible to define a classical PGM like a BN, representing the dependency structure of the SPN, including the latent RVs. In Section 5.2 we find such a BN and make the interpretation of sum-weights as conditional probabilities precise.

8. In [79], *most-probable explanation* (MPE) inference was applied to SPNs for the task of face-image completion. To this end, a Viterbi-style algorithm was used. This algorithm was never justified, besides the fact that the model with latent RV was not fully specified, see point 6. above. We show in Section 5.3.1 that this is indeed a correct algorithm, *when applied to our proposed augmented SPN*. When applied to non-augmented SPNs, this algorithm has an unfortunate property, which we call *low-depth bias*.

9. For face-image completion, or more generally for data-completion, MPE inference in the augmented SPN is sub-optimal, since we maximize the probability over the RVs whose values are missing and *artificially introduced RVs*. We rather wish to find an MPE solution in the original, non-augmented SPN. Equivalently, we can maximize over missing values and *marginalize* the latent RV, i.e. finding *maximum-a-posteriori* (MAP) solution in the augmented SPN. In [71] it was conjectured that this problem NP-hard. In Section 5.3.2, Theorem 5.3, we show that this is indeed the case.

10. The (soft) EM-algorithm for learning sum-weights as sketched in [79] is mistaken. We provide a corrected EM-algorithm in Section 6.3.

11. The data-likelihood with respect to sum-weights is generally non-convex. However, in Section 6.4 (originally in [72]) we identify a sub-class of SPNs, which we call *selective* SPNs, for which globally optimal maximum likelihood parameters can be obtained in closed-form. This restricted model class, however, still has considerable model power.

The results presented in this thesis should give a clearer picture about the foundations of SPNs and even de-mystify some aspects. The thesis is mostly self-contained, where Chapter 2 reviews probability theory and graph theory to the extend as required here. Chapter 3 reviews classical PGMs, i.e. BNs and MNs. The main contributions are developed in Chapters 4–6: In Chapter 4 we first discuss SPNs over RVs with finitely many states, discuss basic notions and provide several insights. Furthermore, SPNs are generalized to RVs with infinitely many states. In Chapter 5, we introduce the latent RV interpretation of sum nodes, investigate the dependency structure of SPNs and establish the interpretation of sum-weights as conditional distributions. Chapter 6 provides basic learning techniques for parameter learning in SPNs and reviews some approaches for structure learning. Experiments and example application for SPNs are discussed in Chapter 7. In Chapter 8 we conclude the thesis and give some possible future research directions. For better readability, lengthy proofs of lemmas and propositions are shifted to the appendix.

2 Background and Notation

In this chapter, we introduce the mathematical tools required throughout the thesis. In section 2.1 we provide an accessible introduction and overview to *probability theory* in the measure theoretic sense. A detailed treatment can be found in [6, 9, 48, 85]. In section 2.2 we introduce the required notions from *graph theory*. This chapter uses adapted parts from [77].

2.1 Probability Theory

In this section we first discuss the basic mathematical formalism of probability theory – the *probability space*, see section 2.1.1. *Conditional probability* and *Bayes rule* is discussed in section 2.1.2. In section 2.1.3, we introduce the notion of *random variables* (RVs), which intuitively represent random quantities or "functions of randomness", and which are the central objects of interest in probabilistic modeling, machine learning and artificial intelligence. Finally, we discuss important tools to work with RVs – marginalization, conditional distributions, Bayes' rule for RVs and expectations, see section 2.1.4. Let us start to define our probabilistic object, the *probability space*.

2.1.1 Probability Spaces

The notion of *randomness* can be understood as a phenomenon with inherent lack of information for the observer. Thereby, it does not matter if we are considering "true" randomness, or randomness stemming from ignorance of facts. To model randomness, we first need a *universe* or *sample space* where randomness can take place. Let us denote this sample space as Ω, which can contain any elements and can be finite, countably infinite or uncountable. The elements ω of Ω are called *elementary events*. The basic mechanism we consider is the *random trial*, picking one element of $\omega \in \Omega$ in random fashion, i.e. we lack of precise knowledge about the identity of the picked ω. Since precisely one element is picked, the results $\omega = \omega_1$ and $\omega = \omega_2$, $\omega_1 \neq \omega_2$, are incompatible, i.e. the elementary events are *mutually exclusive*. In a basic approach to probability theory, we would like to assign a probability to each $\omega \in \Omega$, i.e. a degree of certainty that ω has been picked. As a convention, we agree that 0 is the degree corresponding to 'impossible' and 1 is the degree corresponding to 'certain'. We might now complain that Ω is too detailed or fine-grained for our purposes and that we are interested in assigning a probability for $A \subseteq \Omega$, where A is called an *event*. The probability of A shall represent the degree of belief

that $\omega \in A$ in the random trial. Intuitively, since elementary events are mutually exclusive, we would assess the probability of A as the sum of the probabilities of all $\omega \in A$. When we require that probabilities of the elementary events sum up to one, the probability of Ω is 1. This meets our requirements, since we know with certainty that $\omega \in \Omega$. Since the sum over the empty set yields zero, the probability $\emptyset$ is 0, which also meets our requirements, since we know $\omega \notin \emptyset$. Furthermore, we note that the probability of any event A agrees with the sum of probabilities of any partition of A, i.e. a collection of mutually exclusive sub-events $A_1, A_2, \ldots$, whose union is A. That is, we yield the same probabilities of A, no matter how we partition it, i.e. we get consistency in our theory.

Does this approach work and are we done? Is this what probability theory tries to achieve? In principle yes, but unfortunately this intuitive approach works only in the case when Ω is *countable*, yielding *discrete probability theory* [6]. When Ω is uncountable we run into several problems. First note that we cannot simply combine probabilities using sums over probabilities of mutually exclusive sub-events. Sums over uncountable sets of nonnegative number are defined as the supremum over countable subsets; however, these sums can only be finite when only countably many numbers are strictly positive. This would in effect restrict us again to the countable case.

So, we might hope that we still can assign probabilities to all subsets $A \subseteq \Omega$, not via sums but in some other way, such that we still get a similar behavior as in the countable case. Let us formalize our requirements as a function P, which we call a *probability measure* or *probability distribution*. This function should assign to each $A \subseteq \Omega$ a probability, i.e. a mapping $2^\Omega \mapsto \mathbb{R}$, where 2^Ω denotes the *power set* of Ω. The requirements are:

$$P(A) \geq 0. \tag{2.1}$$

$$P(\Omega) = 1. \tag{2.2}$$

$$P(A) = \sum_{n \in \mathbb{N}} P(A_n), \text{ for } A = \bigcup_{n \in \mathbb{N}} A_n,\ A_n \cap A_m = \emptyset, n \neq m. \tag{2.3}$$

We require probabilities to be non-negative, and that Ω is certain. The requirement $P(\emptyset) = 0$ follows from the third requirement: $P(A) = P(A) + P(\emptyset) \Rightarrow P(\emptyset) = 0$. The third requirement, *countable additivity* is what we are used to have in discrete probability theory: The sum over probabilities of countably many sub-events, which partition an event A, should yield the probability of A.

We refined our approach from the countable case and specified a probability measure P without explicitly constructing it. When Ω is countable, it can be shown that the requirements for P hold if and only if

$$\begin{aligned} P(\omega) &\geq 0,\ \forall \omega \in \Omega \\ \sum_{\omega \in \Omega} P(\omega) &= 1, \\ P(A) &= \sum_{\omega \in A} P(\{\omega\}). \end{aligned} \tag{2.4}$$

Thus, our construction of probabilities in the countable case coincides with our requirements on P. Now, is it reasonable to require such a function in the uncountable case? Measure theory gives arguments that it is not reasonable. When we consider $\Omega = [0, 1]$ and when we want to construct a *uniform* distribution, i.e. "spread" the probability measure uniformly over Ω, we would define for any interval $(a, b]$, $0 \leq a \leq b \leq 1$ that $P((a, b]) = b - a$, i.e. the length of the interval. Since P maps $2^\Omega \mapsto \mathbb{R}$, we need to generalize the notion of *length* to arbitrary subsets $A \subseteq \Omega$. However, it can be shown that there exist sets $[9, 85, 100]$ to which we can not assign a length in a reasonable way, i.e. there exist *non-measurable* sets as a consequence of the *axiom of*

choice. In our case, the existence of such sets contradicts at least one of our requirements on P, i.e. *we can not define a uniform probability on the unit interval*. As a remedy we want to exclude these non-measurable sets. For this purpose, we need to revisit our convenient, but too naive approach to define 2^Ω as domain of P. To this end, we introduce the notion of *sigma-algebra*.

Definition 2.1 (Sigma-algebra)**.** *Let Ω be any set. A* sigma-algebra $\mathcal{A}$ *over Ω is a* system of subsets *of Ω which contains Ω and which is closed under taking complements and countable unions, i.e. $\mathcal{A} \subseteq 2^\Omega$ where*

1. $\Omega \in \mathcal{A}$
2. $A \in \mathcal{A} \Rightarrow A^c = \Omega \setminus A \in \mathcal{A}$
3. $A_n \in \mathcal{A}, \forall n \in \mathbb{N} \Rightarrow \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{A}$

It is easily verified using De Morgan's laws that from 2. and 3. it follows that a sigma-algebra is also closed under countable intersections, i.e.

$$A_n \in \mathcal{A}, \forall n \in \mathbb{N} \Rightarrow \bigcap_{n\in\mathbb{N}} A_n \in \mathcal{A} \tag{2.5}$$

Let us consider some examples of sigma-algebras.

Example 2.1. *Let Ω be any set. $\mathcal{A} = \{\emptyset, \Omega\}$ is a sigma-algebra over Ω. In particular, when $\Omega = \emptyset$, $\mathcal{A} = \{\emptyset\}$ is a sigma-algebra over Ω (in fact the only one).*

Example 2.2. *Let Ω be any set. $\mathcal{A} = 2^\Omega$ is a sigma-algebra over Ω.*

Example 2.3. *Let $\Omega = \{\omega_1, \ldots, \omega_6\}$ and let ω_i represent the event "a fair die shows number i". $\mathcal{A}_1 = \{\emptyset, \{\omega_1, \omega_3, \omega_5\}, \{\omega_2, \omega_4, \omega_6\}, \Omega\}$ and $\mathcal{A}_2 = \{\emptyset, \{\omega_1, \omega_2, \omega_3\}, \{\omega_4, \omega_5, \omega_6\}, \Omega\}$ are sigma-algebras over Ω.*

As pointed out in Example 2.2, the powerset 2^Ω is always a sigma-algebra, actually the largest one, since any sigma-algebra is a subset of it. As already mentioned, the existence of non-measurable sets disqualifies the use of 2^Ω as domain of P when Ω is uncountable. We need to exclude these sets and use *smaller* sigma-algebras which are strict sub-sets of 2^Ω.

Furthermore, sigma-algebras are also useful when Ω is countable, since they represent a *degree of coarseness* we use to "observe" the sample space. In Example 2.3 we have two examples for sigma-algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ which are smaller than 2^Ω and are thus "coarser". $\mathcal{A}_1$ contains the impossible event, "die shows odd number", "die shows even number" and the certain event. $\mathcal{A}_2$ contains the impossible event, "die shows number ≤ 3", "die shows number > 3" and the certain event. Both sigma-algebras are "blind" to the event $\{\omega_2, \omega_3, \omega_5\}$, "die shows a prime number". Thus, using a particular sigma-algebra specifies a collection of those events we are interested in. The coarsest possible sigma-algebra is given in Example 2.1.

Before we succeed to exclude non-measurable sets, let us recapitulate our basic requirements for our probability theory: we have found that we need a universe or *sample space* Ω where randomness can take place. We specified randomness via the *random trial*, which picks an element ω from Ω in random fashion. We required a *probability measure* P which assigns probabilities to events $A \subseteq \Omega$, i.e. to results of the random trial of the form '$\omega \in A$'. For uncountable Ω, we have spotted difficulties from measure theory, and also found it convenient to specify the events A we are interested in. For this purpose we introduced *sigma-algebras*. Are we now done to describe the basic idea of probability theory? Yes, and we are now ready to define the central object in probability theory, the *probability space*.

Definition 2.2 (Probability Space)**.** *A* probability space *is a triple $(\Omega, \mathcal{A}, P)$, where*

- *The* sample space Ω *is any* non-empty set,

- $\mathcal{A}$ *is a* sigma-algebra *over* Ω*,*
- *The* probability measure P *is a function mapping* $\mathcal{A} \mapsto \mathbb{R}$*, satisfying* (2.1)*,* (2.2) *and* (2.3)*.*

Which sigma-algebra should we use for uncountable Ω, avoiding problems about non-measurable sets? From now on, we restrict ourselves to the real numbers as uncountable set. Let us reconsider the problem to define a uniform distribution on the unit interval, see above, i.e. we want to assign $P((a,b]) = b - a$ to any interval $(a,b] \subseteq (0,1]$. In a more general setting, we want assign a *volume* to subsets of $\Omega \subseteq \mathbb{R}^n$. We introduce an important concept from measure theory [9], the *measure space*.

Definition 2.3 (Measure space)**.** *Let* Ω *be any set and* $\mathcal{A}$ *a sigma-algebra over* Ω*. The tuple* $(\Omega, \mathcal{A})$ *is called* measurable space*. A function* $\mu \colon \mathcal{A} \mapsto [0, \infty]$ *with*

- $\mu(\emptyset) = 0$
- $\mu\left(\bigcup_{i\in\mathbb{N}} A_i\right) = \sum_{i\in\mathbb{N}} \mu\left(A_i\right)$*, for* $A_i \in \mathcal{A}$*,* $A_i \cap A_j = \emptyset$*,* $i \neq j$*.*

is called a measure *on* $(\Omega, \mathcal{A})$*. The triplet* $(\Omega, \mathcal{A}, \mu)$ *is called a* measure space*.*

A *probability measure* is a measure P with $P(\Omega) = 1$.

The sigma-algebra containing the sets for which the notion of volume can be defined, should at least contain the empty set and all half-open hypercubes, i.e. we consider

$$\mathcal{F}_H = \{\times_{k=1}^{n} (a_k, b_k] \,|\, a_k \leq b_k,\ k = 1, \ldots, n\}. \tag{2.6}$$

$\mathcal{F}_H$ is clearly no sigma-algebra, since e.g. the union of two hypercubes which are separated by some margin is not a hypercube. Being parsimonious, we want to add further sets and find the smallest sigma-algebra containing $\mathcal{F}_H$. It is difficult to explicitly construct such sigma-algebra, but we can describe it indirectly.

Definition 2.4 (Generated sigma-algebra)**.** *Let* Ω *be any set and* $\mathcal{F}$ *be a set of subsets of* Ω*, i.e.* $\mathcal{F} \subseteq 2^{\Omega}$*. The* sigma-algebra generated by $\mathcal{F}$ *is defined as*

$$\sigma(\mathcal{F}) = \bigcap_{\mathcal{F}\subseteq\mathcal{A}} \mathcal{A}, \tag{2.7}$$

where the intersection is taken over all sigma-algebras over Ω *containing* $\mathcal{F}$*.*

It can be shown that $\sigma(\mathcal{F})$ is indeed a sigma-algebra and uniquely determined by $\mathcal{F}$ [9]. It is also the *smallest* sigma-algebra containing $\mathcal{F}$, in that sense that it is a sub-set of *any* sigma-algebra containing $\mathcal{F}$. We can now apply $\sigma(\cdot)$ to the hypercubes $\mathcal{F}_H$ and yield the *Borel sets*.

Definition 2.5 (Borel sets)**.** *The Borel sets in* $\mathbb{R}^n$ *are defined as* $\mathcal{B}(\mathbb{R}^n) := \sigma(\mathcal{F}_H)$*. For any set* $E \subseteq \mathbb{R}^n$*, let* $\mathcal{B}(E) := \{B \cap E \,|\, B \in \mathcal{B}(\mathbb{R}^n)\}$*.*

The Borel sets in $\mathbb{R}^n$ are also generated by various other families of sets, as for instance

- Closed hypercubes $\{\times_{k=1}^{n} [a_k, b_k] \,|\, a_k < b_k,\ k = 1, \ldots, n\}$
- Open hypercubes $\{\times_{k=1}^{n} (a_k, b_k) \,|\, a_k \leq b_k,\ k = 1, \ldots, n\}$
- Hypercubes with rational corners $\{\times_{k=1}^{n} [a_k, b_k] \,|\, a_k \leq b_k,\ a_k, b_k \in \mathbb{Q}\ k = 1, \ldots, n\}$
- Open sectors $\{\times_{k=1}^{n} (-\infty, b_k)\}$
- Closed sectors $\{\times_{k=1}^{n} (-\infty, b_k]\}$
- Set of all open sets in $\mathbb{R}^n$

- Set of all closed sets in $\mathbb{R}^n$

Did we succeed and exclude all non-measurable sets, which caused problems for defining the notion of volume, or for our purposes, the uniform distribution? Yes, since it can be shown [6,9] that the volume function

$$\mathscr{L} : \mathcal{F}_H \mapsto \mathbb{R},\ \mathscr{L}(\times_{k=1}^{n} (a_k, b_k]) = \prod_{k=1}^{n} (b_k - a_k) \tag{2.8}$$

can be extended in a *unique way* to a measure on the Borel sets, yielding the measure space $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n), \mathscr{L})$. This generalization of volume to any set in $\mathcal{B}(\mathbb{R}^n)$ is called *Lebesque-Borel measure*. For completeness, we note that $\mathcal{B}(\mathbb{R}^n)$ is not the largest sigma-algebra on which $\mathscr{L}$ can be defined. $\mathcal{B}(\mathbb{R}^n)$ does not contain all possible subsets $B \subset A$, where $A \in \mathcal{B}(\mathbb{R}^n) : \mathscr{L}(A) = 0$. The sigma-algebra completed by these *null sets* are the *Lebesque-measurable sets*. For our purposes, however, we will consider Borel sets. We can now characterize the *uniform distribution*.

Example 2.4 (Uniform distribution)**.** *Let $\Omega = (0,1]^n$ and $P := \mathscr{L}$. Then $(\Omega, \mathcal{B}(\Omega), P)$ is a probability space and P is called the* uniform distribution *on the unit hypercube Ω.*

So, we succeeded to define a uniform distribution on the unit hypercube. However, Borel sets also allow us to consider more advanced distributions.

2.1.2 Conditional Probability, Independence and Bayes' Rule

Conditioning is an important concept in probability theory and a central mechanism for reasoning under uncertainty. Recall that the basic mechanism in probability theory is the random trial, which picks a ω from sample space Ω. Probabilities are the degrees of confidence we assign to outcomes $\omega \in A$. Intuitively, conditional probabilities represent an update of these beliefs, once we gathered new information of the random trial, e.g. we learned that $\omega \in B$:

Definition 2.6 (Conditional Probability)**.** *Let $(\Omega, \mathcal{A}, P)$ be a probability space, and $A, B \in \mathcal{A}$ be events. The* conditional probability *of A conditioned on B is defined as*

$$P(A \,|\, B) := \frac{P(A \cap B)}{P(B)}, \tag{2.9}$$

whenever $P(B) > 0$.

Clearly, (2.9) holds with A and B interchanged, i.e.

$$P(B \,|\, A) := \frac{P(A \cap B)}{P(A)}, \tag{2.10}$$

leading to *Bayes' rule*:

$$P(A \,|\, B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \,|\, A)\, P(A)}{P(B)}, \tag{2.11}$$

whenever $P(B) > 0$. Bayes' rule reverts the direction of conditioning and is one of the main working horses in probabilistic reasoning.

Definition 2.7 (Independence of Events)**.** *Let $(\Omega, \mathcal{A}, P)$ be a probability space, and $A, B \in \mathcal{A}$. A and B are independent when*

$$P(A \cap B) = P(A)\, P(B). \tag{2.12}$$

Generally let $A_\theta \in \mathcal{A}$ be a collection of events indexed by θ. A_θ are independent*, when for all finite sub-collections A_n of A_θ it holds that*

$$P(\cap_n A_n) = \prod_n P(A_n). \tag{2.13}$$

We now introduce the notion of *random variables*, which model random quantities as "a function of randomness".

2.1.3 Random Variables

We first need to introduce the notion of *measurable functions*:

Definition 2.8 (Measurable function)**.** *Let $(\Omega_1, \mathcal{A}_1)$ and $(\Omega_2, \mathcal{A}_2)$ be measurable spaces.. A function $f\colon \Omega_1 \mapsto \Omega_2$ is called* $\mathcal{A}_1 - \mathcal{A}_2$-measurable*, when for all $A \in \Omega_2$ it holds that $f^{-1}(A) \in \mathcal{A}_1$, i.e. when the preimage of any set in $\mathcal{A}_2$ under f is contained in $\mathcal{A}_1$.*

Measurability of a function is always defined with respect to particular sigma-algebras on the domain and co-domain of the functions. However, when clear from context, the sigma-algebras are not explicitly mentioned. Random variables (RVs) are defined as follows.

Definition 2.9 (Random variable)**.** *Let $(\Omega, \mathcal{A}, P)$ be a probability space. A* random variable *X defined on $(\Omega, \mathcal{A}, P)$ is a $\mathcal{A} - \mathcal{B}(\mathbb{R})$-measurable function $X\colon \Omega \mapsto \mathbb{R}$. The image of Ω under X, i.e. the set of values X assumes, is denoted as $\mathbf{val}(X)$.*

Here, we define RVs as mappings to the real numbers. However, RVs can be defined as mappings to any field, or more generally, as mappings to any measurable space. Throughout this thesis, we use letters X, Y and Z to denote RVs, also using various subscripts and superscripts, e.g. X_i, Y', etc. Elements from $\mathbf{val}(\cdot)$ are denoted by corresponding lower case letters, e.g. $x \in \mathbf{val}(X)$, $y' \in \mathbf{val}(Y')$, etc.

The *distribution* or *law* of an RVs is defined as follows.

Definition 2.10 (Distribution of a Random Variable)**.** *Let X be a RV defined on some probability space $(\Omega, \mathcal{A}, P)$. The induced probability measure*

$$P_X(B) := P(X^{-1}(B)), \text{ for } B \in \mathcal{B}(\mathbb{R}) \tag{2.14}$$

is called the distribution *of X.*

It can be shown that P_X is indeed a valid probability measure on $\mathcal{B}$ and that $(\mathbb{R}, \mathcal{B}(\mathbb{R}), P_X)$ is a probability space. We can restrict P_X to $\mathbf{val}(X)$ and consider the probability space $(\mathbf{val}(X), \mathcal{B}(\mathbf{val}(X)), P_X)$. Note that when $\mathbf{val}(X)$ is countable, $\mathcal{B}(\mathbf{val}(X)) = 2^{\mathbf{val}(X)}$: For each $x \in \mathbf{val}(X)$, $\mathcal{B}(\mathbf{val}(X))$ clearly contains $\{x\} = [x, x]$. Each $\mathcal{X} \subseteq \mathbf{val}(X)$ is countable and can be described as countable union $\mathcal{X} = \cup_{x \in \mathcal{X}} \{x\}$, and therefore $\mathcal{X} \in \mathcal{B}(\mathbf{val}(X))$. We denote RVs with *countably* many values as *discrete* RVs.

The distribution of an RV is characterized by its *cumulative distribution function*.

Definition 2.11 (Cumulative Distribution Function)**.** *Let X be an RV defined on probability space $(\Omega, \mathcal{A}, P)$. The* cumulative distribution function *(CDF) of X is defined as*

$$F_X(x) := P_X((-\infty, x]) =: P_X(X \leq x). \tag{2.15}$$

The following properties hold for any CDF:

$$F_X \text{ is non-decreasing} \tag{2.16}$$
$$F_X \text{ is right-continuous} \tag{2.17}$$
$$\lim_{x\to\infty} F_X(x) = 1 \tag{2.18}$$
$$\lim_{x\to-\infty} F_X(x) = 0 \tag{2.19}$$

Conversely, it can be shown that for a given function F_X with properties (2.16–2.19), there exists a probability space and RV X having F_X as CDF [6]. Therefore, a CDF *completely describes the distribution of* X, but leaves open the used probability space.

For discrete RVs, it is often more convenient to work with *probability mass functions*.

Definition 2.12 (Probability Mass Function (PMF))**.** *Let X be a discrete RV. The probability mass function (PMF) of X is defined as*

$$p_X(x) := P_X(\{x\}) =: P_X(X = x),\ x \in \mathbf{val}(X). \tag{2.20}$$

The CDF and the distribution of X are fully specified by the PMF:

$$F_X(x) = \sum_{x'\in\mathbf{val}(X)} p_X(x')\,\mathbb{1}(x' \le x), \tag{2.21}$$
$$P_X(A) = \sum_{x'\in A} p_X(x'), \tag{2.22}$$

where $\mathbb{1}$ is the *indicator function*. Thus, in the discrete case, any nonnegative function p_X with $\sum_{x\in\mathbf{val}(X)} p_X(x) = 1$ specifies a probability distribution over some RV X.

For RVs X with uncountable $\mathbf{val}(X)$ there might exist a *probability density function*.

Definition 2.13 (Probability density function (PDF))**.** *Let $F_X(x)$ be the CDF of a continuous RV X. If there exists a function p_X with*

$$F_X(x) = \int_{-\infty}^{x} p_X(x')\,\mathrm{d}x', \tag{2.23}$$

then p_X is called probability density function *(PDF) of X.*

When F_X is differentiable, we obtain p_X by the derivative of F_X:

$$p_X = \frac{\partial F_X}{\partial x}. \tag{2.24}$$

We overload notation and denote both PMFs and PDFs using p_X, since the difference is clear from context. Also the distribution of X is fully specified by the PDF:

$$P_X(A) = \int_A p_X\,\mathrm{d}\mathscr{L}, \tag{2.25}$$

where $\int \mathrm{d}\mathscr{L}$ denotes Lebesque-integral. Thus, any nonnegative integrable function p_X with $\int_{\mathbf{val}(X)} p_X \mathrm{d}\mathscr{L} = 1$ is a PDF and specifies a probability distribution over some RV X.

So far, we only considered single RVs. Most of the time we are interested in *multiple* RVs defined on the same probability space, distributed according to some *joint probability distribution*.

Definition 2.14 (Joint probability distribution)**.** *Let* $(\Omega, \mathcal{A}, P)$ *be a probability space,* X_n *be RVs* $X_n \colon \Omega \mapsto \mathbb{R}$*,* $n = 1, \dots, N$ *and* $\mathbf{X}(\omega) = (X_1(\omega), \dots, X_N(\omega))^T$ *be the corresponding vector-valued function* $\Omega \mapsto \mathbb{R}^N$*. The induced probability measure*

$$P_{\mathbf{X}}(B) := P(\mathbf{X}^{-1}(B)), \text{ for } B \in \mathcal{B}(\mathbb{R}^n) \tag{2.26}$$

is called the joint probability distribution*, or short* joint distribution *of* $\mathbf{X}$*. We define* $\mathbf{val}(\mathbf{X}) := \times_{n=1}^{N} \mathbf{val}(X_n)$*.*

Throughout this thesis, we use boldface letters $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ for vectors of RVs, also using various subscripts and superscripts, e.g. $\mathbf{X}_i$, $\mathbf{Y}'$. Note that in the literature on graphical models, RV vectors are commonly referred to as *sets* of RVs, and set notation is used, e.g. writing $\mathbf{Y} \subseteq \mathbf{X}$ for indicating that $\mathbf{X}$ contains all RVs also contained in $\mathbf{Y}$. This is somewhat a stretch of mathematical notions, and common statements like "$\mathbf{X} = \mathbf{x}$" are actually hard to interpret when $\mathbf{X}$ are sets, since sets are by definition not ordered. However, for consistency with literature, and enjoying light-weight set notation, we also refer as *sets of RVs* to RV vectors.

Furthermore, note that while $\mathbf{val}(X)$ is the image under a *single* RV X, the set $\mathbf{val}(\mathbf{X})$ in Definition 2.14 is not necessarily the image under vector-valued function $\mathbf{X}$, but a superset of it. Elements of $\mathbf{val}(\cdot)$ are denoted by corresponding boldface lowercase letters, e.g. $\mathbf{x} \in \mathbf{val}(\mathbf{X})$, $\mathbf{y}' \in \mathbf{val}(\mathbf{Y}')$. For a particular $\mathbf{x} = (x_1, \dots, x_N) \in \mathbf{val}(\mathbf{X})$ and subset $\mathbf{Y} = \{X_{i_1}, \dots, X_{i_K}\}$, $\{i_1, \dots, i_K\} \subseteq \{1, \dots, N\}$, where $i_k < i_l$ when $k < l$, we define the *projection* $\mathbf{x}[\mathbf{Y}] := (x_{i_1}, \dots, x_{i_K})$. For $X \in \mathbf{X}$ we define $\mathbf{x}[X] := \mathbf{x}[\{X\}]$.

Similar as for single RVs, we define a *joint CDF* for $\mathbf{X}$.

Definition 2.15 (Joint cumulative distribution function)**.** *Let* $\mathbf{X} = \{X_1, \dots, X_N\}$ *be a set of RVs defined on the same probability space. The* joint CDF *of* $\mathbf{X}$ *is defined as*

$$F_{\mathbf{X}}((x_1, \dots, x_N)) := P_{\mathbf{X}}((-\infty, x_1] \times \cdots \times (-\infty, x_N])). \tag{2.27}$$

The following properties hold for any CDF:

$$F_{\mathbf{X}} \text{ is non-decreasing in each } x_n \tag{2.28}$$

$$F_{\mathbf{X}} \text{ is continuous from above} \tag{2.29}$$

$$\forall k \colon \lim_{x_k \to \infty} F_{\mathbf{X}}((x_1, \dots, x_k, \dots, x_N)) = 1 \tag{2.30}$$

$$\forall k \colon \lim_{x_k \to -\infty} F_{\mathbf{X}}((x_1, \dots, x_k, \dots, x_N)) = 0 \tag{2.31}$$

Similar as for single RVs we can construct a probability space and RVs $\mathbf{X}$ from any $F_{\mathbf{X}}$ satisfying (2.28–2.31), i.e. the joint CDF fully specifies the distribution of $\mathbf{X}$ [6]. Furthermore, we can define the *joint PMF* for discrete RVs $\mathbf{X}$.

Definition 2.16 (Joint PMF)**.** *Let* $\mathbf{X}$ *be a set of discrete RVs. The* joint PMF *is defined as*

$$p_{\mathbf{X}}(\mathbf{x}) = P_{\mathbf{X}}(\{\mathbf{x}\}) \tag{2.32}$$

Similar as in the univariate case, the CDF and the distribution are determined by the PMF:

$$F_{\mathbf{X}}(\mathbf{x}) = \sum_{\mathbf{x}' \in \mathbf{val}(\mathbf{X})} p_{\mathbf{X}}(\mathbf{x}') \, \mathbb{1}(\forall X \in \mathbf{X} \colon \mathbf{x}'[X] \leq \mathbf{x}[X]), \tag{2.33}$$

$$P_{\mathbf{X}}(A) = \sum_{\mathbf{x}' \in A} p_{\mathbf{X}}(\mathbf{x}'). \tag{2.34}$$

Thus, any nonnegative $p_{\mathbf{X}}$ with $\sum_{\mathbf{x}\in\mathbf{val}(\mathbf{X})} p_{\mathbf{X}}(\mathbf{x}) = 1$ specifies a distribution over RVs $\mathbf{X}$.

For continuous RVs we define the *joint PDF*.

Definition 2.17 (Joint PDF)**.** *Let* $\mathbf{X}$ *be a set of continuous RVs. If there exists a function* $p_{\mathbf{X}}$ *with*

$$F_{\mathbf{X}}((x_1,\ldots,x_N)) = \int_{-\infty}^{x_1} \ldots \int_{-\infty}^{x_N} p_{\mathbf{X}}((x_1,\ldots,x_N))\,\mathrm{d}x_1\,\ldots\mathrm{d}x_N \tag{2.35}$$

then $p_{\mathbf{X}}$ *is called* joint PDF *of* $\mathbf{X}$*.*

When $F_{\mathbf{X}}$ is N-times differentiable, we obtain $p_{\mathbf{X}}$ by the partial derivative of $F_{\mathbf{X}}$:

$$p_{\mathbf{X}} = \frac{\partial^N F_{\mathbf{X}}}{\partial x_1,\ldots,\partial x_N}. \tag{2.36}$$

The distribution of $\mathbf{X}$ is specified by its PDF:

$$P_{\mathbf{X}}(A) = \int_A p_{\mathbf{X}}\,\mathrm{d}\mathscr{L}. \tag{2.37}$$

As already mentioned, we refer to RVs with countable many values as *discrete*. To RVs with finitely many values we refer as *finite-state* RVs. Non-discrete RVs, for which a PDF exists, are called *absolute continuous*. By the Lebesque decomposition [85], the probability measure of any RV can be decomposed into a discrete, an absolute continuous and a *singular continuous* part. A probability measure P is called singular continuous if it is concentrated on a set with Lebesque measure 0, i.e. there exists a set $S \subseteq \mathbb{R}$ with $\mathscr{L}(S) = 0$ and $P(\mathbb{R} \setminus S) = 0$. Singular continuous probability measures are not an exotic extreme case, but of practical importance: For example, consider two RVs X_1 and X_2 with $X_1 = X_2$, where X_1 (or equivalently X_2) is uniformly distributed on some interval. Thus, all probability mass is assigned to a subset of a line segment in $\mathbb{R}^2$, i.e. a set with Lebesque measure 0, meaning that this probability measure is singular continuous. Equality constraints of RVs are often of practical relevance, naturally requiring singular continuous RVs. However, for ease of discussion, we will assume that each RV is either discrete or absolute continuous, or short *continuous*, i.e. that a PDF exists. We will also, as commonly done in literature, refer as *distributions* to PMFs and PDFs. Thereby, we keep in mind that a distribution is actually a set function, assigning each set in the sigma-algebra a probability and is only *represented* by a PMF or PDF via (2.34) and (2.37), respectively. In that way, we can conveniently work with PMFs and PDFs and do not require to explicitly construct (and think about) a potentially abstract probability space. However, although we will mainly use PMFs and PDFs, many results in this thesis can also be generalized to CDFs, which is a general representation of probability distributions of RVs.

2.1.4 Expectations, Marginals and Conditionals

An important quantity of RVs is the *expected value*.

Definition 2.18 (Expected Value)**.** *Let* X *be an RV with distribution* p_X*. The expected value of* X *is defined as*

$$\mathbb{E}[X] = \begin{cases} \sum_{x\in\mathbf{val}(X)} x\,p_X(x) & \text{if } X \text{ is discrete} \\ \int_{x\in\mathbf{val}(X)} x\,p_X(x)\,\mathrm{d}x & \text{if } X \text{ is continuous.} \end{cases} \tag{2.38}$$

More generally, the expected value of a function $g\colon \mathbf{val}(X) \mapsto \mathbb{R}$ *is defined as*

$$\mathbb{E}[g(X)] = \begin{cases} \sum_{x \in \mathbf{val}(X)} g(x)\, p_X(x) & \text{if } X \text{ is discrete} \\ \int_{\mathbf{val}(X)} g(x)\, p_X(x)\, \mathrm{d}\mathscr{L} & \text{if } X \text{ is continuous.} \end{cases} \tag{2.39}$$

Considering a set of RVs $\mathbf{X}$, and when we are interested only in a subset $\mathbf{Y} \subseteq \mathbf{X}$ of the RVs we define *marginal distribution*:

Definition 2.19 (Marginal Distribution)**.** *Let* $\mathbf{X} = \{X_1, \dots, X_N\}$ *be a set of discrete RVs with distribution* $p_{\mathbf{X}}$*, let* $\mathbf{Y} \subset \mathbf{X}$ *and* $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y} = \{Z_1, \dots, Z_K\}$*. The* marginal distribution $p_{\mathbf{Y}}$ *of* $\mathbf{Y}$ *is given as*

$$p_{\mathbf{Y}}(\mathbf{y}) := \sum_{z_1} \cdots \sum_{z_K} p_{\mathbf{X}}(\mathbf{y}, z_1, \dots, z_K). \tag{2.40}$$

Similarly, when $\mathbf{X} = \{X_1, \dots, X_N\}$ *is a set of continuous RVs, the marginal distribution is given as*

$$p_{\mathbf{Y}}(\mathbf{y}) := \int_{z_1} \dots \int_{z_K} p_{\mathbf{X}}(\mathbf{y}, z_1, \dots, z_K)\, \mathrm{d}z_1 \dots \mathrm{d}z_K \tag{2.41}$$

Note that we also allow the extreme case $p_\emptyset \equiv 1$. In Definition 2.19 we introduced a frequently used and convenient notation style: a PMF or PDF $p_{\mathbf{X}}$ is actually a function $\mathbf{val}(\mathbf{X}) \mapsto \mathbb{R}$, i.e. it takes a tuple $\mathbf{x}$ as argument and maps it to the real numbers. An expression like $p_{\mathbf{X}}(\mathbf{y}, z_1, \dots, z_K)$ should interpreted as taking all arguments and arrange them in a tuple $\mathbf{x}$ using the original order, yielding $p_{\mathbf{X}}(\mathbf{x})$.

As noted in section 2.1.2, conditioning is an important tool in probability theory. In (2.9) we defined the conditional probability of general events A, B. We make similar definitions for distributions of RVs.

Definition 2.20. *Let* p *be a distribution over RVs* $\mathbf{X}$*, where all* $X \in \mathbf{X}$ *are either discrete or continuous. Further let* $\mathbf{Y} \subseteq \mathbf{X}$*, and* $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$*. The* conditional distribution $p_{\mathbf{Y}|\mathbf{Z}}$ *is given as*

$$p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \mid \mathbf{z}) = \frac{p_{\mathbf{X}}(\mathbf{y}, \mathbf{z})}{p_{\mathbf{Z}}(\mathbf{z})}, \tag{2.42}$$

whenever $p_{\mathbf{Z}}(\mathbf{z}) > 0$*.*

We generalize the interpretation of conditional distributions also to the case when $p_{\mathbf{Z}}(\mathbf{z}) = 0$: In this case also $p_{\mathbf{X}}(\mathbf{y}, \mathbf{z}) = 0$, allowing us to write

$$p_{\mathbf{X}}(\mathbf{y}, \mathbf{z}) = \tilde{p}_{\mathbf{Y};\mathbf{z}}(\mathbf{y}; \mathbf{z})\, p_{\mathbf{Z}}(\mathbf{z}) \tag{2.43}$$

for arbitrary distributions $\tilde{p}_{\mathbf{Y};\mathbf{z}}$ over $\mathbf{Y}$, parametrized by $\mathbf{z}$. We will interpret such $\tilde{p}_{\mathbf{Y};\mathbf{z}}$ also a conditional distribution $p_{\mathbf{Y}|\mathbf{Z}} = \tilde{p}_{\mathbf{Y};\mathbf{z}}$, for $\mathbf{z}$ with $p_{\mathbf{Z}}(\mathbf{z}) = 0$. This is the approach taken by several authors such as de Finetti [29], actually *defining* the joint distribution as $p_{\mathbf{X}}(\mathbf{y}, \mathbf{z}) := p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \mid \mathbf{z})\, p_{\mathbf{Z}}(\mathbf{z})$.

Bayes' rule in terms of distributions is stated as

$$p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \mid \mathbf{z}) = \frac{p_{\mathbf{Z}|\mathbf{Y}}(\mathbf{z} \mid \mathbf{y})\, p_{\mathbf{Y}}(\mathbf{y})}{p_{\mathbf{Z}}(\mathbf{z})}, \text{ when } p_{\mathbf{Z}}(\mathbf{z}) > 0, \tag{2.44}$$

$$p_{\mathbf{Z}|\mathbf{Y}}(\mathbf{z} \mid \mathbf{y}) = \frac{p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \mid \mathbf{z})\, p_{\mathbf{Z}}(\mathbf{z})}{p_{\mathbf{Y}}(\mathbf{y})}, \text{ when } p_{\mathbf{Y}}(\mathbf{y}) > 0. \tag{2.45}$$

Using conditional distributions, we can factorize any joint distribution $p_{\mathbf{X}}$ over $\mathbf{X} = \{X_1, \dots, X_N\}$

using the *chain rule of probability*:

$$\begin{aligned} p_{\mathbf{X}}(x_1, \ldots, x_N) &= p_{X_N|X_{N-1}\ldots X_1}(x_N \,|\, x_{N-1}, \ldots, x_1) \ \ldots \ p_{X_2|X_1}(x_2 \,|\, x_1)\, p_{X_1}(x_1) \\ &= \prod_{i=1}^{N} p_{X_i|X_{i-1},\ldots,X_1}(x_i \,|\, x_{i-1}, \ldots, x_1). \end{aligned} \tag{2.46}$$

The chain rule holds for any ordering of the RVs $\mathbf{X}$.

We define the notion of *conditional independence* for RVs.

Definition 2.21 (Conditional Independence)**.** *Let $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ be mutually disjoint sets of RVs with joint distribution $p_{\mathbf{X},\mathbf{Y},\mathbf{Z}}$. $\mathbf{X}$ and $\mathbf{Y}$ are* conditionally independent *given $\mathbf{Z}$, annotated as $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \,|\, \mathbf{Z}$, if one of the following equivalent conditions hold:*

- $p_{\mathbf{Z}}(\mathbf{z}) > 0 \Rightarrow p_{\mathbf{X}|\mathbf{Y},\mathbf{Z}}(\mathbf{x} \,|\, \mathbf{y}, \mathbf{z}) = p_{\mathbf{X}|\mathbf{Z}}(\mathbf{x} \,|\, \mathbf{z})$.
- $p_{\mathbf{Z}}(\mathbf{z}) > 0 \Rightarrow p_{\mathbf{Y}|\mathbf{X},\mathbf{Z}}(\mathbf{y} \,|\, \mathbf{x}, \mathbf{z}) = p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \,|\, \mathbf{z})$.
- $p_{\mathbf{Z}}(\mathbf{z}) > 0 \Rightarrow p_{\mathbf{X},\mathbf{Y}|\mathbf{Z}}(\mathbf{x}, \mathbf{y} \,|\, \mathbf{z}) = p_{\mathbf{X}|\mathbf{Z}}(\mathbf{x} \,|\, \mathbf{z})\, p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} \,|\, \mathbf{z})$.

For the corresponding condition with $\mathbf{Z} = \emptyset$, we say that $\mathbf{X}$ and $\mathbf{Y}$ are independent*, annotated as $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}$.*

So far we only considered sets of RVs $\mathbf{X}$ which are either all discrete or all continuous, thus either considering PMFs or PDFs. However, often we require a *mixed case* where $\mathbf{X}$ can contain both. For example, consider a binary RV X with $\mathbf{val}(X) = \{x_1, x_2\}$ and a continuous RV Y with $\mathbf{val}(Y) = \mathbb{R}$, and let us define the function

$$\begin{aligned} p_{X,Y}(x, y) = 0.4 \times \mathbb{1}(x = x_1) \times \mathcal{N}(y \,|\, \mu_1, \sigma_1) \\ + 0.6 \times \mathbb{1}(x = x_2) \times \mathcal{N}(y \,|\, \mu_2, \sigma_2), \end{aligned} \tag{2.47}$$

where $\mathcal{N}$ is the Gaussian distribution:

$$\mathcal{N}(x \,|\, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{2.48}$$

Function $p_{X,Y}$, which is in fact a *Gaussian mixture*, is a PMF with respect to X and a PDF with respect to Y. Generally, mixed distributions can be defined using conditional distributions:

Definition 2.22 (Mixed Distribution Function)**.** *Let $\mathbf{X}$ be a set of RVs and $\mathbf{Y}$, $\mathbf{Z}$ be a partition of $\mathbf{X}$, where $\mathbf{Y} \neq \emptyset$, $\mathbf{Z} \neq \emptyset$, and $\mathbf{Y}$ contains only discrete RVs and $\mathbf{Z}$ contains only continuous RVs. Let $p_{\mathbf{Y}}$ be a PMF over $\mathbf{Y}$ and $p_{\mathbf{Z}|\mathbf{Y}}$ a conditional PDF over $\mathbf{Z}$. The* mixed distribution function *over $\mathbf{X}$ is defined as*

$$p_{\mathbf{X}}(\mathbf{y}, \mathbf{z}) := p_{\mathbf{Z}|\mathbf{Y}}(\mathbf{z} \,|\, \mathbf{y})\, p_{\mathbf{Y}}(\mathbf{y}) \tag{2.49}$$

Marginal and conditional distributions, Bayes' rule, the chain rule and (conditional) independence naturally carry over to mixed distributions.

To simplify notation, we will omit the sub-scripts of any joint, marginal and conditional distribution, when the meaning is clear from context, and simply write p. Furthermore, we will adopt a common notation style: any distribution p over $\mathbf{X}$ is either a PMF, a PDF or a mixed distribution, and thus a function $\mathbf{val}(\mathbf{X}) \mapsto \mathbb{R}$, e.g. $p(\mathbf{x}) = 0.5$. Often we write $\mathbf{X}$ as *argument* of p when making a general statement, i.e. "statement about $p(\mathbf{X})$" shall be interpreted as "statement about $p(\mathbf{x})$", $\forall \mathbf{x} \in \mathbf{val}(\mathbf{X})$. This notation is also used to signal the identity of the distributions, i.e. writing $p(\mathbf{X})$ when referring to a (marginal) distribution over $\mathbf{X}$.

2.2 Graph Theory

In this section we review the required notions from graph theory. In the context of probabilistic graphical models, see Chapter 3, graphs are used to model joint distributions. In the context of SPNs they generally represent networks of arithmetic operations.

A graph is defined as follows:

Definition 2.23 (Graph)**.** *A* graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *is a tuple consisting of a set of* nodes *or* vertices $\mathbf{V}$ *and a set of* edges $\mathbf{E}$*. The nodes can be any objects and an edge* $E \in \mathbf{E}$ *is a tuple* (V_i, V_j)*,* $V_i, V_j \in \mathbf{V}$*.*

The edges of the graph define direct relationships among the nodes. Edges can be interpreted as *directed* or *undirected*, meaning that the order of V_i and V_j in an edge (V_i, V_j) matters, or not. An undirected edge between V_i and V_j is denoted as $(V_i - V_j)$, and a directed edge as $(V_i \to V_j)$. To make the representation of an undirected edge unique, we can require that for $(V_i - V_j)$ both (V_i, V_j) and (V_j, V_i) are contained in $\mathbf{E}$.

Depending on the types of edges in a graph we define:

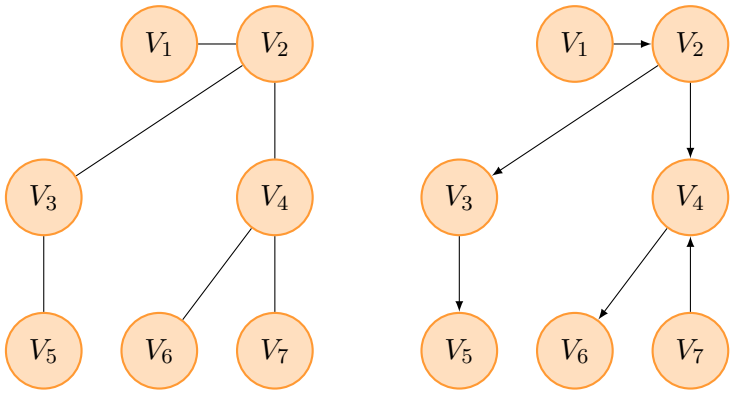
Definition 2.24 (Directed, Undirected Graphs)**.** *A graph* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *is* directed *(*undirected*) if all edges* $E \in \mathbf{E}$ *are directed (undirected).*

There are generalized notions of graphs, for instance allowing multiple edges (directed and undirected) between nodes and hyper-edges, connecting more than two nodes. The graphs in Definition 2.23 are denoted as simple graphs. For our purposes it suffices to consider simple graphs, to which we simply refer as graphs. Furthermore, throughout the thesis we exclude *self-edges* $(V_i \to V_i)$ and $(V_i - V_i)$.

Graphs are visually represented using circles for nodes, line-segments representing undirected edges, and arrows representing directed edges.

Example 2.5. *Consider Figure 2.1, showing an undirected and a directed graph. For both graphs* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *the set of nodes is* $\mathbf{V} = \{V_1, \ldots, V_7\}$*. The corresponding edges are*

1. $\mathbf{E} = \{(V_1 - V_2), (V_2 - V_3), (V_2 - V_4), (V_3 - V_5), (V_4 - V_6), (V_4 - V_7)\}$,
2. $\mathbf{E} = \{(V_1 \to V_2), (V_2 \to V_3), (V_2 \to V_4), (V_3 \to V_5), (V_4 \to V_6), (V_7 \to V_4)\}$.



(a) Undirected graph. (b) Directed graph.

Figure 2.1: Visual representation of a directed and an undirected graph, having the same set of nodes $\mathbf{V} = \{V_1, \ldots, V_7\}$.

In directed graphs we define a parent-child relationship:

Definition 2.25 (Parents and Children)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a directed graph and $V_i, V_j \in \mathbf{V}$. If $(V_i \to V_j) \in \mathbf{E}$, then V_i is a* parent *of V_j and V_j is a* child *of V_i. The set $\mathbf{pa}_{\mathcal{G}}(V_j)$ consists of all parents of V_j and the set $\mathbf{ch}_{\mathcal{G}}(V_i)$ contains all children of V_i. The sub-script is omitted if the graph is clear from context.*

For undirected graphs we define the notion of neighborhood:

Definition 2.26 (Neighbor)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be an undirected graph and $V_i, V_j \in \mathbf{V}$. If $(V_i - V_j) \in \mathbf{E}$, then V_i is a neighbor of V_j. The set of all neighbors of V_i is denoted as $\mathbf{nb}_{\mathcal{G}}(V_i)$. The sub-script is omitted if the graph is clear from context.*

While edges represent *direct* relationships between two nodes, *paths* and *trails* describe *indirect* relationships across several nodes:

Definition 2.27 (Path, Trail)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a graph (directed or undirected). A tuple of nodes $\Pi = (V_1, \ldots, V_n)$, $n \geq 1$, with $V_1, \ldots, V_n \in \mathbf{V}$ is a* path *from V_1 to V_n if*

$$\mathcal{G} \text{ is a directed graph and } \forall\, i \in \{1, \ldots, n-1\} : (V_i \to V_{i+1}) \in \mathbf{E}, \text{ or} \tag{2.50}$$

$$\mathcal{G} \text{ is an undirected graph and } \forall\, i \in \{1, \ldots, n-1\} : (V_i - V_{i+1}) \in \mathbf{E}. \tag{2.51}$$

In a directed *graph, a tuple of nodes $\Pi = (V_1, \ldots, V_n)$, $n \geq 1$, with $V_1, \ldots, V_n \in \mathbf{V}$ is called a* trail *between V_1 to V_n if*

$$\forall\, i \in \{1, \ldots, n-1\} : (V_i \to V_{i+1}) \in \mathbf{E} \vee (V_{i+1} \to V_i) \in \mathbf{E}. \tag{2.52}$$

In an undirected *graph, every path is a trail and vice versa. The length n of a path or trail is denoted as $|\Pi|$.*

Note that (V_i) is always a path of length 1, i.e. there is always a path from V_i to itself. *Cycles* are defined as follows:

Definition 2.28 (Cycle)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a directed or undirected graph, and $V \in \mathbf{V}$. A path Π with $|\Pi| \geq 2$ from V to V is denoted as a* cycle*.*

We could also include cycles of length 1 if there is an self-edge present for the node under consideration. However, as already mentioned, we do not consider self-edges in this thesis, thus any cycle has at least a length of 2. For undirected graphs, *chordal graphs* are defined as follows:

Definition 2.29 (Chordal Graph)**.** *An undirected graph $\mathcal{G}$ is* chordal *or* triangulated*, if there is no cycle of length ≥ 4 without an edge joining two non-neighboring nodes in the cycle.*

For directed graphs, we define acyclic graphs:

Definition 2.30 (Directed Acyclic Graphs)**.** *A graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is a* directed acyclic graph[2] *(DAG) if it contains no cycles.*

For example, the undirected graph shown Figure 2.1(b) is a DAG. In DAGs we can naturally define ancestors and descendants.

Definition 2.31 (Ancestor, Descendant)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a DAG and $V_i, V_j \in \mathbf{V}, i \neq j$. Node V_i is an* ancestor *of V_j if there exists a path from V_i to V_j. V_j a* descendant *of V_i if V_i is an ancestor of V_j. In particular, node V_i is a descendant and an ancestor of itself. The set of all ancestors and descendants of some node $V_j \in \mathbf{V}$ are denoted as $\boldsymbol{anc}_{\mathcal{G}}(V_j)$ and $\boldsymbol{desc}_{\mathcal{G}}(V_i)$, respectively. We further define the nondescendants of V_i as*

$$\boldsymbol{ndesc}_{\mathcal{G}}(V_i) = \mathbf{V} \setminus \boldsymbol{desc}_{\mathcal{G}}(V_i). \tag{2.53}$$

The sub-scripts are omitted when the graph is clear from context.

[2] Actually acyclic directed graph. However, we use the common acronym DAG.

DAGs with a dedicated root are called *rooted DAGs*:

Definition 2.32 (Rooted DAG)**.** *A DAG $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is* rooted *when it has a unique node V_1 with $\mathbf{pa}(V_1) = \emptyset$ and for all other nodes V_i, $i \neq 1$, we have $|\mathbf{pa}(V_i)| \geq 1$. V_1 is called the* root *of $\mathcal{G}$.*

We further define *trees*:

Definition 2.33 (Directed/Undirected Tree)**.** *An undirected graph $\mathcal{G}$ is an* undirected tree *or simply* tree*, if any two distinct nodes are connected by exactly one path.*

A rooted DAG $\mathcal{G}$ is a directed tree *if for all nodes V_i we have $|\mathbf{pa}(V_i)| \leq 1$.*

For example, the undirected graph shown Figure 2.1(a) is a tree. The directed graph shown Figure 2.1(b) is not a directed tree tree, because V_4 has two parents; by replacing edge $(V_7 \to V_4)$ by $(V_4 \to V_7)$, it would be transformed into a directed tree.

We will also consider graphs which are *sub-graphs* or *super-graphs* of other graphs:

Definition 2.34 (Sub-graph, Super-graph)**.** *A graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is a* sub-graph *of $\mathcal{G}' = (\mathbf{V}', \mathbf{E}')$, if $\mathbf{V} \subseteq \mathbf{V}'$ and $E \in \mathbf{E} \Rightarrow E \in \mathbf{E}'$. Further, $\mathcal{G}'$ is a* super-graph *of $\mathcal{G}$ if $\mathcal{G}$ is a sub-graph of $\mathcal{G}'$.*

We further define induced graphs as follows:

Definition 2.35 (Induced Graph)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a graph and $\mathbf{V}' \subseteq \mathbf{V}$. The* graph induced *by $\mathbf{V}'$ is defined as $\mathcal{G}' = (\mathbf{V}', \mathbf{E}')$, where $\mathbf{E}' = \{(V_i, V_j) \in \mathbf{E} \,|\, V_i, V_j \in \mathbf{V}'\}$*

Cliques in undirected graphs are defined as follows.

Definition 2.36 (Clique, Maximal Clique)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be an undirected graph. A set of nodes $\mathbf{C} \subseteq \mathbf{V}$ is a* clique *if there exists an edge between all pairs of nodes in the subset $\mathbf{C}$, i.e. if*

$$\forall C_i, C_j \in \mathbf{C}, i \neq j : (C_i - C_j) \in \mathbf{E}. \tag{2.54}$$

The clique $\mathbf{C}$ is a maximal clique *if adding any node $V \in \mathbf{V} \setminus \mathbf{C}$ makes it no longer a clique.*

Cliques are related to *complete* graphs:

Definition 2.37 (Complete Graph)**.** *An undirected graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is* complete *if every pair of distinct nodes is connected by an edge, i.e. if*

$$\forall V_i, V_j \in \mathbf{V}, i \neq j : (V_i - V_j) \in \mathbf{E}. \tag{2.55}$$

Thus, set $\mathbf{C} \subseteq \mathbf{V}$ is a clique if the *induced* graph $\mathcal{G}' = (\mathbf{C}, \mathbf{E}')$ is complete.

Furthermore, we introduce the notion of a topological ordering for directed graphs:

Definition 2.38 (Topological Ordering)**.** *Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be a directed graph. A list $V_1, \ldots, V_N$ of the nodes in $\mathbf{V}$ is* topologically ordered *if $i < j \Rightarrow (V_j \to V_i) \notin \mathbf{E}$.*

A topological ordering does not need to be unique. We will sometimes also refer to $V_1, \ldots, V_N$ with $i > j \Rightarrow (V_j \to V_i) \notin \mathbf{E}$ as (reversed) topological ordering.

Example 2.6. *$V_1, V_2, V_3, V_5, V_7, V_4, V_6$ and $V_1, V_7, V_2, V_4, V_3, V_6, V_5$ are topological orderings of the nodes in the graph in Figure 2.1(b).*

A well-known and important connection between DAGs and topological orderings is:

Theorem 2.1. *A directed graph $\mathcal{G}$ is acyclic (and thus a DAG) if and only if there exists a topological ordering of its nodes.*

Finally, we define *fully connected* DAGs.

Definition 2.39 (Fully Connected DAG)**.** *Let* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *be a DAG and* $V_1, \ldots, V_N$ *be a topological ordering of the nodes in* $\mathbf{V}$*. Then* $\mathcal{G}$ *is called* fully connected *when*

$$(V_n \to V_m) \in E,\ \forall n = 1, \ldots, N-1,\ \forall m = n+1, \ldots, N. \tag{2.56}$$

A fully connected DAG has a unique topological ordering.

3
Classical Probabilistic Graphical Models

Probabilistic graphical models (PGMs) are, so to say, a marriage of the two theories reviewed in the last chapter: *Probability theory* and *graph theory*. In this chapter we review the two most prominent types of classical graphical models – *Bayesian networks* (BNs), and *Markov networks* (MNs). We refer to them as "classical" PGMs, since sum-product networks, the models of main interest in this thesis, can be considered as "new-style" PGMs: similar as BNs and MNs they form a graphical description of probability distributions; however, their syntax and semantics largely differ from classical PGMs.

As motivated in Chapter 1, the central point of interest is to define models capable to represent probabilities and to perform reasoning in automated fashion. Thus, we require probability theory, but why do we require graphs? What are the problems, for which graphical models should be helpful to solve them? The following example illustrates the issue.

Example 3.1 (Probability Distribution over 100 Coins)**.** *Consider an experiment where we want to model the outcome of* 100 *parallel coin tosses, where the* i^{th} *coins is modeled by a binary RV* X_i *with* $\mathbf{val}(X_i) = \{0, 1\}$*,* 0 *standing for 'heads' and* 1 *for 'tails'. Our task is to model the joint distribution* $p(X_1, \ldots, X_{100})$*. Since we assume that coins usually are independent of each other, we assume*

$$p(X_1, \ldots, X_{100}) = \prod_{i=1}^{100} p(X_i). \tag{3.1}$$

The i^{th} *distribution* $p(X_i)$ *is easily represented as a single parameter* $\theta^i \in [0, 1]$*, thus our model uses* 100 *free parameters. Reasoning using this model is also computationally cheap, and de facto trivial.*

Now, assume that we live in a universe where coins do not behave independently, but in a complicated and entangled manner and (3.1) *does not hold. Thus we aim to model the joint distribution* $p_{\mathbf{X}}$*, assigning each* $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ *a probability. However, there are* 2^{100} *elements in* $\mathbf{val}(\mathbf{X})$*, so that a direct implementation storing* $p_{\mathbf{X}}$ *in a table requires*

$$2^{100} - 1 = 1,267,650,600,228,229,401,496,703,205,375$$

free parameters. Even when using just a single byte for encoding each parameter, this number of parameters can not be stored on any computer architecture today, besides the need to process this huge table in order to perform reasoning.

Example 3.1 demonstrates two extreme cases which can occur when we want to model the joint distribution of a set of RVs $\mathbf{X}$, already as in a rather simple example as modeling 100 coins. These cases are,

- that the RVs are fully *independent*, leading to a computationally easy and trivial model, which is rarely useful in practice, or
- that the RVs are fully *dependent*, and that we can not find any structure or regularity in the joint distribution we can exploit, so that we are forced to exhaustively model the joint distribution.

PGMs aim at the domain in between these extreme cases – the important concept here is *conditional independence*. That is, when using PGMs, we aim to identify suitable conditional independence assumptions in our model domain and use these to reduce the potential huge computational burden. These conditional independencies are encoded in graphs, where each node corresponds to an RV and vice versa. Therefore, we use $\mathbf{V} := \mathbf{X}$ as node set of graphs in this chapter.

We review BNs and MNs in Sections 3.1 and 3.2, respectively. Learning and inference for PGMs is reviewed in Sections 3.3 and 3.4, respectively.

3.1 Bayesian Networks

We introduce BNs by means of their *factorization properties* and then discuss their conditional independence assertions in Section 3.1.2.

3.1.1 Definition via Factorization

Definition 3.1 (Bayesian Network)**.** *A* Bayesian network $\mathscr{B}$ *over RVs* $\mathbf{X}$ *is a tuple* $(\mathcal{G}, \{p_X\}_{X\in\mathbf{X}})$*, where* $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ *is a DAG over RVs* $\mathbf{X}$*, and* p_X *are conditional probability distributions (CPDs)* $p_X(X \mid \mathbf{pa}(X))$*, for each* $X \in \mathbf{X}$*. To the graph* $\mathcal{G}$ *we also refer as* BN structure*. The BN* $\mathscr{B}$ *defines a joint probability distribution* $p_{\mathscr{B}}(\mathbf{X})$ *according to*

$$p_{\mathscr{B}}(\mathbf{X}) = \prod_{X\in\mathbf{X}} p_X(X \mid \mathbf{pa}(X)). \tag{3.2}$$

As usual, we drop the index of $p_X(X \mid \mathbf{pa}(X))$ and write $p(X \mid \mathbf{pa}(X))$ when the meaning is clear from context. A BN always defines a valid probability distribution; (3.2) can be interpreted as the chain rule of probability, were certain conditional independence assumptions dictated by graph $\mathcal{G}$ are made. This is illustrated in the following example.

Example 3.2 (Bayesian Network)**.** *Let* $\mathscr{B}$ *be a BN having the DAG* $\mathcal{G}$ *in Figure 2.1(b) as structure, where* $X_i = V_i$*. According to Example 2.6, we know that* $X_1, X_2, X_3, X_5, X_7, X_4, X_6$ *is a topological order of* $\mathcal{G}$*. Using the chain rule following this order, we get*

$$\begin{aligned} p(X_1, \ldots, X_7) = {} & p(X_1)\, p(X_2 \mid X_1)\, p(X_3 \mid X_2, X_1)\, p(X_5 \mid X_3, X_2, X_1)\, p(X_7 \mid X_5, X_3, X_2, X_1) \\ & p(X_4 \mid X_7, X_5, X_3, X_2, X_1)\, p(X_6 \mid X_4, X_7, X_5, X_3, X_2, X_1), \end{aligned} \tag{3.3}$$

which holds for any distribution. The BN contains the conditional distribution $p(X_4 \mid X_7, X_2)$ *for* X_4*, since* $\mathbf{pa}(X_4) = \{X_7, X_2\}$*. Thus, the graph* $\mathcal{G}$ *"dictates" that* $p(X_4 \mid X_7, X_5, X_3, X_2, X_1)$ *shall be assumed as* $p(X_4 \mid X_7, X_2)$*, i.e. that* $X_4 \perp\!\!\!\perp X_5, X_3, X_1 \mid X_7, X_2$*. Doing this for all conditional*

distributions in (3.3)*, we yield the joint probability distribution*

$$\begin{aligned} p_{\mathscr{B}}(X_1,\dots,X_7) &= p(X_1)\,p(X_2\,|\,X_1)\,p(X_3\,|\,X_2)\,p(X_4\,|\,X_3)\,p(X_5)\,p(X_6\,|\,X_2,X_5)\,p(X_7\,|\,X_6) \\ &= \prod_{i=1}^{7} p(X_i\,|\,\mathbf{pa}_{\mathcal{G}}(X_i)), \end{aligned}$$

i.e. the BN factorization (3.2)*.*

Often the conditional probability distributions p_X are specified by a parameter set $\boldsymbol{\theta}^X$, writing $p(X\,|\,\mathbf{pa}(X);\boldsymbol{\theta}^X)$. In this case, we collect all parameters in a set $\boldsymbol{\Theta} = \{\boldsymbol{\theta}^X\}_{X\in\mathbf{X}}$ and write $\mathscr{B} = (\mathcal{G}, \{p_X\}_{X\in\mathbf{X}})$ as $\mathscr{B} = (\mathcal{G}, \boldsymbol{\Theta})$. Let us consider two commonly used types of BNs, one with finite-state RVs, the other over continuous RVs.

Example 3.3 (BN over Finite-State RVs)**.** *Assume a BN $\mathscr{B}$ with finite-state RVs $\mathbf{X}$. In this case, the conditional distributions $p(X\,|\,\mathbf{pa}(X),\boldsymbol{\theta}^X)$ are naturally represented by conditional PMFs*

$$p(x\,|\,\mathbf{x_{pa}};\boldsymbol{\theta}^X) = \theta^X_{x|\mathbf{x_{pa}}}, \tag{3.4}$$

Clearly, for each X, the parameters $\boldsymbol{\theta}^X = \{\theta^X_{x|\mathbf{x_{pa}}}\}$ must satisfy

$$\theta^X_{x|\mathbf{x_{pa}}} \geq 0 \quad \forall\, x, \mathbf{x_{pa}} \tag{3.5}$$

$$\sum_{x\in\mathbf{val}(X)} \theta^X_{x|\mathbf{x_{pa}}} = 1 \quad \forall\, \mathbf{x_{pa}}. \tag{3.6}$$

The CPDs corresponding to X can be stored in a conditional probability table *(CPT) with $(|\mathbf{val}(X)|-1)\cdot|\mathbf{val}(\mathbf{pa}(X))|$ entries. The BN distribution in* (3.2) *becomes*

$$p_{\mathscr{B}}(\mathbf{x}) = \prod_{X\in\mathbf{X}} \theta^X_{\mathbf{x}_X|\mathbf{x}_{\mathbf{pa}(X)}}. \tag{3.7}$$

Example 3.4 (Conditional Gaussian Variables)**.** *In this example we assume that the RVs $\mathbf{X}$ are real-valued and that the CPDs are specified by Gaussian distributions, where the means depend linearly on the value of the parents, and using a shared variance.*

$$p(X\,|\,\mathbf{x_{pa}},\boldsymbol{\theta}^X) = \mathcal{N}\left(X \;\middle|\; (\boldsymbol{\alpha}^X)^T\,\mathbf{x_{pa}} + \beta^X, \sigma\right), \tag{3.8}$$

Here, the parameter set $\boldsymbol{\theta}^X = \{\boldsymbol{\alpha}^X, \beta^X\}$ contains the linear weights $\boldsymbol{\alpha}^X = (\alpha^X_1,\dots,\alpha^X_{|\mathbf{pa}(X)|})$ and the bias β^X. In this case the BN distribution in (3.2) *is a multivariate Gaussian distribution, with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, which are recursively given as [7]:*

$$\mu^X = \mathbb{E}[X] = (\boldsymbol{\alpha}^X)^T\,\boldsymbol{\mu}^{\mathbf{pa}(X)} + \beta^X, \tag{3.9}$$

$$\begin{aligned} \Sigma_{X,Y} &= \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] \\ &= \sum_{Z\in\mathbf{pa}(Y)} \alpha^Y_Z\,\Sigma_{X,Z} + \mathbb{1}(X=Y)\,\sigma^2. \end{aligned} \tag{3.10}$$

Here μ^X and $\boldsymbol{\mu}^{\mathbf{pa}(X)}$ are the entries and the sub-vector of $\boldsymbol{\mu}$ corresponding to X and $\mathbf{pa}(X)$, respectively. $\Sigma_{X,Y}$ is the entry in $\boldsymbol{\Sigma}$ corresponding to X and Y and α^Y_Z is the entry in $\boldsymbol{\alpha}^Y$ corresponding to parent Z of Y.

We now turn to the conditional independence assertions in the BN distribution.

3.1.2 Conditional Independence Assertions

As already illustrated in Example 3.2, the factorization properties in BNs are connected with certain conditional independence assertions in the BN distribution. Exploiting these conditional independencies is key for efficient representation, learning and inference in BNss. The factorization properties dictated by the graph directly relate to the so-called *local* conditional independencies [52]:

Theorem 3.1 (Local Conditional Independencies)**.** *Let $\mathscr{B} = (\mathcal{G}, \{p_X\})$ be a BN and $p_{\mathscr{B}}$ the joint probability defined by $\mathscr{B}$. Then, $p_{\mathscr{B}}$ satisfies the* local independencies

$$X \perp\!\!\!\perp (\mathbf{ndesc}(X) \setminus \mathbf{pa}(X)) \,|\, \mathbf{pa}(X). \tag{3.11}$$

In words, an RV X is conditional independent of its nondescendants given its parents.

Conversely, any distribution satisfying (3.11) for some graph $\mathcal{G}$ factorizes according to the BN distribution (3.2) with $\mathcal{G}$ as structure: We apply the chain rule (2.46) according to some topological ordering and eliminate as many conditioning RVs from the CPDs as permitted by (3.11), yielding (3.2).

Additionally to the local conditional independencies, several *global* conditional independencies hold in BNs, which can be read off the DAG using the notion of *d-separation*.

Definition 3.2 (d-separation)**.** *Let $\mathscr{B} = ((\mathbf{X}, \mathbf{E}), \{p_X\})$ be a BN. Two RVs $Y, Y' \in \mathbf{X}$, $Y \neq Y'$ are* d-separated *by $\mathbf{Z} \subset \mathbf{X}$, $Y, Y' \notin \mathbf{Z}$ if for all trails $\Pi = (X_1, \ldots, X_n)$ between Y and Y' there is an intermediate RV X_k, $1 < k < n$, fulfilling one the following conditions:*

- *The connection is* serial *(head-to-tail), i.e.*
  - $(X_{k-1} \to X_k), (X_k \to X_{k+1}) \in \mathbf{E}$*, or*
  - $(X_{k+1} \to X_k), (X_k \to X_{k-1}) \in \mathbf{E}$*,*

  and $X_k \in \mathbf{Z}$.
- *The connection is* diverging *(tail-to-tail), i.e.*
  - $(X_k \to X_{k-1}), (X_k \to X_{k+1}) \in \mathbf{E}$

  and $X_k \in \mathbf{Z}$.
- *the connection is* converging *(head-to-head), i.e.*
  - $(X_{k-1} \to X_k), (X_{k+1} \to X_k) \in \mathbf{E}$

  and $\mathbf{desc}(X_k) \cap \mathbf{Z} = \emptyset$*, i.e. none of the descendants of X_k (in particular X_k) is in* $\mathbf{Z}$.

Two sets $\mathbf{Y}, \mathbf{Y}' \subseteq \mathbf{X}$, $\mathbf{Y} \cap \mathbf{Y}' = \emptyset$, are d-separated *by $\mathbf{Z} \subset \mathbf{X}$, $\mathbf{Y} \cap \mathbf{Z} = \emptyset$,$\mathbf{Y}' \cap \mathbf{Z} = \emptyset$, if all $Y \in \mathbf{Y}$, $Y' \in \mathbf{Y}'$ are d-separated by $\mathbf{Z}$.*

Let us consider three canonical structures addressed in Definition 3.2: the *serial*, *diverging* and *converging* connections:

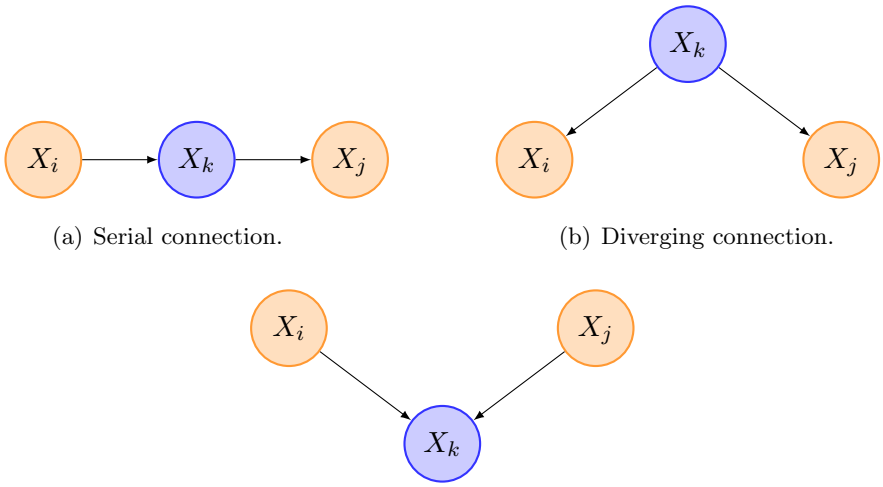
- **Serial connection.** The BN $\mathscr{B}$ shown in Figure 3.1(a) is a minimal example of a *serial connection* (head-to-tail). According to (3.2) its distribution factorizes as

  $$p_{\mathscr{B}}(X_i, X_k, X_j) = p(X_i)\, p(X_k \,|\, X_i)\, p(X_j \,|\, X_k). \tag{3.12}$$

  When we condition on X_k, we get for $p(X_k) > 0$

  $$p_{\mathscr{B}}(X_i, X_j \,|\, X_k) = \frac{\overbrace{p(X_i)\, p(X_k \,|\, X_i)}^{=p_{\mathscr{B}}(X_i, X_k)}\, p(X_j \,|\, X_k)}{p_{\mathscr{B}}(X_k)} \tag{3.13}$$

  $$= p_{\mathscr{B}}(X_i \,|\, X_k)\, p_{\mathscr{B}}(X_j \,|\, X_k), \tag{3.14}$$



(a) Serial connection.

(b) Diverging connection.

(c) Converging connection.

Figure 3.1: Three canonical examples for d-separation: (a): serial connection $X_i \to X_k \to X_j$. (b): diverging connection $X_k \to X_i$, $X_k \to X_j$. (c): converging connection $X_i \to X_k$, $X_i \to X_k$.

i.e. X_i and X_j are *conditionally independent* given X_k: $X_i \perp\!\!\!\perp X_j \,|\, X_k$. This can be interpreted as follows: when X_k is *observed*, X_i and X_j are rendered independent. On the other hand, when X_k is not observed, the RVs X_i and X_j are dependent in general, since

$$p_{\mathscr{B}}(X_i, X_j) = \sum_{x_k} p(X_i)\, p(X_k \,|\, X_i)\, p(X_j \,|\, X_k), \tag{3.15}$$

does in general *not* factorize into $p_{\mathscr{B}}(X_i)\, p_{\mathscr{B}}(X_j)$.

- **Diverging connection.** The BN $\mathscr{B}$ shown in Figure 3.1(b) is a minimal example of a *diverging connection* (tail-to-tail), factorizing as:

$$p_{\mathscr{B}}(X_i, X_k, X_j) = p(X_k)\, p(X_i \,|\, X_k)\, p(X_j \,|\, X_k). \tag{3.16}$$

Again, conditioning on X_k yields

$$p_{\mathscr{B}}(X_i, X_j \,|\, X_k) = \frac{\overbrace{p(X_k)\, p(X_i \,|\, X_k)}^{=p_{\mathscr{B}}(X_i, X_k)}\, p(X_j \,|\, X_k)}{p_{\mathscr{B}}(X_k)} \tag{3.17}$$

$$= p_{\mathscr{B}}(X_i \,|\, X_k)\, p_{\mathscr{B}}(X_j \,|\, X_k), \tag{3.18}$$

i.e. $X_i \perp\!\!\!\perp X_j \,|\, X_k$, similar as for the serial connection. Also, $p_{\mathscr{B}}(X_i, X_j)$ does in general *not* factorize into $p_{\mathscr{B}}(X_i)\, p_{\mathscr{B}}(X_j)$.

- **Converging connection.** The BN $\mathscr{B}$ shown in Figure 3.1(c) is a minimal example of a *converging connection* (head-to-head), factorizing as:

$$p_{\mathscr{B}}(X_i, X_k, X_j) = p(X_i)\, p(X_j)\, p(X_k \,|\, X_i, X_j). \tag{3.19}$$

In contrast to the two cases before, the RVs X_i and X_j are a-priori independent since

$$p_{\mathscr{B}}(X_i, X_j) = \sum_{x_k \in \mathbf{val}(X_k)} p_{\mathscr{B}}(x_k, X_i, X_j) \tag{3.20}$$

$$= p(X_i)\, p(X_j) \overbrace{\sum_{x_k \in \mathbf{val}(X_k)} p(x_k \mid X_i, X_j)}^{=1} \tag{3.21}$$

$$= p(X_i)\, p(X_j), \tag{3.22}$$

When we condition on X_k, we get

$$p_{\mathscr{B}}(X_i, X_j \mid X_k) = \frac{p(X_i)\, p(X_j)\, p(X_k \mid X_i, X_j)}{p_{\mathscr{B}}(X_k)}. \tag{3.23}$$

In general, this does not factorize into $p_{\mathscr{B}}(X_i \mid X_k)\, p_{\mathscr{B}}(X_j \mid X_k)$, i.e. X_i and X_j are independent if X_k is not observed but become in general dependent when X_k is observed. This known as *explaining away phenomenon*.

In the serial and diverging connection, the RVs X_i and X_j are rendered independent when X_k is observed, and in the converging connection, X_i and X_j are rendered independent when X_k is not observed. In these three examples, let $\mathbf{Z}$ be either $\emptyset$ or $\{X_k\}$. Then we see that in all three examples X_i and X_j are conditional independent given $\mathbf{Z}$, when they are d-separated by $\mathbf{Z}$. Is this coincidence? It is not, as the d-separation Theorem tells us [52, 70]:

Theorem 3.2 (d-separation)**.** *Let $\mathscr{B}$ be a BN with structure $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ and let $\mathbf{Y}$, $\mathbf{Y}'$, and $\mathbf{Z}$ be mutually disjoint subsets of $\mathbf{X}$. If $\mathbf{Y}$ and $\mathbf{Y}'$ are d-separated by $\mathbf{Z}$, then in $p_{\mathscr{B}}$ we have*

$$\mathbf{Y} \perp\!\!\!\perp \mathbf{Y}' \mid \mathbf{Z}. \tag{3.24}$$

In any distribution modeled by an BN with structure $\mathcal{G}$ the conditional independence assertions (3.24) hold. An interesting observation here is that these assertions are *solely* by properties of $\mathcal{G}$. They hold, no matter if the RVs are discrete or continuous, and no matter which representation the CPDs we use.

The converse of Theorem 3.2 is clearly not true, since the effective conditional independence properties of the BN distribution is a *joint effect* of structure and CPDs. As an extreme example, consider a BN over finite-state RVs, with a fully connected DAG $\mathcal{G}$ and uniform CPTs. The resulting joint distribution $p_{\mathscr{B}}$ is clearly uniform over all RVs. Moreover, all possible conditionals and marginals are also uniform, i.e. *all* possible conditional independence statements hold, since the uniform distribution can be factored according to all involved RVs. On the other hand, *no* two sets $\mathbf{Y}$ and $\mathbf{Y}'$ are d-separated by *any* set $\mathbf{Z}$, since any two RVs are connected by an edge. That is, d-separation does not encode a *single* conditional independence assertion. However, the converse is often true in a somewhat weaker sense: An example is given in [52]: when the CPDs for finite-state RVs are represented using general CPTs, then for *almost all* parametrizations $\boldsymbol{\Theta}$ the converse of Theorem 3.2 holds, i.e. a conditional independence property in $p_{\mathscr{B}}$ implies the corresponding d-separation.

3.2 Markov Networks

Similar as in BNs, an MN encodes certain factorization and conditional independence properties of the represented probability distribution. We start with the factorization properties.

3.2.1 Definition via Factorization

Definition 3.3 (Markov Network)**.** *A* Markov network *over RVs* $\mathbf{X} = \{X_1, \ldots, X_N\}$ *is a tuple* $\mathscr{M} = (\mathcal{G}, \{\Psi_{\mathbf{C}_l}\}_{l=1}^{L})$*, where* $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ *is an undirected graph with maximal cliques* $\mathbf{C}_1, \ldots, \mathbf{C}_L \subseteq \mathbf{X}$*. The* factors *or* potentials $\Psi_{\mathbf{C}_i}$ *are nonnegative functions over the maximal cliques:* $\Psi_{\mathbf{C}_l} \colon \mathbf{val}(\mathbf{C}_l) \mapsto [0, \infty)$*. To the graph* $\mathcal{G}$ *we also refer as* MN-structure*. The MN defines a probability distribution according to*

$$p_{\mathscr{M}}(\mathbf{x}) = \frac{1}{\mathscr{Z}_{\mathscr{M}}} \Phi_{\mathscr{M}}(\mathbf{x}) = \frac{1}{\mathscr{Z}_{\mathscr{M}}} \prod_{l=1}^{L} \Psi_{\mathbf{C}_l}(\mathbf{x}_{\mathbf{C}_l}), \tag{3.25}$$

where $\mathscr{Z}_{\mathscr{M}}$ *is the* normalization constant *or* partition function*, given as*

$$\mathscr{Z}_{\mathscr{M}} = \int_{\mathbf{val}(X_1)} \ldots \int_{\mathbf{val}(X_N)} \Phi_{\mathscr{M}}(x_1, \ldots, x_N) \, \mathrm{d}x_1 \ldots \mathrm{d}x_N, \tag{3.26}$$

where integrals are replaced by sums for discrete RVs. $\Phi_{\mathscr{M}}$ *is called the* unnormalized MN distribution.

The definition is illustrated in following example.

Example 3.5 (Markov Network)**.** *Consider the undirected graph shown in Figure 3.2. The maximal cliques are surrounded by dotted boxes and are given as*

$$\mathbf{C}_1 = \{X_1, X_2, X_3\}, \tag{3.27}$$
$$\mathbf{C}_2 = \{X_3, X_4\}, \text{ and} \tag{3.28}$$
$$\mathbf{C}_3 = \{X_3, X_5\}. \tag{3.29}$$

Hence, the joint probability distribution of an MN with this graph has the form

$$p_{\mathscr{M}}(X_1, \ldots, X_5) = \frac{1}{\mathscr{Z}_{\mathscr{M}}} \Psi_{\mathbf{C}_1}(X_1, X_2, X_3) \Psi_{\mathbf{C}_2}(X_3, X_4) \Psi_{\mathbf{C}_3}(X_3, X_5). \tag{3.30}$$

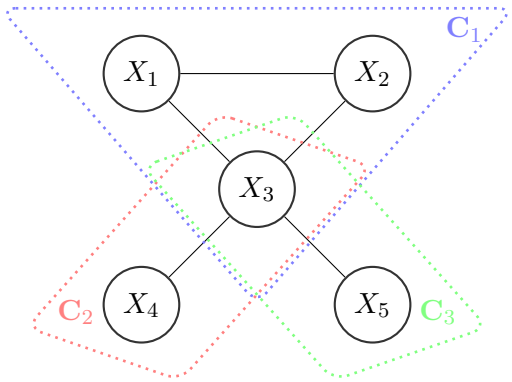


Figure 3.2: Example of an MN.

The assumption that the potentials are defined over the maximal cliques is arbitrary, but does not restrict generality. We are free to define potentials over non-maximal cliques – these, however, can be absorbed into the potential of any enclosing maximal clique. In contrast to BNs, MNs give up the immediate probabilistic interpretation of the used factors. One advantage of BNs over MNs is, that a BN distributions is readily normalized, while the partition function of MNs is a-priori unknown. Determining the partition function is generally NP-hard [52]. Thus, if

we wish to evaluate the probability of a single sample is easy in BNs, assuming that evaluation of CPDs is efficient and that the number of RVs is reasonable small. The same task can be infeasible in MNs just by not knowing the partition function. Furthermore, as discussed in the next section, BNs and MNs differ in the conditional independence assertions which can be read off from their graph.

3.2.2 Conditional Independence Assertions

Similarly as in BNs, certain conditional independence assertions can be stated for MNs. To this end, we make following definition.

Definition 3.4 (Separation)**.** *Let $\mathscr{M}$ be an MN over $\mathbf{X}$ with graph $\mathcal{G}$. Two RVs $Y, Y' \in \mathbf{X}$, $Y \neq Y'$ are* separated *by $\mathbf{Z} \subset \mathbf{X}$, $Y, Y' \notin \mathbf{Z}$, if for all paths $\Pi = (X_1, \ldots, X_n)$ between Y and Y' there is an intermediate RV $X_k \in \mathbf{Z}$. Two sets $\mathbf{Y}, \mathbf{Y}'$ are* separated *by $\mathbf{Z} \subset \mathbf{X}$, $\mathbf{Y} \cap \mathbf{Z} = \emptyset$, $\mathbf{Y}' \cap \mathbf{Z} = \emptyset$, when all $Y \in \mathbf{Y}$, $Y' \in \mathbf{Y}'$ are separated by $\mathbf{Z}$.*

Theorem 3.3. *Let $\mathscr{M}$ be an MN over $\mathbf{X}$. The following conditional independence assertions hold in $p_{\mathscr{M}}$:*

1. ***Local Markov property.*** *A RV X is conditionally independent from all other RVs given its neighbors, i.e.*

$$X \perp\!\!\!\perp \mathbf{X} \setminus (\{X\} \cup \mathbf{nb}(X)) \,|\, \mathbf{nb}(X), \tag{3.31}$$

   In this context, the neighbors X are also called the Markov blanket *of X.*

2. ***Pairwise Markov property.*** *Any two non-adjacent RVs X, Y are conditionally independent given all other RVs, i.e.*

$$X \perp\!\!\!\perp Y \,|\, \mathbf{X} \setminus \{X, Y\}, \text{ when } (X - Y) \notin E \tag{3.32}$$

3. ***Global Markov property.*** *Let $\mathbf{Y}, \mathbf{Y}', \mathbf{Z} \subset \mathbf{X}$ be mutually disjoint, where $\mathbf{Y}$ and $\mathbf{Y}'$ are separated by $\mathbf{Z}$. Then*

$$\mathbf{Y} \perp\!\!\!\perp \mathbf{Y}' \,|\, \mathbf{Z} \tag{3.33}$$

These conditional independence statements are illustrated Figure 3.3. Two simple examples demonstrate that the conditional independencies encoded in BNs and MNs differ from each other:

Example 3.6 (Conditional Independencies in BNs and MNs)**.** *Consider the BN with RVs X_1, X_2 and X_3 in Figure 3.4(a). We want to represent this BN by an MN that captures all the conditional independence properties of the BN:*

- *An appropriate MN must contain the edges $(X_1 - X_3)$ and $(X_2 - X_3)$. Otherwise, X_3 would not depend on X_2 and X_3 as in the BN. Such an MN is shown in Figure 3.4(b). In this MN it holds that $X_2 \perp\!\!\!\perp X_3 \,|\, X_1$, but not in the BN.*
- *When we additionally need to add the edge $(X_1 - X_2)$ resulting in the MN shown in Figure 3.4(c). However, now the MN does in general not satisfy that $X_1 \perp\!\!\!\perp X_2$ while the BN does.*

Consequently, the conditional independencies of the BN in Figure 3.4(a) cannot be represented by an MN.

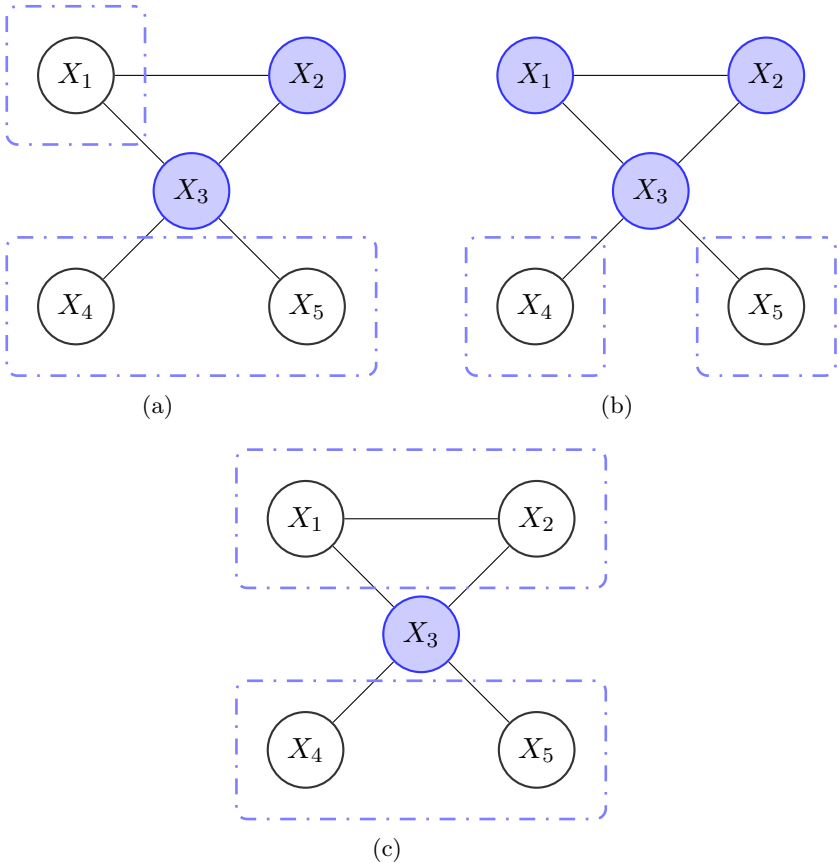


Figure 3.3: Example for conditional independencies in MNs: (a): Local Markov property: X_1 is independent from X_4 and X_5, given its Markov blanket $\{X_2, X_3\}$. (b): Pairwise Markov property: X_4 and X_5 are independent, given all other RVs. (c): Global Markov property: $\{X_1, X_2\}$ and $\{X_4, X_5\}$ are independent given $\{X_3\}$.

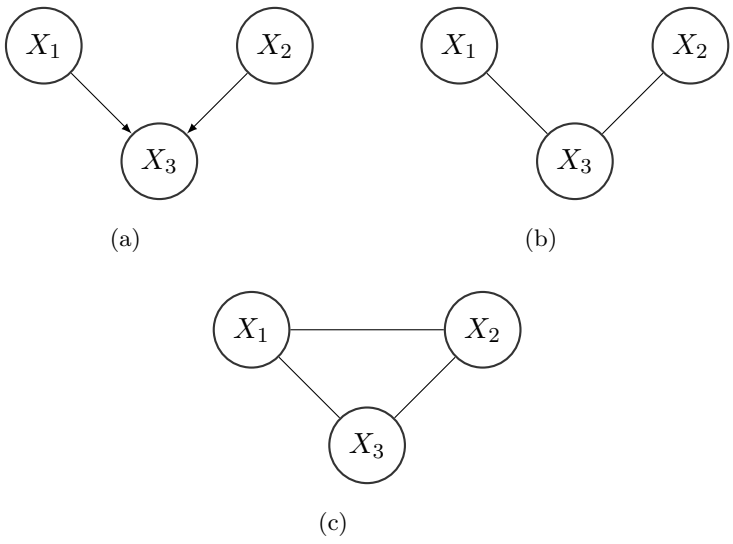


Figure 3.4: Example for a BN, whose conditional independence assertions can not be represented by an MN. (a): BN to be represented as an MN. (b), (c): Candidate MNs.

Example 3.7 (Conditional Independencies in BNs and MNs)**.** *Consider the MN in Figure 3.5. We want to represent this MN by a BN capturing all the independence properties that hold in the MN. By the pairwise Markov property we have*

$$X_1 \perp\!\!\!\perp X_4 \,|\, \{X_2, X_3\}, \text{ and} \tag{3.34}$$
$$X_2 \perp\!\!\!\perp X_3 \,|\, \{X_1, X_4\}. \tag{3.35}$$

However, these conditional independencies cannot be encoded in a BN, cf. [52].

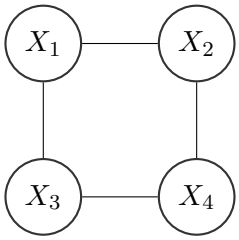


Figure 3.5: Example for an MN, whose conditional independence assertions can not be represented by a BN.

We see that BNs and MNs do not represent the same independency assertions. Nevertheless, there exist distributions which can be represented by both MNs and BNs, as for example chain structured models. We can visualize this situation using the notion of *perfect maps* (PMAPS). We call a PGM an *independency map* (IMAP) for a distribution p, when all conditional independence assertions represented by the graph of the PGM hold in p [52]. Conversely, we call such a structure a *dependency map* (DMAP), when when all conditional independencies of p hold in the graph of the PGM. A PGM is a PMAP when it is both an IMAP and a DMAP for p. Now, consider the distributions $\{p\}$ over a fixed set of RVs, let $\{p_{\mathscr{B}}\}$ be the distributions for which some BN is a perfect map and $\{p_{\mathscr{M}}\}$ be the distributions for which some MN is a perfect map. The relation of $\{p\}$, $\{p_{\mathscr{B}}\}$ and $\{p_{\mathscr{M}}\}$ is symbolically illustrated by the Venn diagram in Figure 3.6, cf. [7].

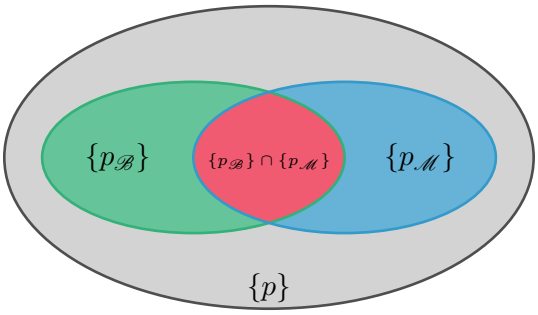


Figure 3.6: Venn diagram depicting the set of all probability distributions $\{p\}$ for a fixed set of RVs, and the sets $\{p_{\mathscr{B}}\}$ and $\{p_{\mathscr{M}}\}$, i.e. the distributions for which some BN or some MN is a perfect map, respectively.

3.3 Learning PGMs

After having specified BNs and MNs as models for probability distributions, we discuss how to come up with a model for the task under consideration. One possibility is to specify a PGM by hand. Especially BNs are suited for this approach, since the directed edges can often be interpreted as causal directions by a human domain expert. Local probability tables can be elicited

relatively easily using various techniques [70]. However, from machine learning perspective, we want to automatically determine a PGM from a set of training data..

Assume we want to find a PGM for a set of random variables $\mathbf{X}$ which are distributed according to an unknown joint distribution p^*. Further, we have a collection of L independent and identically distributed (i.i.d.) samples $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\}$, drawn from p^*. A natural goal of learning a PGM can be described as follows: given $\mathcal{D}$, return a PGM which approximates p^* best. This type of learning is known as *generative learning*, since we aim to model the process which generated the data. Generative learning can be seen as a "multi-purpose" tool, since we directly strive for the universal object describing our data: the joint distribution p^*. Other quantities, such as marginal and conditional distributions, and expectations can in principle be derived from the joint distribution.

However, it is rarely possible to exactly retrieve p^*, especially when using a finite sample $\mathcal{D}$. This can be obstructive, when the final goal is not to retrieve p^*, but to use the PGM in a specialized way, e.g. to use the PGM as *classifier*. In the classification context we are primarily interested in minimizing the loss resulting from erroneous decisions, not in modeling the overall generative process. It can be easily shown that knowledge of p^* leads to optimal classification, and therefore generative learning seems to be suitable to learn classifiers. However, a consistent observation in machine learning is that *discriminative* learning methods, which refrain from modeling p^*, but directly address the classification problem, often yield better classification results than the generative approach.

Furthermore, there are two key aspects of learning a PGM: the structure $\mathcal{G}$ and the CPDs or clique factors, which we assume to be represented by a parameter set $\boldsymbol{\Theta}$. Although $\mathcal{G}$ and $\boldsymbol{\Theta}$ are coupled, e.g. the number of parameters depends on $\mathcal{G}$, an isolated consideration of $\mathcal{G}$ and $\boldsymbol{\Theta}$ is often reasonable. This leads to the two tasks of learning PGMs: *parameter learning* and *structure learning*. Thus, we can consider many kinds of learning approaches: generative/discriminative parameter learning, generative/discriminative structure learning, combinations thereof, e.g. learning structure discriminatively and learning parameters generatively. Furthermore, there exist *hybrid* approaches, e.g. *hybrid generative-discriminative* learning objectives. More information about discriminative/hybrid learning can be found in [52, 71, 77]. In this thesis, however, we focus on the generative approach.

Let us assume a specific class of models $\mathcal{C}$, e.g. the class of all BNs, or the class of all MNs. Following a Bayesian approach to learning, we define a prior distribution $p(\mathcal{C})$ over the model class, which represents our preference for certain models. For example, if $\mathcal{C}$ is the class of all BNs, we may prefer networks with fewer edges, and therefore define a prior which decreases with the number of edges. We are interested in the posterior probability distribution over the model class, conditioned on the given data. Using Bayes' law, this distribution is

$$p(\mathcal{C} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathcal{C})\, p(\mathcal{C})}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \mathcal{C})\, p(\mathcal{C})}{\int_{\mathcal{C}} p(\mathcal{D} \mid M')\; p(M')\, \mathrm{d}M'} \propto p(\mathcal{D} \mid \mathcal{C})\, p(\mathcal{C}). \tag{3.36}$$

The data generation probability $p(\mathcal{D} \mid \mathcal{C})$ is known as *likelihood* $\mathcal{L}(\mathcal{C}; \mathcal{D})$.

There exist two main approaches to use the posterior distribution:

1. **Maximum a-posteriori (MAP) approach.** This approach aims to find the most probable model $M_{\mathrm{MAP}} \in \mathcal{C}$ for given data, i.e.

$$M_{\mathrm{MAP}} = \arg\max_{M \in \mathcal{C}} p(M \mid \mathcal{D}) \tag{3.37}$$

$$= \arg\max_{M \in \mathcal{C}} \mathcal{L}(M; \mathcal{D})\, p(M). \tag{3.38}$$

   In the MAP approach we accept the single model M_{MAP} as best explanation for the data and use it for further tasks.

2. **Bayesian model averaging.** In model averaging, we refrain to decide for a single model out of $\mathcal{C}$. Instead, we maintain the uncertainty about which model might be the "true" one and keep the whole model class $\mathcal{C}$, together with the posterior distribution $p(\mathcal{C} \mid \mathcal{D})$.

As an illustrating example, consider the probability of an unseen sample $\mathbf{x}'$, after observing $\mathcal{D}$. The MAP approach dictates that

$$p(\mathbf{x}' \mid \mathcal{D}) = p(\mathbf{x}' \mid M_{\text{MAP}}), \tag{3.39}$$

while the model averaging approach results in

$$p(\mathbf{x}' \mid \mathcal{D}) = \int_{\mathcal{C}} p(\mathbf{x}' \mid M)\, p(M \mid \mathcal{D})\, \mathrm{d}M. \tag{3.40}$$

Model averaging proceeds more carefully and treats the uncertainty by marginalizing over all possible models. They often agree in the large sample limit, since for large L the mass of the posterior $p(\mathcal{C} \mid \mathcal{D})$ is typically concentrated on a single model. For small sample sizes, model averaging often outperforms the MAP approach.

Parameter Learning

For parameter learning, we assume that the network structure $\mathcal{G}$ is fixed, maybe because a domain expert specified the independence relationships among the model variables, or because a structure learning algorithm already provided an appropriate structure. Therefore, for pure parameter learning, the model class $\mathcal{C}$ is represented by parameters $\boldsymbol{\Theta}$. In the special case when the prior distribution $p(\boldsymbol{\Theta})$ is flat, i.e. no model is preferred over the other, the posterior $p(\boldsymbol{\Theta} \mid \mathcal{D})$ is proportional to the likelihood $\mathcal{L}(\boldsymbol{\Theta}; \mathcal{D})$, and the MAP solution coincides with the classical *maximum-likelihood* (ML) solution, i.e.

$$M_{\text{MAP}} = M_{\text{ML}} = \arg\max_{M \in \boldsymbol{\Theta}} \mathcal{L}(M; \mathcal{D}). \tag{3.41}$$

When the data samples $\mathcal{D}$ are i.i.d., the likelihood is given as

$$\mathcal{L}(\boldsymbol{\Theta}; \mathcal{D}) = p(\mathcal{D} \mid \boldsymbol{\Theta}) = \prod_{l=1}^{L} p(\mathbf{x}^{(l)} \mid \boldsymbol{\Theta}). \tag{3.42}$$

It is usually more convenient to work with the *log-likelihood*

$$\log \mathcal{L}(\boldsymbol{\Theta}; \mathcal{D}) = \sum_{l=1}^{L} \log p(\mathbf{x}^{(l)} \mid \boldsymbol{\Theta}). \tag{3.43}$$

Maximizing log-likelihood is equivalent to maximizing likelihood. Intuitively, the log-likelihood is a measure of goodness of fit, i.e. a model with maximum likelihood is considered as best explanation for the training data. Furthermore, maximizing likelihood yields under rather mild assumptions a *consistent estimator* of the model parameters, i.e. if p^* lies in our model class, its parameters are recovered in the large sample limit.

Consider first the class of BNs. For parameter learning with fixed BN structure $\mathcal{G}$ and i.i.d. samples $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}\}$, the log-likelihood of a BN model is given as

$$\log \mathcal{L}(\mathscr{B}; \mathcal{D}) = \sum_{l=1}^{L} \sum_{X \in \mathbf{X}} \log p(\mathbf{x}_X^{(l)} \mid \mathbf{x}_{\mathbf{pa}(X)}^{(l)}; \boldsymbol{\theta}^X) = \sum_{X \in \mathbf{X}} \ell(\boldsymbol{\theta}^X; \mathcal{D}). \tag{3.44}$$

We see that the log-likelihood decomposes into a sum over *local terms*, one for each RV X, given

as

$$\ell(\boldsymbol{\theta}^X, \mathcal{D}) = \sum_{l=1}^{L} \log p(\mathbf{x}_X^{(l)} \mid \mathbf{x}_{\mathbf{pa}(X)}^{(l)}; \boldsymbol{\theta}^X). \tag{3.45}$$

Therefore, the overall BN log-likelihood is maximized by maximizing the individual local terms w.r.t. the local parameters $\boldsymbol{\theta}^X$.

This decomposition property simplifies generative learning in BNs and often leads to closed form solutions. In the case of finite-state RVs, see Example 3.3, the ML parameters are given by [77]:

$$\hat{\theta}^X_{x|\mathbf{x_{pa}}} = \frac{\sum_{l=1}^{L} \mathbb{1}(\mathbf{x}_X^{(l)} = x \wedge \mathbf{x}_{\mathbf{pa}(X)}^{(l)} = \mathbf{x_{pa}})}{\sum_{x' \in \mathbf{val}(X)} \sum_{l=1}^{L} \mathbb{1}(\mathbf{x}_X^{(l)} = x' \wedge \mathbf{x}_{\mathbf{pa}(X)}^{(l)} = \mathbf{x_{pa}})} = \frac{n^X_{x|\mathbf{x_{pa}}}}{\sum_{x' \in \mathbf{val}(X)} n^X_{x'|\mathbf{x_{pa}}}}, \tag{3.46}$$

where the data counts $n^X_{x|\mathbf{x_{pa}}} = \sum_{l=1}^{L} \mathbb{1}(\mathbf{x}_X^{(l)} = x \wedge \mathbf{x}_{\mathbf{pa}(X)}^{(l)} = \mathbf{x_{pa}})$ are the *sufficient statistics* for this problem. When $n^X_{x'|\mathbf{x_{pa}}} = 0$ for all x', the solution in (3.46) is not defined. In this case, however, any arbitrary CPT for the particular X and $\mathbf{x_{pa}}$ is an optimal solution. In practice, one often uses smoothed ML parameters, i.e. a small constant is added to each $n^X_{x'|\mathbf{x_{pa}}}$.

Finding ML parameters for MNs is typically harder than for BNs, and usually no closed form solution exists. When the clique factors are assumed to be *log-linear*, i.e.

$$\Psi_{\mathbf{C}_l} = \exp(\theta_l \, f_l(\mathbf{C}_l)), \tag{3.47}$$

where $f_l(\mathbf{C}_l))$ is an arbitrary *feature* function, then the log-likelihood is concave. In this case, an ML solution can be found using convex optimization, e.g. gradient methods. However, this requires computing the partition function in each iteration, which renders ML-parameter learning in MNs generally NP-hard [52].

So far, we assumed *complete data*, i.e. all values in a sample $\mathbf{x}$ are available. However, missing data scenarios are common in machine learning, where possible reasons for missing data are:

- We fail to record the values of all RVs, e.g. because of corrupted measurements.
- We have *latent* RVs in our model, because they are justified from our domain knowledge, but they can not be directly observed.
- We introduce *latent* RVs in our model, as a *modeling technique* per se. For example, in a Bayesian approach, we introduce the model parameters explicitly in the model as additional RVs. As another example, *mixture model* are often interpreted as an *augmented* model where a hypothetical latent RVs is marginalized.

Although reasons for missing data might be various, its treatment is consistent in the probabilistic framework, using the expectation-maximization (EM) algorithm [30, 65]. Let $\mathbf{Y}^{(l)}$ be the set of observed RVs in the l^{th} sample, $\mathbf{y}^{(l)}$ the corresponding observed value, and $\mathbf{Z}^{(l)} = \mathbf{X} \setminus \mathbf{Y}^{(l)}$. One step of the EM algorithm is given as

$$\boldsymbol{\Theta}^{\text{t+1}} = \arg\max_{\boldsymbol{\Theta}} \mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^t), \tag{3.48}$$

$$\mathcal{Q}(\boldsymbol{\Theta}, \boldsymbol{\Theta}^t) = \sum_{l=1}^{L} \sum_{\mathbf{z}^{(l)} \in \mathbf{val}(\mathbf{Z}^{(l)})} p(\mathbf{z}^{(l)} \mid \mathbf{y}^{(l)}; \boldsymbol{\Theta}^t) \log p(\mathbf{z}^{(l)}, \mathbf{y}^{(l)}; \boldsymbol{\Theta}). \tag{3.49}$$

$\mathcal{Q}$ is the *expected data likelihood* of $\boldsymbol{\Theta}$, where the expectation is taken with respect to the current parameters $\boldsymbol{\Theta}^t$, and has the natural interpretation of probabilistic data completion. In BNs over

finite-state RVs the EM algorithm takes a particular intuitive form [52]: In a first step, the so-called *E-step*, we compute the *expected sufficient statistics*:

$$\bar{n}^X_{x|\mathbf{x_{pa}}} = \sum_{1=l}^{L} p_{\mathscr{B}}(X, \mathbf{pa}_X \mid \mathbf{y}^{(l)}; \boldsymbol{\Theta}^t). \tag{3.50}$$

In the so-called *M-step* we treat the expected sufficient statistics as if they were actual sufficient statistic, yielding parameter updates as in (3.46), using $\bar{n}^X_{x|\mathbf{x_{pa}}}$ instead of $n^X_{x|\mathbf{x_{pa}}}$. This approach generally carries over to BNs with CPDs from the *exponential family* [52]. The EM algorithm for MNs is generally less appealing, since we generally lack of closed form ML solutions for the clique factors.

Structure Learning

There exist two main approaches to structure learning in PGMs:

1. **Constraint-based approaches.** In this approach, one aims to find a structure $\mathcal{G}$ which represents the same conditional independencies as present in the underlying distribution of data. Although theoretically sound, these methods rely on statistical independence tests performed on the data, which are usually too inaccurate in practice. Further information about this approach can be found in [52, 91, 99].

2. **Scoring-based approach.** Here one aims to find a graph which represents a suitable distribution to represent the true distribution p^*, where suitability is measured by a scoring function score($\mathcal{G}$). There are two key issues here: Defining an appropriate scoring function, and developing a search algorithm which finds a score-maximizing structure $\mathcal{G}$.

A first choice for the scoring function would be the log-likelihood, i.e. to find the structure $\mathcal{G}$ which, when equipped with ML parameters, maximizes the log-likelihood. However, it easy to see that a fully connected graph will always be a maximizer of the log-log-likelihood, leading to overfitting. As a remedy, one modifies the pure likelihood-score in order to account for model complexity. One approach is the *minimum description length* (MDL) principle [82], which aims to find the shortest, most compact representation of the data $\mathcal{D}$ within the considered model class. The MDL principle additionally accounts for the description length of the model, which is measured by the number of parameters. For example, the MDL score for BNs with discrete variables was derived as [55, 94]:

$$\mathrm{MDL}(\mathcal{G}) = -\sum_{l=1}^{L} \log p(\mathbf{x}^{(l)} \mid \mathcal{G}, \boldsymbol{\Theta}_{\mathrm{ML}}) + \frac{\log L}{2} T(\mathcal{G}), \tag{3.51}$$

where $T(\mathcal{G})$ is the number of free parameters $\boldsymbol{\Theta}$. We want to *minimize* MDL, thus it is actually a negative score.

An alternative way to avoid overfitting is delivered by a Bayesian approach. The main problem with the likelihood as score is that it is not aware of model complexity, i.e. the number of free parameters. While with more densely connected graphs $\mathcal{G}$ the dimensionality of the parameter space $\boldsymbol{\Theta}$ increases exponentially, nevertheless only a single point estimate is used from this space, i.e. the ML parameters $\boldsymbol{\Theta}_{\mathrm{ML}}$. Therefore, the ML approach can be described as overly optimistic, trusting in the single "true" parameters $\boldsymbol{\Theta}_{\mathrm{ML}}$. In contrast to the ML approach, the Bayesian approach uses the parameter space in its entirety. Introducing a prior distribution $p(\boldsymbol{\Theta})$ and

marginalizing over $\boldsymbol{\Theta}$, leads to the Bayesian score (cf. [77]):

$$\text{Bayes}(\mathcal{G}) = p(\mathcal{D}|\mathcal{G}) = \int_{\boldsymbol{\Theta}} p(\mathcal{D}|\boldsymbol{\Theta}, \mathcal{G})\, p(\boldsymbol{\Theta})\, \mathrm{d}\boldsymbol{\Theta} \tag{3.52}$$

$$= \int_{\boldsymbol{\Theta}} \prod_{l=1}^{L} p(\mathbf{x}^{(l)}|\boldsymbol{\Theta}, \mathcal{G})\, p(\boldsymbol{\Theta})\, \mathrm{d}\boldsymbol{\Theta}. \tag{3.53}$$

For BNs over finite-state RVs, using Dirichlet priors for the local parameters leads under rather mild assumptions to the closed form Bayesian-Dirichlet score (BD) [11, 22, 44]. For undirected models, the Bayesian score is usually not feasible. The large sample limit of the BD score is known as *Bayesian information criterion* (BIC) and is equivalent to MDL. BIC can also defined for MNs and takes the same form as (3.51).

A common approach for MNs is to formulate structure learning as parameter learning problem. The most prominent score is ℓ^1-penalized likelihood:

$$L_1(\boldsymbol{\Theta}) = \sum_{l=1}^{L} \log p(\mathbf{x}^{(l)} \,|\, \mathcal{G}, \boldsymbol{\Theta}_{\text{ML}}) - \lambda \,\|\boldsymbol{\Theta}\|_1. \tag{3.54}$$

This approach is actually a hybrid of parameter and structure learning: a general structure $\mathcal{G}$ is hold fixed, where minimization of $\|\boldsymbol{\Theta}\|_1$ leads to an effective sparsification of the graph, determining the effective structure.

In BNs, pure structure learning is often considered, especially when closed form parameters are available. Having chosen an appropriate scoring function, how do we find a graph $\mathcal{G}$ which maximizing it? Explicit enumeration of all DAGs is clearly infeasible, since the number of DAGs grows super-exponentially with the number of nodes [83]. All the scores discussed so far, namely log-likelihood, MDL and BIC, and (the logarithm of) the BD/BDe score can be written in the form

$$\text{score}(\mathcal{G}; \mathcal{D}) = \sum_{X \in \mathbf{X}} \text{score}_X(\mathbf{pa}(X); \mathcal{D}), \tag{3.55}$$

which means that the global network score decomposes into a sum of local scores, depending only on the *set of parents* for each RV. This property helps to improve the efficiency structure learning algorithms. Nevertheless, the problem of finding a score-maximizing BN structure $\mathcal{G}$ is NP-hard in general, even when using decomposable scores and even when the maximum number of parents is restricted to $K \geq 2$ [15, 17]. For $K = 1$, i.e. when the BN is restricted to a directed tree (or actually a directed forest), the Chow-Liu algorithm [20] learns an optimal network structure in polynomial time, maximizing the log-likelihood over the class of BNs with a directed tree structure. The class of directed trees is typically restrictive enough to avoid overfitting, i.e. the likelihood score is an appropriate choice in this case.

For learning more complicated structures, there exist a variety of approximative techniques. One of the first general search algorithms was the K2 algorithm [22] which relies on an initial variable ordering, and greedily adds parents for each node, in order to increase the BD score. A more general scheme is greedy hill-climbing (GHC) [18, 44], which does not rely on an initial variable ordering. In each GHC iteration, the current graph candidate $\mathcal{G}$ is replaced with a neighboring graph $\mathcal{G}'$ when $\text{score}(\mathcal{G}') > \text{score}(\mathcal{G})$, where neighboring graphs are resulting from the current graph when applying certain graph transformations, e.g. single edge-insertions, edge-deletions and edge-reversals. GHC can be combined with tabu-search [41] in order to escape local maxima in the search space. A further method is simulated annealing (SA) which randomly generates a neighbor solution in each step; If the new solution has higher score than the current one, it is immediately accepted, while a solution with lower score is accepted with a certain probability. While this probability is high in the beginning, leading to a large exploration over

the search space, it is successively reduced according to a cooling schedule. Other approaches consider other search spaces than the space of all DAGs: In [16] it is proposed to search over the space of equivalence classes, i.e. classes of BN structures which represent the same independence assertions. In [96], a search over possible variable orderings is proposed. This approach uses the fact, as already mentioned in [22], that for a given variable ordering the optimal structure can be found in polynomial time.

There exist also *exact* approaches, globally optimizing the BN structure. Methods based on dynamic programming [51, 88, 89] have exponential time and memory requirements, which restricts their application to approximately 30 variables. Furthermore, there are branch-and-bound methods [28, 45, 95] which cast the combinatorial structure learning problem to a relaxed continuous problem. Although branch-and-bound methods also have exponential run-time, they provide two key advantages: (i) They maintain (after some initial time) a feasible solution, i.e. they can be prematurely terminated, returning the currently best solution. (ii) They provide upper and lower bounds on the maximal score, which represents a worst-case certificate of sub-optimality. Exact methods can also be applied in the discriminative case [76].

3.4 Inference in PGMs

In the last sections we introduced the BNs and MNs as probabilistic models and discussed learning approaches to fit a PGM to given data. To process of performing reasoning with a PGM, i.e. to answer queries using our model, is generally called *inference*. Assume that p is our model distribution over RVs $\mathbf{X}$. Common inference scenarios are:

1. **Marginalization.** We wish to obtain the marginal distribution $p(\mathbf{X}^q)$ for some subset of *query* RVs $\mathbf{X}^q \subset \mathbf{X}$, marginalizing $\mathbf{X} \setminus \mathbf{X}^q = \mathbf{X}^m = \{X_1^m, \dots, X_K^m\}$, i.e. compute

$$p(\mathbf{X}^q) = \int_{\mathbf{val}(X_1^m)} \dots \int_{\mathbf{val}(X_K^m)} p(\mathbf{X}^q, x_1^m, \dots, x_K^m) \, \mathrm{d}x_1^m \dots \mathrm{d}x_K^m. \tag{3.56}$$

   In the case of discrete RVs, the integrals are replaced by sums.

2. **Conditionals.** Here $\mathbf{X}$ is split into disjoint *query* RVs $\mathbf{X}^q$, *observed* RVs $\mathbf{X}^o$ and *marginalized* RVs $\mathbf{X}^m$. The goal is to find the posterior distribution of the *query variables* $\mathbf{X}^q$ conditioned on the *observation* $\mathbf{x}^o$, i.e. compute

$$p(\mathbf{X}^q \,|\, \mathbf{x}^o) = \frac{p(\mathbf{X}^q, \mathbf{x}^o)}{p(\mathbf{x}^o)} \tag{3.57}$$

   The terms $p(\mathbf{X}^q, \mathbf{x}^o)$ and $p(\mathbf{x}^o)$ are determined by marginalization over $\mathbf{X}^m$ and $\mathbf{X}^m \cup \mathbf{X}^q$.

3. **Most probable explanation. (MPE)** Here $\mathbf{X}$ is split into *query* RVs $\mathbf{X}^q$ and *observed* RVs $\mathbf{X}^o$. The goal is to find the *most probable explanation* (MPE) for $\mathbf{X}^q$, i.e. find

$$\mathbf{x}^{q*} = \underset{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)}{\arg\max}\, p(\mathbf{x}^q \,|\, \mathbf{x}^o) = \underset{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)}{\arg\max}\, p(\mathbf{x}^q, \mathbf{x}^o) \tag{3.58}$$

4. **Maximum a-posteriori. (MAP)** Here $\mathbf{X}$ is split into *query* RVs $\mathbf{X}^q$, *observed* RVs $\mathbf{X}^o$ and *marginalized* RVs $\mathbf{X}^m = \{X_1^m, \dots, X_K^m\}$. Similar as MPE we aim to find the most

probable explanation for $\mathbf{X}^q$ given $\mathbf{X}^o$, but additionally ignore $\mathbf{X}^m$, i.e. compute

$$\mathbf{x}^{q*} = \underset{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)}{\arg\max}\ p(\mathbf{x}^q \mid \mathbf{x}^o) \tag{3.59}$$

$$= \underset{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)}{\arg\max}\ p(\mathbf{x}^q, \mathbf{x}^o) \tag{3.60}$$

$$= \underset{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)}{\arg\max} \int_{\mathbf{val}(X_1^m)} \dots \int_{\mathbf{val}(X_K^m)} p(\mathbf{x}^q, \mathbf{x}^o, x_1^m, \dots, x_K^m)\, \mathrm{d}x_1^m \dots \mathrm{d}x_K^m. \tag{3.61}$$

Unfortunately, all these inference scenarios are NP-hard in general [52]. Furthermore, MAP is inherently harder than MPE [8, 68, 69]. However, the essential aim of PGMs is to target at models which allow tractable inference. To this end one aims to design inference algorithms whose computational cost adapts to the complexity of the PGMs. An alternative approach is to give up the aim to solve an inference scenario exactly, and to resort to *approximate* inference methods.

Exact inference methods aim to use the *factorization properties* of the PGM at hand. In the case of BNs, common practice is to first transform the BN into a MN by a process called *moralization*: Any two parent of an RV are connected by an undirected edge, any directed edge is replaced by an undirected one and the CPDs of BNs are interpreted as clique potentials. The resulting MN then represents the same distribution as the BN, i.e. it is an IMAP for the BN, but obscures the typical 'explaining away' independence assertion in BNs, see also Example 3.6. An example of the moralization process is shown in Figure 3.7 (b). In this way, one can treat general purpose inference for BNs and MNs in a common framework. The basic observation is that marginalization and maximization in *tree-shaped MNs* can be solved efficiently, leading to a message passing scheme called *belief-propagation* [7, 62, 70]. The main idea is now to generalize this positive result to more general graphs, by transforming them into tree shaped models by clustering together RVs. The way this is achieved is by finding a so-called *junction tree* or *clique tree*, where each node in the tree corresponds to a clique in the original model [23, 24, 49, 58]. In order to yield consistent results, a junction tree has to fulfill the so-called *running intersection property*, meaning that when an RV is contained in two different cliques $\mathbf{C}_i$ and $\mathbf{C}_j$, then it must also be contained in each clique on the path connecting these two cliques. Such a clique tree does not necessarily exist for arbitrary graphs. However, it always exist for *chordal* graphs. So we can introduce additional edges until the graph is chordal, a process called *triangulation*. This effectively removes independence assertions from the model. A simple way to find a triangulation found is by the *elimination algorithm* [52]. Inference cost in the resulting junction tree is exponential in the size of the largest clique in the used chordal graph, thus we wish to find a triangulation which minimizes the maximal clique size. This problem, unfortunately, is NP hard, requiring heuristics to find triangulated graphs with reasonable small cliques. Finally, a junction tree can be generated from the triangulated graph, by finding the maximal cliques and the connections [52]. A junction tree of the example in Figure 3.7 is shown in Figure 3.8.

Exact inference using belief propagation on junction trees is feasible only for PGM whose graphs are not too densely connected. There are various approximate inference methods, such as *variational methods*, *loopy belief propagation* and *sampling methods* [52, 77].

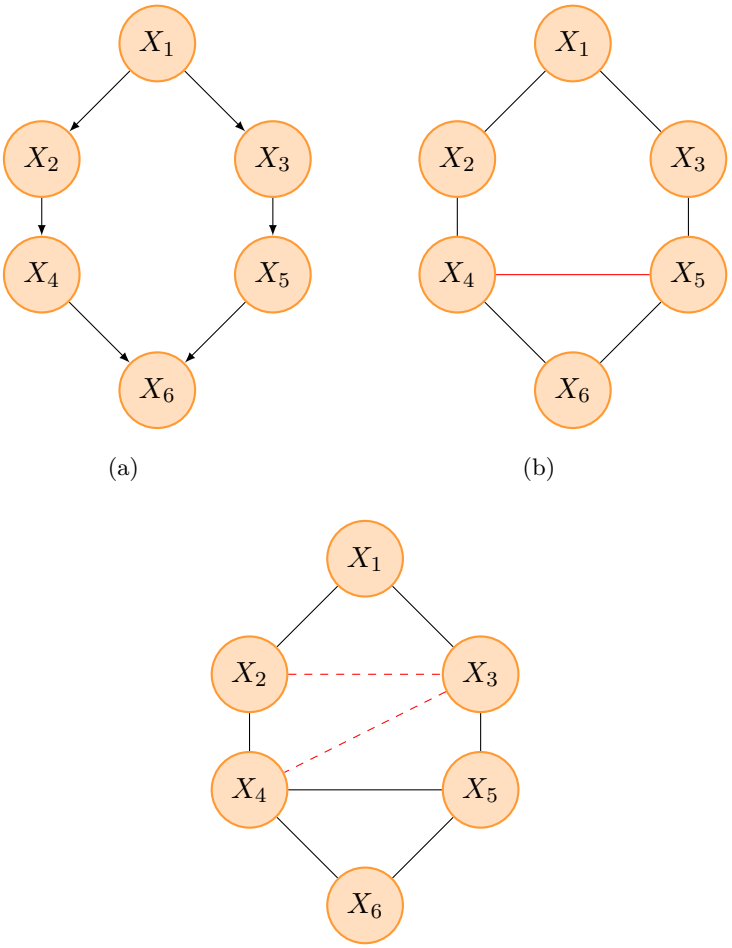


Figure 3.7: Example of moralization and triangulation. (a): BN to be moralized. (b): Moralized graph, yielding a MN; directed edge are replaced by undirected ones and an undirected edge is inserted between X_4 and X_5, which are co-parents of X_6. (c): Triangulated graph.

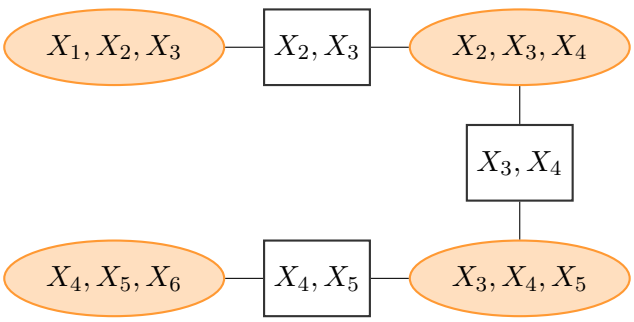


Figure 3.8: Example junction tree for the example in Figure 3.7.

4 Sum-Product Networks

Parts of this chapter are published in [75]. Similar as classic PGMs, SPNs are a graphical description of probability distributions – their syntax and semantics, however, differ largely from BNs and MNs. Before we introduce SPNs, let us point out to some weak points of classic PGMs which call for an alternative model type.

In classic PGMs, the process of *learning* and *inference* is traditionally treated in a decoupled way. This separation is somewhat artificial and myopic, since inference is often a sub-process of learning. Thus, models in which *exact* inference is intractable can often also not be properly learned. Sometimes one seeks to solve these issues using *approximate* inference for learning. This approach has several drawbacks:

- We do not know which approximate inference method will work for our task at hand. Trying out and cross-tuning several approximate inference methods is tedious work.
- Using an approximate inference method in a correct learning algorithm can yield unpredictable results. Correctness of learning is sometimes not guaranteed, when approximate inference is used.
- Approximate inference, such as sampling methods, often trade off computation time and accuracy. Finding a good trade-off for a problem at hand is usually difficult. This is also a problem in the actual inference stage.

Thus, if we want to avoid these difficulties about approximate inference, we are forced to use models in which exact inference is tractable. However, this approach will generally restrict us to sparsely connected and usually overly simplistic PGMs. In this context, PGMs appear somewhat bulky and counterintuitive from a modeler's perspective, since a model might be easily rendered intractable, just by adding a single, seemingly harmless edge.

This suggest that the conditional independence assertions in BNs and MNs are too coarse in practice, meaning that in many cases we do not want to consider conditional independencies like

$$\mathbf{Y} \perp\!\!\!\perp \mathbf{Y}' \,|\, \mathbf{X} \tag{4.1}$$

but also *context-specific independencies* (CSI) like

$$\mathbf{Y} \perp\!\!\!\perp \mathbf{Y}' \,|\, \mathbf{X} = \mathbf{x}, \tag{4.2}$$

i.e. when independence between $\mathbf{Y}$ and $\mathbf{Y}'$ hold just for certain values of $\mathbf{X}$, but not for all values. This approach is not new and several learning algorithms exploiting CSI were proposed, see for instance [10, 19]. However, CSI can not be captured in the overall DAG, but has to be encoded in the CPDs. For instance, this can done using default tables or decision trees, leading to a intransparent, nested model. These models also need specialized inference algorithms.

A simple and often used technique to design models which exhibit little or no conditional independencies among the modeled RVs $\mathbf{X}$, is to use *mixtures of distributions*. Mixtures of distributions can be interpreted as augmenting the set of RVs by a *latent* RV Z which is marginalized. This principle can be extended to multiple, hierarchically organized latent RVs, see for instance the work in [34–37, 56, 57, 102, 103].

SPNs, are a type of probabilistic model which addresses these points. In their basic description, SPNs are simply networks of sum and product operations with certain numeric inputs. Thus, they can be interpreted as a potentially deep *neural network* with two types of neurons: sums and products. SPNs in the form as discussed in this thesis admit following features:

- SPNs represent probability distributions over RVs $\mathbf{X}$.
- Many inference scenarios are stated as conceptional simple and easily implementable network evaluations.
- The computational cost of these inference scenarios scales *linearly* with the network size.
- SPNs can be interpreted as potentially deep and hierarchical structured latent RV model.

Although SPNs have been discussed in literature for some while, some aspects are not yet well understood. The main goal of this thesis is to revisit the theoretic foundations and to gain a deeper understanding about SPNs. In Section 4.2 we review an important tool related to SPNs: the network polynomial. We introduce SPNs over finite-state RVs in Section 4.3 and generalize them to discrete RVs with infinitely many states and to continuous RVs in Section 4.4. Before we start, it will be helpful to introduce some notation about representing *evidence*.

4.1 Representing Evidence

For a given set of RVs $\mathbf{X}$, the elements of $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ can be interpreted as *complete evidence*, assigning each RV in $\mathbf{X}$ a value. Given a distribution p over $\mathbf{X}$, a typical task is to evaluate the distribution for this evidence, i.e. compute $p(\mathbf{x})$.

Now assume that for a subset $\mathbf{Y} \subset \mathbf{X}$ we have complete evidence $\mathbf{y}$ available. For evaluating the probability of this evidence, we need to marginalize $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$. This can be interpreted that we have *partial evidence* for each $Z \in \mathbf{Z}$, namely that

$$Z(\omega) \in \mathbf{val}(Z) \tag{4.3}$$

This is of course an extreme case of "evidence", stating that Z assumes a value in $\mathbf{val}(Z)$, i.e. that we have no evidence about Z at all. However, this can be generalized to *partial evidence* that

$$Z(\omega) \in \mathcal{Z}, \text{ where } \mathcal{Z} \subseteq \mathbf{val}(Z). \tag{4.4}$$

To avoid problems with non-measurable sets we require $\mathcal{Z} \in \mathcal{A}_Z := \mathcal{B}(\mathbf{val}(Z))$, i.e. the Borel sets over $\mathbf{val}(Z)$, cf. Section 2.1. Recall that for countable $\mathbf{val}(Z)$, we have $\mathcal{A}_Z = 2^{\mathbf{val}(Z)}$. We use calligraphic letters to denote partial evidence, i.e. $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ is partial evidence for X, Y, Z, respectively. To represent *partial evidence about sets of RVs* $\mathbf{X} = \{X_1, \ldots, X_N\}$, we use the

product sets

$$\mathcal{H}_{\mathbf{X}} := \left\{ \bigtimes_{n=1}^{N} \mathcal{X}_n \,|\, \mathcal{X}_n \in \mathcal{A}_{X_n} \right\}. \tag{4.5}$$

Elements of $\mathcal{H}_{\mathbf{X}}, \mathcal{H}_{\mathbf{Y}}, \mathcal{H}_{\mathbf{Z}}$ are denoted as $\boldsymbol{\mathcal{X}}$, $\boldsymbol{\mathcal{Y}}$, $\boldsymbol{\mathcal{Z}}$, respectively. When $\mathbf{Y} \subseteq \mathbf{X}$, $Y \in \mathbf{X}$ and $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$, we define $\boldsymbol{\mathcal{X}}[\mathbf{Y}] := \{\mathbf{x}[\mathbf{Y}] \mid \mathbf{x} \in \boldsymbol{\mathcal{X}}\}$ and $\boldsymbol{\mathcal{X}}[Y] = \{\mathbf{x}[Y] \mid \mathbf{x} \in \boldsymbol{\mathcal{X}}\}$.

Given a distribution p over $\mathbf{X} = \{X_1, \ldots, X_N\}$, a typical task is to compute the probability of partial evidence $\boldsymbol{\mathcal{X}}$, i.e. compute

$$P(\boldsymbol{\mathcal{X}}) = \int_{\boldsymbol{\mathcal{X}}[X_1]} \ldots \int_{\boldsymbol{\mathcal{X}}[X_N]} p(x_1, \ldots, x_N)\, \mathrm{d}x_1 \ldots \mathrm{d}x_N =: \int_{\boldsymbol{\mathcal{X}}} p(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{4.6}$$

where the vector-style integral on the right-hand side of (4.6) is a shorthand. For discrete RVs, these integrals are replaced by sums. When working with both discrete and continuous RVs, we will always use integrals as in (4.6) for notational simplicity, without explicitly mentioning that they have to be replaced by sums for discrete RVs.

The most general representation of evidence used in this thesis is that $\mathbf{X}$ is split into disjoint $\mathbf{Y}, \mathbf{Y}'$, where for $\mathbf{Y}'$ we have complete evidence $\mathbf{y}'$ and for $\mathbf{Y}$ we have partial evidence $\boldsymbol{\mathcal{Y}}$. This covers several types of evidence and inference scenarios:

- $\mathbf{Y} = \emptyset$: Evaluate complete evidence $p(\mathbf{x}) = p(\mathbf{y}')$.
- $\mathbf{Y}' = \emptyset$: Evaluate partial evidence $P(\boldsymbol{\mathcal{X}}) = \int_{\boldsymbol{\mathcal{X}}} p(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_{\boldsymbol{\mathcal{Y}}} p(\mathbf{y})\, \mathrm{d}\mathbf{y}$.
- $\mathbf{Y} \neq \emptyset$, $\mathbf{Y}' \neq \emptyset$: Evaluate evidence $\int_{\boldsymbol{\mathcal{Y}}} p(\mathbf{y}, \mathbf{y}')\, \mathrm{d}\mathbf{y} = p(\mathbf{y}' \,|\, \boldsymbol{\mathcal{Y}})\, P(\mathbf{Y} \in \boldsymbol{\mathcal{Y}})$.

These types of evidence and inference scenarios also cover cases when $\mathbf{Y} \cup \mathbf{Y}' \subset \mathbf{X}$, i.e. when evidence does not include all RVs in $\mathbf{X}$ and we *marginalize* $\mathbf{X} \setminus \mathbf{Y} \cup \mathbf{Y}'$. As already pointed out in (4.3), this is just a special case of partial evidence and we simply add $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{Y} \cup \mathbf{Y}')$ to $\mathbf{Y}$ with partial evidence $\boldsymbol{\mathcal{Y}}[\mathbf{Z}] = \mathbf{val}(\mathbf{Z})$. Note that for discrete RVs it suffices to consider partial evidence only, since when all $\boldsymbol{\mathcal{X}}[X] = \{x\}$ are singletons, then $P(\boldsymbol{\mathcal{X}}) = p(\mathbf{x})$, where $\mathbf{x}$ is composed of the singletons. When working with continuous RVs, we need to consider both complete and partial evidence: complete evidence corresponds to *evaluating* a PDF, while partial evidence corresponds to *integration*.

4.2 Network Polynomials

In this section we review the notion of *network polynomials*, a description of distributions over finite-state RVs. Network polynomials were introduced in [27] for BNs, and generalized to arbitrary unnormalized distributions in [79]. Marginalization using network polynomials can be compactly expressed by setting *indicator variables* corresponding to the RVs states. The derivatives of the network polynomial deliver certain probabilistic interpretations, which is called the *differential approach to inference*.

4.2.1 Representation of Distributions as Network Polynomials

Let $\mathbf{X}$ be a set of finite-state RVs. For each RV X and each state $x \in \mathbf{val}(X)$, we introduce an *indicator variable* (IV) $\lambda_{X=x} \in \mathbb{R}$. The vector containing all IVs is denoted as $\boldsymbol{\lambda}$. Darwiche [27] defines the network polynomial as a function corresponding to a BN $\mathscr{B}$ over $\mathbf{X}$. Recall that $\mathscr{B}$

defines the probability distribution (cf. Example 3.3)

$$p_{\mathscr{B}}(\mathbf{x}) = \prod_{X \in \mathbf{X}} \theta^X_{\mathbf{x}[X],\mathbf{x}[\mathbf{pa}(X)]}. \tag{4.7}$$

The network polynomial of a BN is a multi-linear function of $\boldsymbol{\lambda}$ defined as follows.

Definition 4.1 (Network polynomial of a Bayesian network)**.** *Let $\mathscr{B}$ be a BN over finite-state RVs $\mathbf{X}$. The network polynomial $f_{\mathscr{B}}$ of $\mathscr{B}$ is defined as*

$$f_{\mathscr{B}}(\boldsymbol{\lambda}) = \sum_{\mathbf{x} \in \mathbf{val}(\mathbf{X})} \prod_{X \in \mathbf{X}} \lambda_{X=\mathbf{x}[X]}\, \theta^X_{\mathbf{x}[X],\mathbf{x}[\mathbf{pa}(X)]}. \tag{4.8}$$

As shown in [27], the network polynomial represents the BN distribution in the following sense. Let us restrict $\lambda_{X=x}$ to values $\{0, 1\}$ and define them as function of $\mathbf{x} \in \mathbf{val}(\mathbf{X})$:

$$\lambda_{X=x}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}[X] = x \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

Let $\boldsymbol{\lambda}(\mathbf{x})$ be the corresponding vector-valued function, containing all $\lambda_{X=x}(\mathbf{x})$. This $\boldsymbol{\lambda}(\mathbf{x})$ represents a 1-of-K coding of $\mathbf{x}$. Clearly, the image of function $\boldsymbol{\lambda}(\mathbf{x})$ is not the set of *all* binary vectors, i.e. there are "invalid" binary vectors not representing some $\mathbf{x}$. When we input $\boldsymbol{\lambda}(\mathbf{x})$ to the network polynomial, all but one of the exponentially many terms in the sum in (4.8) vanish and we have

$$f_{\mathscr{B}}(\boldsymbol{\lambda}(\mathbf{x})) = \prod_{X \in \mathbf{X}} \theta^X_{\mathbf{x}[X],\mathbf{x}[\mathbf{pa}(X)]} = p_{\mathscr{B}}(\mathbf{x}), \tag{4.10}$$

At a first view, the representation of the BN distribution as network polynomial seems cumbersome and not very useful. However, the network polynomial allows us to perform *marginalization* over partial evidence in a compact way. To this end, let us generalize function (4.9) to the domain $\mathcal{H}_{\mathbf{X}}$. For any $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$, let

$$\lambda_{X=x}(\boldsymbol{\mathcal{X}}) = \begin{cases} 1 & \text{if } x \in \boldsymbol{\mathcal{X}}[X] \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

Let $\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})$ be the corresponding vector-valued function, containing all $\lambda_{X=x}(\boldsymbol{\mathcal{X}})$. In contrast to $\boldsymbol{\lambda}(\mathbf{x})$, the function $\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})$ maps to *all* binary vectors and is *invertible*. That is, $\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})$ uniquely encodes the elements of $\mathcal{H}_{\mathbf{X}}$. When we input $\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})$ to the network polynomial, it evaluates to

$$f_{\mathscr{B}}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})) = \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \prod_{X \in X} \theta^X_{\mathbf{x}[X],\mathbf{x}[\mathbf{pa}(X)]} = \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} p_{\mathscr{B}}(\mathbf{x}) = P_{\mathscr{B}}\left(\boldsymbol{\mathcal{X}}\right). \tag{4.12}$$

The network polynomial marginalizes over partial evidence, by simply setting the corresponding IVs to 1. In other words, when plugging in (4.11), the network polynomial evaluates the probability measure on $\mathcal{H}_{\mathbf{X}}$. We use the shorthands $f_{\mathscr{B}}(\mathbf{x})$ and $f_{\mathscr{B}}(\boldsymbol{\mathcal{X}})$ for $f_{\mathscr{B}}(\boldsymbol{\lambda}(\mathbf{x}))$ and $f_{\mathscr{B}}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}}))$, respectively. The notion of the network polynomial can be easily generalized to arbitrary unnormalized distributions, as done in [79]:

Definition 4.2 (Network Polynomial of Unnormalized Distributions)**.** *Let Φ be an unnormalized probability distribution over discrete RVs $\mathbf{X}$, i.e. $\forall \mathbf{x} \in \mathbf{val}(\mathbf{X})\colon \Phi(\mathbf{x}) \geq 0$, $\exists \mathbf{x} \in \mathbf{val}(\mathbf{X}) : \Phi(\mathbf{x}) > 0$. The* network polynomial *f_Φ of Φ is defined as*

$$f_\Phi(\boldsymbol{\lambda}) = \sum_{\mathbf{x} \in \mathbf{val}(\mathbf{X})} \Phi(\mathbf{x}) \prod_{X \in \mathbf{X}} \lambda_{X=\mathbf{x}[X]}. \tag{4.13}$$

The network polynomial according to Definition 4.2 subsumes the one according to Definition 4.1, by setting $\Phi \equiv p_{\mathscr{B}}$. Similar as for $f_{\mathscr{B}}$, we have for $\mathbf{x} \in \mathbf{val}(\mathbf{X})$

$$f_\Phi(\boldsymbol{\lambda}(\mathbf{x})) = \Phi(\mathbf{x}), \tag{4.14}$$

and for $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$

$$f_\Phi(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})) = \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \Phi(\mathbf{x}). \tag{4.15}$$

Note that when $\boldsymbol{\lambda} \equiv 1$, $f_\Phi(\boldsymbol{\lambda})$ returns the *normalization constant* of Φ. We define $f_\Phi(\mathbf{x}) := f_\Phi(\boldsymbol{\lambda}(\mathbf{x}))$ and $f_\Phi(\boldsymbol{\mathcal{X}}) := f_\Phi(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}}))$.

Besides compactly describing marginalization, the network polynomial can be used in the so-called *differential approach to inference*, described in the next section.

4.2.2 Derivatives of the Network Polynomial

First let us consider the derivatives with respect to some IV. The network polynomial (4.13) is an exponential sum over terms

$$t_{\mathbf{x}}(\boldsymbol{\lambda}) = \Phi(\mathbf{x}) \prod_{X \in \mathbf{X}} \lambda_{X=\mathbf{x}[X]}. \tag{4.16}$$

The derivative of $t_{\mathbf{x}}(\boldsymbol{\lambda})$ is

$$\frac{\partial t_{\mathbf{x}}(\boldsymbol{\lambda})}{\partial \lambda_{X=x}} = \begin{cases} \Phi(\mathbf{x}) \prod_{X' \in \mathbf{X} \setminus \{X\}} \lambda_{X'=\mathbf{x}[X']} & \text{if } \mathbf{x}[X] = x \\ 0 & \text{otherwise.} \end{cases} \tag{4.17}$$

The derivative of the network polynomial is the sum over the derivatives of the form (4.17), which equals the evaluation of f_Φ at a modified IV vector $\boldsymbol{\lambda}'$:

$$\frac{\partial f_\Phi}{\partial \lambda_{X=x}}(\boldsymbol{\lambda}) = f_\Phi(\boldsymbol{\lambda}'), \text{ where} \tag{4.18}$$

$$\lambda'_{Y=y} = \begin{cases} \lambda_{Y=y} & \text{if } Y \neq X \\ 1 & \text{if } Y = X, y = x \\ 0 & \text{if } Y = X, y \neq x \end{cases} \tag{4.19}$$

For $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$, the derivative of the network polynomial equals

$$\frac{\partial f_\Phi}{\partial \lambda_{X=x}}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})) = \sum_{\mathbf{x}' \in \boldsymbol{\mathcal{X}}[\mathbf{X} \setminus \{X\}]} \Phi(x, \mathbf{x}'). \tag{4.20}$$

i.e. the derivatives with respect to the IVs yield evaluations for *modified evidence*, replacing $\boldsymbol{\mathcal{X}}[X]$ by $\{x\}$. In particular, for any $\mathbf{x} \in \mathbf{val}(\mathbf{X})$, we have

$$\frac{\partial f_\Phi}{\partial \lambda_{X=x}}(\boldsymbol{\lambda}(\mathbf{x})) = \Phi(x, \mathbf{x}[\mathbf{X} \setminus \{X\}]), \tag{4.21}$$

Using these derivatives, we obtain the conditional probabilities of the form

$$\Phi(x \,|\, \boldsymbol{\mathcal{X}}[\mathbf{X} \setminus \{X\}]) \propto \frac{\partial f_\Phi}{\partial \lambda_{X=x}}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})). \tag{4.22}$$

These conditional distributions, to which we refer as *marginal posteriors*, are for instance useful for approximate MAP solvers [68, 69]. Higher order derivatives with respect to the IVs have similar semantics, e.g. for the second order derivative we have

$$\frac{\partial^2 f_\Phi}{\partial \lambda_{X=x} \partial \lambda_{Y=y}}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})) = \begin{cases} \sum_{\mathbf{x}' \in \boldsymbol{\mathcal{X}}[\mathbf{X} \setminus \{X,Y\}]} \Phi(x, y, \mathbf{x}') & \text{if } X \neq Y \\ 0 & \text{otherwise.} \end{cases} \tag{4.23}$$

For network polynomials of BNs (Definition 4.1), the derivatives with respect to the network parameters also have a probabilistic interpretation [27]. Derivatives with respect to the network parameters can further be used for learning BN parameters, using gradient techniques. Furthermore, they are important for sensitivity analysis [12, 13] and for learning BNs with certain parameter constraints, such as reduced numeric precision [97].

4.2.3 Arithmetic Circuits

A direct implementation of network polynomials in the form (4.13) is not practical, due to its exponentially many summation terms, and one strives for a more compact representation. One approach uses *arithmetic circuits* [26, 27].

Definition 4.3 (Arithmetic circuit)**.** *An arithmetic circuit (AC) is a rooted acyclic directed graph, whose leaves are numeric inputs and whose internal nodes are arithmetic operations (+, −, ×, ÷) of its children. The* output of the AC *is the value computed by its root. The* size *of the AC is the number of its edges.*

The size of the AC reflects the computational cost of evaluation, since each edge corresponds to exactly one arithmetic operation. While in general ACs can represent arbitrary arithmetic functions, the focus in [26, 27] lies on representing the network polynomial $f_{\mathscr{B}}$ of some BN as a compact AC. The leaves of the AC are used to represent parameters and IVs of the network polynomial, and the arithmetic operations are restricted to additions and multiplications. It is pointed out that the junction tree algorithm can be interpreted as one particular choice for compiling a BN to an AC. This work further proposes to represent BNs via logical theories which in turn can be compiled to ACs. The approach in [64] goes one step further and directly learns an AC optimizing a scoring function, trading off the training likelihood, number of parameters, and inference complexity in terms of AC size. Basically, this algorithm learns a BN with context-specific independencies [19], where CPTs are organized as decision trees [19]. In [63], an AC is learned by optimizing a trade-off of likelihood and number of parameters, where the AC represents a Markov network over discrete variables. This method starts with an AC consisting of a product over all single variable features and iteratively introduces new features by conditioning an existing feature on all values of a "splitting" variable. For both methods [63, 64] the log-likelihood is concave in the model parameters, i.e. a global maximum likelihood solution is obtained.

Since these ACs represent a network polynomial, they can be used for efficient evaluation of any $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$ using a single upwards-pass in the network. Therefore, inference cost is directly related with the size of the AC. Furthermore, following the differential approach, one can obtain the first-order derivatives of IVs and network parameters. Differentiating a given AC can be done by *back-propagation*, where a single back-propagation pass delivers simultaneously all derivatives with respect to IVs and parameter nodes [27].

SPNs, introduced in the next section, are actually a special type of ACs, restricted to weighted sum and product operations. The weighted sums can actually be implemented by a product operation and an additional numeric input. Therefore, SPNs are not a "more powerful" computation architecture than ACs – in fact they can be seen as a subset of ACs – however, the probabilistic interpretation of SPNs are defined in a more direct manner.

4.3 Finite-State Sum-Product Networks

We start with the definition of SPNs as presented in [79], which represent probability distributions over finite-state RVs. We generalize SPNs to RVs with countably infinitely many states and continuous RVs in Section 4.4.

Definition 4.4 (Finite-State Sum-Product Network)**.** *A sum-product network $\mathcal{S} = (\mathcal{G}, \boldsymbol{w})$ over finite-state RVs $\mathbf{X} = \{X_1, \ldots, X_N\}$ is a rooted acyclic directed graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ and a set of nonnegative parameters $\boldsymbol{w}$. The nodes $V \in \mathbf{V}$ are functions mapping $\boldsymbol{\lambda}$ to $\mathbb{R}$. There are three types of nodes:*

1. *Indicators: $\lambda_{X=x}$*

2. *Sums:*

$$\mathsf{S}(\boldsymbol{\lambda}) = \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \mathsf{C}(\boldsymbol{\lambda}), \tag{4.24}$$

   where the weight $w_{\mathsf{S},\mathsf{C}} \in \boldsymbol{w}$ is associated with edge $(\mathsf{S} \to \mathsf{C}) \in \mathbf{E}$.

3. *Products:*

$$\mathsf{P}(\boldsymbol{\lambda}) = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathsf{C}(\boldsymbol{\lambda}). \tag{4.25}$$

All leaves *of $\mathcal{G}$ are IVs and all* internal *nodes are either sums or products. $\boldsymbol{w}$ is the set of all weights associated with sum nodes. The function computed by $\mathcal{S}$ is the function computed by the root and denoted as $\mathcal{S}(\boldsymbol{\lambda})$.*

We denote nodes of SPNs with λ, S, P, N, C, F where we use the following rules to ease discussion. N denotes a general SPN node without specifying its type, i.e. whether it is an IVs, a sum or a product. C and F are also general nodes, but are used to highlight their role as as children and parents of other nodes, respectively. For IVs, sums and products, we use the symbols λ, S and P, respectively. The set of all sum nodes and product nodes of an SPN $\mathcal{S}$ are denoted as $\mathbf{S}(\mathcal{S})$ and $\mathbf{P}(\mathcal{S})$, respectively. Similarly, $\boldsymbol{\lambda}(\mathcal{S})$, $\boldsymbol{\lambda}_X(\mathcal{S})$ denote the set of all IVs and the set of all IVs related to X, respectively.

The functions which can be computed or represented by SPNs are the polynomials in $\boldsymbol{\lambda}$, with nonnegative coefficients and without constant term, i.e. any SPN $\mathcal{S}$ computes a polynomial of the form

$$\begin{aligned} &\sum_i \alpha_i \prod_{\lambda \in \boldsymbol{\lambda}(\mathcal{S})} \lambda^{k_{\lambda,i}}, \text{ where} \\ \forall i\colon &\sum_{\lambda \in \boldsymbol{\lambda}(\mathcal{S})} k_{\lambda,i} \geq 1,\ k_{\lambda,i} \in \mathbb{N}_0,\ \alpha_i \geq 0. \end{aligned} \tag{4.26}$$

Conversely, for any function of this form we can easily find an SPN computing it. In particular, any network polynomial (4.13) has this form, i.e. any network polynomial can be represented by some SPN. For a trivial representation of a network polynomial we can use a single sum with $|\mathbf{val}(\mathbf{X})|$ products (monomials) $\prod_{X \in \mathbf{X}} \lambda_{X=\mathbf{x}[X]}$ as children and having $\Phi(\mathbf{x})$ as weights. This representation is a *shallow* SPN. A point of major interest is to represent network polynomials more compactly, using *deep* architectures. Similar as for network polynomials, we can plug in

the functions $\boldsymbol{\lambda}(\mathbf{x})$ and $\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})$ from (4.9) and (4.11) and define

$$\mathsf{N}(\mathbf{x}) := \mathsf{N}(\boldsymbol{\lambda}(\mathbf{x})) \tag{4.27}$$
$$\mathsf{N}(\boldsymbol{\mathcal{X}}) := \mathsf{N}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}}) \tag{4.28}$$
$$\mathcal{S}(\mathbf{x}) := \mathcal{S}(\boldsymbol{\lambda}(\mathbf{x})) \tag{4.29}$$
$$\mathcal{S}(\boldsymbol{\mathcal{X}}) := \mathcal{S}(\boldsymbol{\lambda}(\boldsymbol{\mathcal{X}})). \tag{4.30}$$

The number of *addition operations* of an SPN $\mathcal{S}$ is $A_{\mathcal{S}} := \sum_{\mathsf{S}\in\mathbf{S}(\mathcal{S})} |\mathbf{ch}(\mathsf{S})| = |\boldsymbol{w}|$ and the number of *multiplication operations* is $M_{\mathcal{S}} := \sum_{\mathsf{P}\in\mathbf{P}(\mathcal{S})}(|\mathbf{ch}(\mathsf{P})| - 1) + |\boldsymbol{w}|$. $A_{\mathcal{S}}$ and $M_{\mathcal{S}}$ are the worst-case number of required arithmetic operations to evaluate $\mathcal{S}(\boldsymbol{\lambda})$; when some $w_{\mathsf{S},\mathsf{C}}$ or some $\lambda_{X=x}$ equal zero, evaluation of $\mathcal{S}(\boldsymbol{\lambda})$ can be cheaper.

We introduce the notion of the *scope* of SPN nodes.

Definition 4.5 (Scope of SPN Node)**.** *Let $\mathcal{S}$ be an SPN over RVs $\mathbf{X}$ and N be some node in $\mathcal{S}$. The* scope *of N, denoted as $\mathbf{sc}(\mathsf{N})$, is defined as*

$$\mathbf{sc}(\mathsf{N}) = \begin{cases} \{X\} & \text{if } \mathsf{N} \text{ is some IV } \lambda_{X=x} \\ \bigcup_{\mathsf{C}\in\mathbf{ch}(\mathsf{N})} \mathbf{sc}(\mathsf{C}) & \text{otherwise} \end{cases} \tag{4.31}$$

To avoid pathological cases, we assume that for each N there exists $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ such that $\mathsf{N}(\mathbf{x}) > 0$. It is easily shown that this is equivalent to the condition that each sum node has at least one strictly positive weight. We now define the *distribution* of an SPN.

Definition 4.6 (Distribution of an SPN)**.** *Let $\mathcal{S}$ be an SPN over discrete RVs $\mathbf{X}$. The* distribution represented by $\mathcal{S}$ *is defined as*

$$p_{\mathcal{S}}(\mathbf{x}) = \frac{\mathcal{S}(\mathbf{x})}{\mathscr{Z}_{\mathcal{S}}}, \tag{4.32}$$

where

$$\mathscr{Z}_{\mathcal{S}} = \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{X})} \mathcal{S}(\mathbf{x}), \tag{4.33}$$

is the normalization constant *of the SPN.*

From Definition 4.6 we see that, in contrast to prior work, the distribution of an SPN is defined in a direct manner, by specifying a PMF.

We further define sub-SPNs.

Definition 4.7 (Sub-SPN)**.** *Let $\mathcal{S} = (\mathcal{G}, \boldsymbol{w})$ be an SPN and N be some node in $\mathcal{S}$. The* sub-SPN *rooted at N, denoted as $\mathcal{S}_{\mathsf{N}} = (\mathcal{G}_{\mathsf{N}}, \boldsymbol{w}_{\mathsf{N}})$, is the graph $\mathcal{G}_{\mathsf{N}}$ induced by $\mathbf{desc}(\mathsf{N})$, together with the set of nonnegative weights $\boldsymbol{w}_{\mathsf{N}}$ associated with outgoing sum edges in $\mathcal{G}_{\mathsf{N}}$.*

Clearly, every $\mathcal{S}_{\mathsf{N}}$ is an SPN over $\mathbf{sc}(\mathsf{N})$, so we define the distributions

$$p_{\mathsf{N}} := p_{\mathcal{S}_{\mathsf{N}}}, \tag{4.34}$$

i.e. every node in an SPN represents a distribution over $\mathbf{sc}(\mathsf{N})$.

At a first glance, SPNs and the modeled distribution do not seem very useful. Definition 4.6 can be made for arbitrary models with nonnegative output, e.g. for multi-layer perceptrons with a nonnegative activation function for the output neuron. Besides *defining* a distribution, a probabilistic model should also allow to perform efficient inference tasks. To this end, we require *valid* SPNs, which enables efficient marginalization of the SPN distribution.

4.3.1 Valid Sum-Product Networks

Validity of SPNs [79] is defined as follows.

Definition 4.8 (Validity of SPNs)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$. $\mathcal{S}$ is* valid *if for each $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$*

$$\sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} p_{\mathcal{S}}(\mathbf{x}) = \frac{\mathcal{S}(\boldsymbol{\mathcal{X}})}{\mathscr{Z}_{\mathcal{S}}} \tag{4.35}$$

A valid SPN performs marginalization in the same manner as a network polynomial (4.13). However, as we will see, a valid SPN does not necessarily compute a network polynomial. In [79], two conditions are given for guaranteeing validity of an SPN, *completeness* and *consistency*, which are defined as follows.

Definition 4.9 (Completeness)**.** *A sum node S in SPN $\mathcal{S}$ is* complete *if $\mathbf{sc}(\mathsf{C}') = \mathbf{sc}(\mathsf{C}'')$, $\forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S})$. $\mathcal{S}$ is complete if every sum node in $\mathcal{S}$ is complete.*

Definition 4.10 (Consistency)**.** *A product node P in SPN $\mathcal{S}$ is* consistent *if for every two of its children $\mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P}), \mathsf{C}' \neq \mathsf{C}''$, it holds that $\lambda_{X=x} \in \mathbf{desc}(\mathsf{C}') \Rightarrow \forall x' \neq x : \lambda_{X=x'} \notin \mathbf{desc}(\mathsf{C}'')$. $\mathcal{S}$ is consistent if every product node in $\mathcal{S}$ is consistent.*

When completeness holds for a sum node, it has the same scope as each of its children. An alternative condition to consistency is *decomposability* [79], which implies consistency.

Definition 4.11 (Decomposability)**.** *A product node P in SPN $\mathcal{S}$ is* decomposable *if for every two of its children $\mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P}), \mathsf{C}' \neq \mathsf{C}''$ it holds that $\mathbf{sc}(\mathsf{C}') \cap \mathbf{sc}(\mathsf{C}'') = \emptyset$. $\mathcal{S}$ is decomposable if every product node in $\mathcal{S}$ is decomposable.*

Decomposability requires that the scopes of the product's children do not overlap, while the notion of consistency is somewhat harder to grasp. The following definition and proposition provide an equivalent condition to consistency.

Definition 4.12 (Shared RVs)**.** *Let P be a consistent product node. The* shared *RVs of P are defined as*

$$\mathbf{Y} = \bigcup_{\substack{\mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P}) \\ \mathsf{C}' \neq \mathsf{C}''}} \mathbf{sc}(\mathsf{C}') \cap \mathbf{sc}(\mathsf{C}''), \tag{4.36}$$

i.e. the subset of $\mathbf{sc}(\mathsf{P})$ which shared by at least two children.

Proposition 4.1. *Let P be a product node and $\mathbf{Y}$ be its shared RVs. P is consistent if and only if for each $Y \in \mathbf{Y}$ there exists a* unique *$y^* \in \mathbf{val}(Y)$ with $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{P})$.*

Definition 4.13 (Consistent State of Shared RVs)**.** *Let P be a non-decomposable product node in some complete and consistent SPN, and let $\mathbf{Y}$ be the shared RVs of P. The unique $\mathbf{y}^* \in \mathbf{val}(\mathbf{Y})$ with $\lambda_{Y=\mathbf{y}^*[Y]} \in \mathbf{desc}(\mathsf{P})$, $\forall Y \in \mathbf{Y}$, is called the* consistent state *of shared RVs $\mathbf{Y}$.*

The following theorem shows that the distribution represented by a consistent product is *deterministic with respect to the shared RVs*. Furthermore, the distribution of any descendant of this product is deterministic with respect to the RVs which overlap with $\mathbf{Y}$. To prove the theorem, we need two lemmas.

Lemma 4.1. *Let N be a node in some complete and consistent SPN, $X \in \mathbf{sc}(\mathsf{N})$ and $x \in \mathbf{val}(X)$. When $\lambda_{X=x} \notin \mathbf{desc}(\mathsf{N})$, then $\forall \mathbf{x} \in \mathbf{val}(\mathbf{X})$ with $\mathbf{x}[X] = x$ we have $\mathsf{N}(\mathbf{x}) = 0$.*

Lemma 4.2. *Let p be a PMF over $\mathbf{X}$ and $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{Z} = \mathbf{X} \backslash \mathbf{Y}$ such that there exists a $\mathbf{y}^* \in \mathbf{val}(\mathbf{Y})$ with $p(\mathbf{z}, \mathbf{y}) = 0$ when $\mathbf{y} \neq \mathbf{y}^*$. Then we have $p(\mathbf{z}, \mathbf{y}) = \mathbb{1}(\mathbf{y} = \mathbf{y}^*)\, p(\mathbf{z})$.*

Theorem 4.1. *Let $\mathcal{S}$ be a complete and consistent SPN and P be a non-decomposable product in $\mathcal{S}$, $\mathbf{Y}$ be the shared RVs of P and $\mathbf{y}^*$ the consistent state of $\mathbf{Y}$. For $\mathsf{N} \in \mathbf{desc}(\mathsf{P})$ define $\mathbf{Y}^{\mathsf{N}} := \mathbf{Y} \cap \mathbf{sc}(\mathsf{N})$ and $\mathbf{X}^{\mathsf{N}} := \mathbf{sc}(\mathsf{N}) \setminus \mathbf{Y}^{\mathsf{N}}$. Then for all $\mathsf{N} \in \mathbf{desc}(\mathsf{P})$ it holds*

$$p_{\mathsf{N}} = \mathbb{1}(\mathbf{Y}^{\mathsf{N}} = \mathbf{y}^*[\mathbf{Y}^{\mathsf{N}}])\, p_{\mathsf{N}}(\mathbf{X}^{\mathsf{N}}). \tag{4.37}$$

Proof. For any $\mathsf{N} \in \mathbf{desc}(\mathsf{P})$ with $\mathbf{Y}^{\mathsf{N}} = \emptyset$, (4.37) clearly holds, so assume $\mathbf{Y}^{\mathsf{N}} \neq \emptyset$. From Proposition 4.1 we know that $\forall Y \in \mathbf{Y}\colon \lambda_{Y=\mathbf{y}^*[Y]} \in \mathbf{desc}(\mathsf{P})$ and $\forall y \neq \mathbf{y}^*[Y]\colon \lambda_{Y=y} \notin \mathbf{desc}(\mathsf{P})$. Consequently, we have for all $Y \in \mathbf{Y}^{\mathsf{N}}$ that $\lambda_{Y=\mathbf{y}^*[Y]} \in \mathbf{desc}(\mathsf{N})$ and $\forall y \neq \mathbf{y}^*[Y]\colon \lambda_{Y=y} \notin \mathbf{desc}(\mathsf{N})$. With Lemma 4.1 it follows that for all $\mathbf{x} \in \mathbf{val}(\mathbf{sc}(\mathsf{N}))$, where $\mathbf{x}[\mathbf{Y}^{\mathsf{N}}] \neq \mathbf{y}^*[\mathbf{Y}^{\mathsf{N}}]$, we have $\mathsf{N}(\mathbf{x}) = 0$ and consequently $p_{\mathsf{N}}(\mathbf{x}) = 0$. (4.37) follows with Lemma 4.2. □

Corollary 4.1. *Let $\mathcal{S}$ be a complete and consistent SPN, P be a non-decomposable product in $\mathcal{S}$, $\mathbf{Y}$ be the shared RVs of P and $\mathbf{y}^*$ the consistent state of $\mathbf{Y}$. For $\mathsf{C} \in \mathbf{ch}(\mathsf{P})$, let $\mathbf{X}^{\mathsf{C}} := \mathbf{sc}(\mathsf{C}) \setminus \mathbf{Y}$, i.e. the part of $\mathbf{sc}(\mathsf{C})$ which is exclusive to C. Then it holds that*

$$p_{\mathsf{P}} = \mathbb{1}(\mathbf{Y} = \mathbf{y}^*) \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathsf{C}(\mathbf{X}^{\mathsf{C}}). \tag{4.38}$$

A decomposable product has the intuitive interpretation of a distribution assuming independence among the scopes of its children. Corollary 4.1 shows that a consistent product assumes independence among the shared RVs $\mathbf{Y}$ and the RVs $\mathbf{X}^{\mathsf{C}}$ which are exclusive to P's children. Independence of $\mathbf{Y}$ stems from the fact that $\mathbf{Y}$ is deterministically set to the consistent state $\mathbf{y}^*$. Theorem 4.1 shows more generally that all descendants of P, which have some RVs from $\mathbf{Y}$ in their scope, are deterministically with respect to $\mathbf{y}^*$.

4.3.2 Differential Approach in Sum-Product Networks

Valid SPNs perform *marginalization* similar as network polynomials, which is essential for tractable inference. However, a simple example shows that complete and consistent SPNs in general do not compute network polynomials. The SPN depicted in Figure 4.1[3] is complete and consistent, but not decomposable, since P^1 is not decomposable. It computes the function

$$\begin{aligned}\mathcal{S}(\boldsymbol{\lambda}) = {} & w_{\mathsf{S}^1,\mathsf{P}^1}\, w_{\mathsf{S}^2,\mathsf{P}^2}\, \lambda^2_{X=x_1}\, \lambda_{Y=y_1} \\ & + w_{\mathsf{S}^1,\mathsf{P}^1}\, w_{\mathsf{S}^2,\mathsf{P}^3}\, \lambda^2_{X=x_1}\, \lambda_{Y=y_2} \\ & + w_{\mathsf{S}^1,\mathsf{P}^3}\, \lambda_{X=x_1}\, \lambda_{Y=y_2} \\ & + w_{\mathsf{S}^1,\mathsf{P}^4}\, \lambda_{X=x_2}\, \lambda_{Y=y_1},\end{aligned} \tag{4.39}$$

which is clearly no network polynomial since $\lambda_{X=x_1}$ is raised to the power of 2 in two terms. This generally happens in non-decomposable SPNs and as a consequence, *Darwiche's differential approach is not applicable to consistent, non-decomposable SPNs.* For example, consider the derivative

$$\frac{\partial \mathcal{S}}{\partial \lambda_{X=x_1}}(\boldsymbol{\lambda}) = 2\, w_{\mathsf{S}^1,\mathsf{P}^1}\, w_{\mathsf{S}^2,\mathsf{P}^2}\, \lambda_{X=x_1}\, \lambda_{Y=y_1} + 2\, w_{\mathsf{S}^1,\mathsf{P}^1}\, w_{\mathsf{S}^2,\mathsf{P}^3}\, \lambda_{X=x_1}\, \lambda_{Y=y_2} + w_{\mathsf{S}^1,\mathsf{P}^3}\, \lambda_{Y=y_2}, \tag{4.40}$$

which does *not* have the probabilistic interpretation of evaluating modified evidence, such as the derivatives of a network polynomial, cf. (4.20). However, complete and *decomposable* SPNs *do* compute network polynomials as stated in the following proposition.

Proposition 4.2. *A complete and decomposable SPN computes the network polynomial of some unnormalized distribution.*

[3] We use nodes with symbol ∘ to denote IVs.

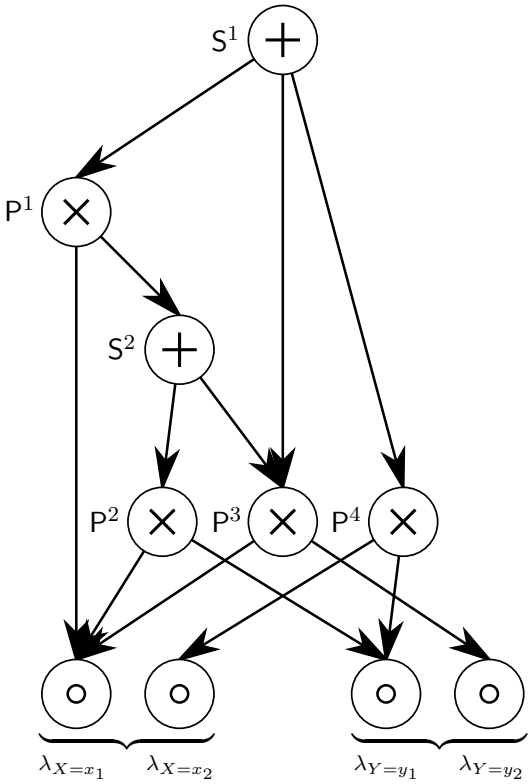


Figure 4.1: Example of a complete and consistent SPN over two binary RVs $\{X, Y\}$. The SPN is not decomposable, since P^1 has children S^2 and $\lambda_{X=x_1}$, and $\lambda_{X=x_1} \in \mathbf{desc}(\mathsf{S}^2)$.

Therefore, since complete and decomposable SPNs compute network polynomials, they are amenable for the differential approach, discussed in Section 4.2.2. As in ACs, we can use *back-propagation*, i.e. to find *all* derivatives with respect to the IVs with a *single* back-wards pass.

When N is the root, we clearly have

$$\frac{\partial \mathcal{S}}{\partial \mathsf{N}}(\boldsymbol{\lambda}) = 1, \tag{4.41}$$

since $\mathcal{S} \equiv \mathsf{N}$ per definition.

For non-roots N the derivative is given as

$$\frac{\partial \mathcal{S}}{\partial \mathsf{N}}(\boldsymbol{\lambda}) = \sum_{\mathsf{F} \in \mathbf{pa}(\mathsf{N})} \frac{\partial \mathcal{S}}{\partial \mathsf{F}} \frac{\partial \mathsf{F}}{\partial \mathsf{N}}(\boldsymbol{\lambda}). \tag{4.42}$$

The factors $\frac{\partial \mathcal{S}}{\partial \mathsf{F}}$ are recursively given by back-propagation The *local derivatives* $\frac{\partial \mathsf{F}}{\partial \mathsf{N}}$ are given as follows. When F is a sum node then

$$\mathsf{F}(\boldsymbol{\lambda}) = \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{F})} w_{\mathsf{F},\mathsf{C}}\, \mathsf{C}(\boldsymbol{\lambda}) \tag{4.43}$$

and therefore

$$\frac{\partial \mathsf{F}}{\partial \mathsf{N}}(\boldsymbol{\lambda}) = w_{\mathsf{F},\mathsf{N}}. \tag{4.44}$$

When F is a product node then

$$\mathsf{F}(\boldsymbol{\lambda}) = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{F})} \mathsf{C}(\boldsymbol{\lambda}) \tag{4.45}$$

and therefore

$$\frac{\partial \mathsf{F}}{\partial \mathsf{N}}(\boldsymbol{\lambda}) = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{F}) \setminus \{\mathsf{N}\}} \mathsf{C}(\boldsymbol{\lambda}). \tag{4.46}$$

Using a trick described in [27], some computational effort can be saved in the evaluation of (4.46). In a practical implementation of SPNs, for fixed input $\boldsymbol{\lambda}$, we evaluate the nodes from bottom to top, storing the values $\mathsf{N}(\boldsymbol{\lambda})$ for each node N. We denote these stored values as V_{N}. Now, to compute (4.46) more efficiently, we store for each product node P the product over *non-zero* children in V_{P} instead of the product over *all* children. Additionally, we store the number of children of a product node which evaluate to 0, denoted as O_{P}. If $O_{\mathsf{P}} > 0$, we know that $\mathsf{P}(\boldsymbol{\lambda}) = 0$, and otherwise $\mathsf{P}(\boldsymbol{\lambda}) = V_{\mathsf{P}}$. The derivative (4.46) is given as

$$\frac{\partial \mathsf{P}}{\partial \mathsf{N}}(\boldsymbol{\lambda}) = \begin{cases} \frac{V_{\mathsf{P}}}{V_{\mathsf{N}}} & \text{if } O_{\mathsf{P}} = 0 \\ V_{\mathsf{P}} & \text{if } O_{\mathsf{P}} = 1 \wedge V_{\mathsf{N}} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.47}$$

This trick avoids to re-compute many similar products (4.46) for the same product node during backpropagation.

The derivative with respect to some weight $w_{\mathsf{S},\mathsf{C}}$ is given as

$$\frac{\partial \mathcal{S}}{\partial w_{\mathsf{S},\mathsf{C}}}(\boldsymbol{\lambda}) = \frac{\partial \mathcal{S}}{\partial \mathsf{S}} \mathsf{C}(\boldsymbol{\lambda}). \tag{4.48}$$

These derivatives can be used for learning SPNs using gradient ascend.

For complete and decomposable SPNs, the derivatives with respect to the IVs represent evaluations of modified evidence and can be used to infer the marginal posteriors of all RVs, cf. section 4.2.2. This is an advantage to consistency, which is in general not amenable to the differential approach. On the other hand, consistency and completeness still guarantee validity and consistency is less restricting than decomposability, i.e. for a fixed number of nodes there are more *consistent* SPNs than *decomposable* SPNs. In Section 4.3.4, we compare the notions of consistency and decomposability in more detail and investigate "how much we loose" in modeling power, when using decomposable SPNs. In the following section, we investigate *normalized* SPNs, i.e. SPNs whose normalization constant equals 1.

4.3.3 Normalized Sum-Product Networks

For valid SPNs, the normalization constant $\mathscr{Z}_{\mathcal{S}} = \sum_x \mathcal{S}(\mathbf{x})$ can be determined efficiently by a single upwards pass, setting $\boldsymbol{\lambda} \equiv 1$. We call SPNs with $\mathscr{Z}_{\mathcal{S}} = 1$ *normalized* SPNs, for which we have $p_{\mathcal{S}}(\mathbf{x}) = \mathcal{S}(\mathbf{x})$. A trivial way to normalize SPNs, is to introduce an additional sum node on top of the root node, whose single weight performs the normalization.

We call SPNs *locally normalized*, when for each sum node S we have $\sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} = 1$. As pointed out in [79], complete, consistent and locally normalized SPNs are readily normalized. This is easy to see, since

- IVs are normalized, when interpreting them as distributions defined as in (4.9).
- Complete sum nodes with normalized weights are normalized when their children are

normalized, since these sum nodes represent mixture distributions.

- Consistent product nodes are normalized when their children are normalized, following from Corollary 4.1.

Clearly, any sub-SPN of a locally normalized SPNs is also locally normalized. Thus, for a complete, consistent and locally normalized SPN $\mathcal{S}$ and any N in $\mathcal{S}$, we have

$$p_{\mathsf{N}} \equiv \mathcal{S}_{\mathsf{N}} \tag{4.49}$$

The question rises, if locally normalized SPNs are a weaker class of models than non-normalized SPNs, i.e. are there distributions which can be represented by an SPN with structure $\mathcal{G}$, but *not* by a locally normalized SPN having the same structure? The answer to this question is no, as stated in the following theorem.

Algorithm 1 Normalize SPN

$\boldsymbol{w} \leftarrow \boldsymbol{w}'$
Find some topological ordering $\mathsf{N}_1, \ldots, \mathsf{N}_K$ of nodes in $\mathcal{G}$
For all product nodes P initialize $\alpha_{\mathsf{P}} \leftarrow 1$
for $k = 1 : K$ **do**
    if N_k is an IV **then**
        skip
    end if
    if N_k is a sum node **then**
        $\alpha \leftarrow \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{N}_k)} w_{\mathsf{N}_k, \mathsf{C}}$
        $\forall \mathsf{C} \in \mathbf{ch}(\mathsf{N}_k) : w_{\mathsf{N}_k, \mathsf{C}} \leftarrow \frac{w_{\mathsf{N}_k, \mathsf{C}}}{\alpha}$
    end if
    if N_k is a product node **then**
        $\alpha \leftarrow \alpha_{\mathsf{N}_k}$
        $\alpha_{\mathsf{N}_k} \leftarrow 1$
    end if
    for $\mathsf{F} \in \mathbf{pa}(\mathsf{N}_k)$ **do**
        if F is a sum node **then**
            $w_{\mathsf{F}, \mathsf{N}_k} \leftarrow \alpha \, w_{\mathsf{F}, \mathsf{N}_k}$
        end if
        if F is a product node **then**
            $\alpha_{\mathsf{F}} \leftarrow \alpha \, \alpha_{\mathsf{F}}$
        end if
    end for
end for

Theorem 4.2. *For each complete and consistent SPN $\mathcal{S}' = (\mathcal{G}', \boldsymbol{w}')$, there exists a locally normalized SPN $\mathcal{S} = (\mathcal{G}, \boldsymbol{w})$ with $\mathcal{G}' = \mathcal{G}$, such that we have $\forall \mathsf{N} \in \mathcal{G} : p_{\mathcal{S}'_{\mathsf{N}}} = \mathcal{S}_{\mathsf{N}}(\mathbf{x})$.*

Proof. Algorithm 1 constructs locally normalized parameters $\boldsymbol{w}$ without changing the distribution of any node.[4] For deriving the algorithm, we introduce a *correction factor* α_{P} for each product node P, initialized to $\alpha_{\mathsf{P}} = 1$, and redefine the product node P as $\mathsf{P}(\boldsymbol{\lambda}) := \alpha_{\mathsf{P}} \mathsf{P}(\boldsymbol{\lambda})$. At the end of the algorithm, all α_{P} will be 1 again.

Let $\mathsf{N}'_1, \ldots, \mathsf{N}'_K$ be a (reversed) topologically ordered list of all nodes in $\mathcal{S}'$, i.e. $k > l \Rightarrow \mathsf{N}'_k \notin \mathbf{desc}(\mathsf{N}'_l)$. Let $\mathsf{N}_1, \ldots, \mathsf{N}_K$ be the corresponding list of nodes in $\mathcal{S}$, which will be the locally normalized version after the algorithm has terminated. We show that we have the following loop invariant in the main loop. Given that at the k^{th} entrance of the main loop

[4] This algorithm is similar in spirit as the renormalization algorithm for discriminative BNs by Wettig et al. [101].

1. $p_{\mathsf{N}'_l} = \mathcal{S}_{\mathsf{N}_l}$, for $1 \leq l < k$
2. $\mathcal{S}_{\mathsf{N}'_m} = \mathcal{S}_{\mathsf{N}_m}$, for $k \leq m \leq K$

the same will hold for $k+1$ at the end of the loop.

The first point holds since we normalize N_k during the main loop: All nodes prior in the topological order, and therefore all children of N_k are normalized by assumption. If N_k is a sum node, then it represents a mixture distribution after step 10. If N_k is a product node, then it will be normalized after step 14, since we simply set $\alpha_{\mathsf{N}_k} \leftarrow 1$.

The second point holds since we only modify N_k, which can change any N_m, $m > k$, only via $\mathbf{pa}(\mathsf{N}_k)$. The change of N_k is compensated for all its parents either in step 18 or 21, depending on whether the parent is a sum or product node.

Points 1 and 2 clearly also hold at the first entrance ($k = 1$), since there are no nodes N_k with $k < 1$ and all higher nodes compute the unnormalized distributions, since $\boldsymbol{w} = \boldsymbol{w}'$ and all $\alpha_{\mathsf{P}} = 1$. Therefore, by induction, all nodes will compute the correctly normalized distribution after the K^{th} iteration. □

Algorithm 1 provides a way to find locally normalized parameters for any complete and consistent SPN without changing the distribution of any sub-SPN. Thus, from a representational point of view, we will assume from now on, without loss of generality, that *all* SPNs are locally normalized. Therefore, we also drop the distinction of p_{N} and $\mathcal{S}_{\mathsf{N}}$, and simply use the latter, in particular $p_{\mathcal{S}} \equiv \mathcal{S}$. Note that it might be an advantage to consider the parameter space of non-normalized parameters during *learning*. However, after learning is completed, or as a sub-routine of learning algorithms, we can use Algorithm 1 to find a set of locally normalized parameters. Locally normalized SPNs can be naturally interpreted as latent variable models, where sum weights correspond to certain conditional probabilities. This is discussed in Chapter 5.

4.3.4 Consistency Versus Decomposability

As already discussed, consistent but non-decomposable SPNs are valid, but are not amenable to the differential approach, i.e. for evaluating modified evidence for all X simultaneously, cf. (4.21). When these evaluations are not required, we might want to use the more general condition of consistency instead of decomposability, since we could possibly represent distributions more compactly, i.e. using smaller networks. An interesting question is, if there is a *significant* advantage of consistency over decomposability. The following theorem shows that this saving in network size is relatively modest.

Theorem 4.3. *Every complete and consistent SPN $\mathcal{S} = ((\mathbf{V}, \mathbf{E}), \boldsymbol{w})$ over $\mathbf{X}$ can be transformed into a complete and decomposable SPN $\mathcal{S}' = ((\mathbf{V}', \mathbf{E}'), \boldsymbol{w}')$ over $\mathbf{X}$ such that $\mathcal{S} \equiv \mathcal{S}'$, and where $|\mathbf{V}'| \in \mathcal{O}(|\mathbf{V}|^2)$, $A_{\mathcal{S}'} = A_{\mathcal{S}}$ and $M_{\mathcal{S}'} \in \mathcal{O}(M_{\mathcal{S}}\,|\mathbf{X}|)$.*

Proof. Without loss of generality we assume that $\mathcal{S}$ not contain products with just a single child – if it does, we repeatedly connect the parents of such product with its single child and remove the product, yielding a smaller SPN computing the same function. Algorithm 2 transforms $\mathcal{S}$ into a complete and decomposable SPN, representing the same distribution. First it finds a topologically ordered list $\mathsf{N}_1, \ldots, \mathsf{N}_K$ of all sum and product nodes, i.e. $k > l \Rightarrow \mathsf{N}_k \notin \mathbf{desc}(\mathsf{N}_l)$. Then, in steps 2–7, it considers all sum nodes S and all children $\mathsf{C} \in \mathbf{ch}(\mathsf{S})$; if the child C has further parents except S, a newly generated product node $\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}$ is interconnected between S and C, i.e. $\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}$ is connected as child of S with weight $w_{\mathsf{S},\mathsf{C}}$, C is disconnected from S and connected as child of $\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}$. To $\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}$ we refer as *link* between S and C. Note that the link has *only* S as parent, i.e. the link represents a *private* copy of child C for sum node S. Clearly, after step 7, the SPN still computes the same function.

Algorithm 2 Transform to decomposable SPN

1: Let $\mathbf{N} = \mathsf{N}_1, \ldots, \mathsf{N}_K$ be a topologically ordered list of all sums and products

Introduce links:
2: **for** all sum nodes S and all $\mathsf{C} \in \mathbf{ch}(\mathsf{S})$ **do**
3: **if** $\mathbf{pa}(\mathsf{C}) > 1$ **then**
4: Generate a new product node $\mathsf{P}_\mathsf{S}^\mathsf{C}$
5: Interconnect $\mathsf{P}_\mathsf{S}^\mathsf{C}$ between S and C
6: **end if**
7: **end for**

Render products decomposable:
8: **while** exist non-decomposable products in $\mathbf{N}$ **do**
9: $\mathsf{P} \leftarrow \mathsf{N}_{\min\{k' \mid \mathsf{N}_{k'} \text{ is a non-decomposable product}\}}$
10: $\mathbf{Y} \leftarrow$ shared RVs of P
11: $\mathbf{y}^* \leftarrow$ consistent state of $\mathbf{Y}$
12: **if** $\mathbf{sc}(\mathsf{P}) = \mathbf{Y}$ **then**
13: Replace P by $\prod_{Y \in \mathbf{Y}} \lambda_{Y=\mathbf{y}^*[Y]}$
14: **else**
15: $\mathbf{N}^d \leftarrow$ sums and products in $\mathbf{desc}(\mathsf{P})$
16: $\mathbf{N}^o \leftarrow \{\mathsf{N} \in \mathbf{N}^d \colon \mathbf{sc}(\mathsf{N}) \not\subseteq \mathbf{Y}, \mathbf{sc}(\mathsf{N}) \cap \mathbf{Y} \neq \emptyset\}$
17: **for** $\mathsf{N} \in \mathbf{N}^o \setminus \{\mathsf{P}\}$ **do**
18: $\mathbf{F} \leftarrow \mathbf{pa}(\mathsf{N}) \setminus \mathbf{N}^d$
19: $\forall Y \in \mathbf{Y} \cap \mathbf{sc}(\mathsf{N})\colon$ connect $\lambda_{Y=\mathbf{y}^*[Y]}$ as child of all $\mathbf{F}$
20: **end for**
21: **for** $\mathsf{P}^o \in \mathbf{N}^o$ **do**
22: Disconnect $\mathsf{C} \in \mathbf{ch}(\mathsf{P}^o)$ if $\mathbf{sc}(\mathsf{C}) \subseteq \mathbf{Y}$
23: **end for**
24: $\forall Y \in \mathbf{Y}\colon$ connect $\lambda_{Y=\mathbf{y}^*[Y]}$ as child of P
25: **end if**
26: **end while**

27: Delete all unreachable sums and products

In each iteration of the main loop 8–26, the algorithm finds the lowest non-decomposable product node $\mathsf{N}_k = \mathsf{P}$ with respect to the topological ordering. We distinguish two cases: $\mathbf{sc}(\mathsf{P}) = \mathbf{Y}$ and $\mathbf{sc}(\mathsf{P}) \neq \mathbf{Y} \Leftrightarrow \mathbf{Y} \subset \mathbf{sc}(\mathsf{P})$.

In the first case, we know from Corollary 4.1 that $\mathsf{P}(\mathbf{y}) = \mathbb{1}(\mathbf{y} = \mathbf{y}^*)$, which is equivalent to the decomposable product $\prod_{Y \in \mathbf{Y}} \lambda_{Y=\mathbf{y}^*[Y]}$ replacing P, i.e. this new product is connected as child of all parents of P, and P itself is deleted. Deletion of P might render some nodes unreachable. However, these unreachable nodes do not "influence" the root node and will be safely deleted in step 27.

In the second case, when $\mathbf{Y} \subset \mathbf{sc}(\mathsf{P})$, the algorithm first finds the set $\mathbf{N}^d$ of all sum and product descendants of P. It also finds the subset $\mathbf{N}^o$ of $\mathbf{N}^d$, containing all nodes whose scope overlaps with $\mathbf{Y}$, but is no subset of $\mathbf{Y}$. Clearly, P is contained in $\mathbf{N}^o$. The basic strategy is to "cut" $\mathbf{Y}$ from the scope of P, i.e. that $\mathbf{Y}$ is marginalized, rendering P decomposable. Then, by re-connecting all indicators $\lambda_{Y=\mathbf{y}^*[Y]}$ to P in step 24, P computes the same distribution as before due to Corollary 4.1, but is rendered *decomposable* now. Steps 21–23 cut $\mathbf{Y}$ from *all* nodes in $\mathbf{N}^o$, in particular from P, but leave all sub-SPNs rooted at any node in $\mathbf{N}^d \setminus \mathbf{N}^o$ unchanged.

To see this, note that $\mathbf{N}^d \setminus \mathbf{N}^o$ contains two types of nodes:

- nodes $\mathbf{N}^s$ whose scope is a subset of $\mathbf{Y}$, i.e. $\mathbf{N}^s = \{\mathsf{N} \in \mathbf{N}^d \,|\, \mathbf{sc}(\mathsf{N}) \subseteq \mathbf{Y}\}$, and
- nodes $\mathbf{N}^n$ whose scope does not contain any $\mathbf{Y}$, i.e. $\mathbf{N}^n = \{\mathsf{N} \in \mathbf{N}^d \mid \mathbf{sc}(\mathsf{N}) \cap \mathbf{Y} = \emptyset\}$.

Clearly, sub-SPNs rooted at any node in $\mathbf{N}^s$ or $\mathbf{N}^n$ do not contain any $\mathbf{N}^o$. Steps 21–23 only delete some outgoing edges of some $\mathsf{P}^o \in \mathbf{N}^o$; thus all sub-SPNs rooted at nodes in $\mathbf{N}^s$ or $\mathbf{N}^n$ remain unchanged, and do not change the output of the overall SPN via nodes outside of $\mathbf{N}^d$. Now consider a topologically ordered list $\mathsf{N}^o_1, \ldots, \mathsf{N}^o_L$ of the nodes in $\mathbf{N}^o$, i.e. $k > l \Rightarrow \mathsf{N}^o_k \notin \mathbf{desc}(\mathsf{N}^o_l)$. N^o_1 must be a product. If it was a sum, all its children would have the same scope as N^o_1, due to completeness. Therefore, all its children would be contained in $\mathbf{N}^o$, contradicting that N^o_1 is the first node in the topological order. Thus N^o_1 is a product. N^o_1 can have three types of children: nodes in $\mathbf{N}^s$, nodes in $\mathbf{N}^n$ and IVs. Nodes in $\mathbf{N}^s$ and IVs corresponding to some $Y \in \mathbf{Y}$ are disconnected from product N^o_1 in steps 21–23. These children make up the deterministic term $\mathbb{1}(\cdot)$ in Theorem 4.1. By disconnecting them, $\mathbf{sc}(\mathsf{N}^o_1) \cap \mathbf{Y}$ is "cut" from N^o_1, which now computes the marginal distribution over $\mathbf{sc}(\mathsf{N}^o_1) \setminus \mathbf{Y}$. This is the induction basis. Now assume that after steps 21–23 all $\mathsf{N}^o_1, \ldots, \mathsf{N}^o_l$, $l < L$, compute the marginal, $\mathbf{Y}$ being marginalized. Then also N^o_{l+1} computes such a marginal. If N^o_{l+1} is a sum and thus a mixture distribution, this clearly holds, since mixture sums and marginalization sums can be swapped, i.e. "the mixture of marginals is the marginal of the mixture". If N^o_{l+1} is a product, it can have four types of children: nodes in $\mathbf{N}^o$, nodes in $\mathbf{N}^s$, nodes in $\mathbf{N}^n$ and IVs. By induction hypothesis $\mathbf{Y}$ is already cut from all children in $\mathbf{N}^o$. Nodes in $\mathbf{N}^s$ and IVs corresponding to some $Y \in \mathbf{Y}$ are disconnected from product N^o_{l+1}. These nodes make up the deterministic term $\mathbb{1}(\cdot)$ in Theorem 4.1. A subset of $\mathbf{Y}$ might already have been removed, since $\mathbf{Y}$ has been removed from all children in $\mathbf{N}^o$. By disconnecting children in $\mathbf{N}^s$ and IVs corresponding to $\mathbf{Y}$, this deterministic term is fully removed, and N^o_{l+1} now computes the marginal distribution over $\mathbf{sc}(\mathsf{N}^o_{l+1}) \setminus \mathbf{Y}$. Thus, by induction, after steps 21–23 all nodes in $\mathbf{N}^o$ compute the marginal distribution, $\mathbf{Y}$ being marginalized.

Although we achieve our primary goal to render P decomposable, steps 21–23 also cut $\mathbf{Y}$ from any other node in $\mathsf{N} \in \mathbf{N}^o$, which would modify the SPN output via N's parents *outside* of $\mathbf{N}^d$, i.e. via $\mathbf{F} = \mathbf{pa}(\mathsf{N}) \setminus \mathbf{N}^d$. Note that all nodes in $\mathbf{F}$ must be products. To see this, assume that $\mathbf{F}$ contains a sum S. This would imply that N is a link, which can reached from P only via its single parent S. This implies $\mathsf{S} \in \mathbf{N}^d$, a contradiction. By Theorem 4.1, the distribution of N is deterministic with respect to $\mathbf{Y} \cap \mathbf{sc}(\mathsf{N})$. Steps 21–23 cut $\mathbf{Y}$ from N, which could change the distribution of the nodes in $\mathbf{F}$. Thus, in step 19 the IVs corresponding to $\mathbf{Y} \cap \mathbf{sc}(\mathsf{N})$ are connected to all $\mathbf{F}$, such that they still "see" the same distribution after steps 21–23. It is easy to see that if some $\mathsf{F} \in \mathbf{F}$ was decomposable beforehand, it will also be decomposable after step 23, i.e. steps 15–24 do not render other products non-decomposable. Thus, in each iteration, one non-decomposable product is rendered decomposable, without changing the SPN distribution.

Since the only newly introduced nodes are the links between sum nodes and their children, and the number of sum-edges is in $\mathcal{O}(|\mathbf{V}|^2)$, we have $|\mathbf{V}'| \in \mathcal{O}(|\mathbf{V}|^2)$. The number of summations is the same in $\mathcal{S}$ and $\mathcal{S}'$, i.e. $A_{\mathcal{S}'} = A_{\mathcal{S}}$, since both have the same sum nodes with the same number of children. Introducing the links in steps 2–7 does not introduce further multiplications, since the number of multiplications required by link $\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}$ is $\mathbf{ch}(\mathsf{P}^{\mathsf{C}}_{\mathsf{S}}) - 1 = 0$ after step 7. FNote that after step 7, the SPN does not contain more than $M_{\mathcal{S}}$ product nodes, since

1. Every product already present in $\mathcal{S}$ has at least 2 children and requires at least one multiplication.
2. We introduce at most as many links as sum-edges and each sum edge requires one multiplication for the sum-weights.

The while-loop in Algorithm 2, besides sometimes disconnecting nodes, only connects IVs as children of product nodes. Since every product is decomposable after Algorithm 2, and therefore consistent, the while-loop cannot more than one IV per $X \in \mathbf{X}$ to each product node. Since the

number of product nodes in $\mathcal{S}'$ is smaller than $M_{\mathcal{S}}$ and at most one IV per X can be connected as additional child of each product, we have $M_{\mathcal{S}'} \in \mathcal{O}(M_{\mathcal{S}} |\mathbf{X}|)$. □

A graphical example of Algorithm 2 is given in Figure 4.2 – see caption text for details. We see that any complete and consistent SPN $\mathcal{S}$ can be transformed into a complete and decomposable SPN $\mathcal{S}'$ with the same number of addition operations, and whose number of multiplication operations grows at most linearly with $M_{\mathcal{S}} |\mathbf{X}|$. Any distribution which can be encoded by a consistent SPN using polynomially many arithmetic operations in $|\mathbf{X}|$, can also be polynomially encoded by a decomposable SPN. Consequently, the class of consistent SPNs is *not exponentially* more compact than the class of decomposable SPNs.

Furthermore, Algorithm 2 shows that using consistent but non-decomposable products is actually *wasteful* for a particular sub-class of SPNs we denote as *sum-product trees*:

Definition 4.14 (Sum-Product Tree)**.** *A sum-product tree (SPT) is an SPN where each sum and product has at most one parent.*

Proposition 4.3. *Every complete and consistent, but non-decomposable SPT* $\mathcal{S} = ((\mathbf{V}, \mathbf{E}), \boldsymbol{w})$ *over* $\mathbf{X}$ *can be transformed into a complete and decomposable SPT* $\mathcal{S}' = ((\mathbf{V}', \mathbf{E}'), \boldsymbol{w}')$ *over* $\mathbf{X}$ *such that* $\mathcal{S} \equiv \mathcal{S}'$*, and where* $|\mathbf{V}'| \leq |\mathbf{V}|$*,* $A_{\mathcal{S}'} \leq A_{\mathcal{S}}$ *and* $M_{\mathcal{S}'} < M_{\mathcal{S}}$*.*

An example of applying Algorithm 2 to a consistent but non-decomposable SPT is given in Figure 4.3.

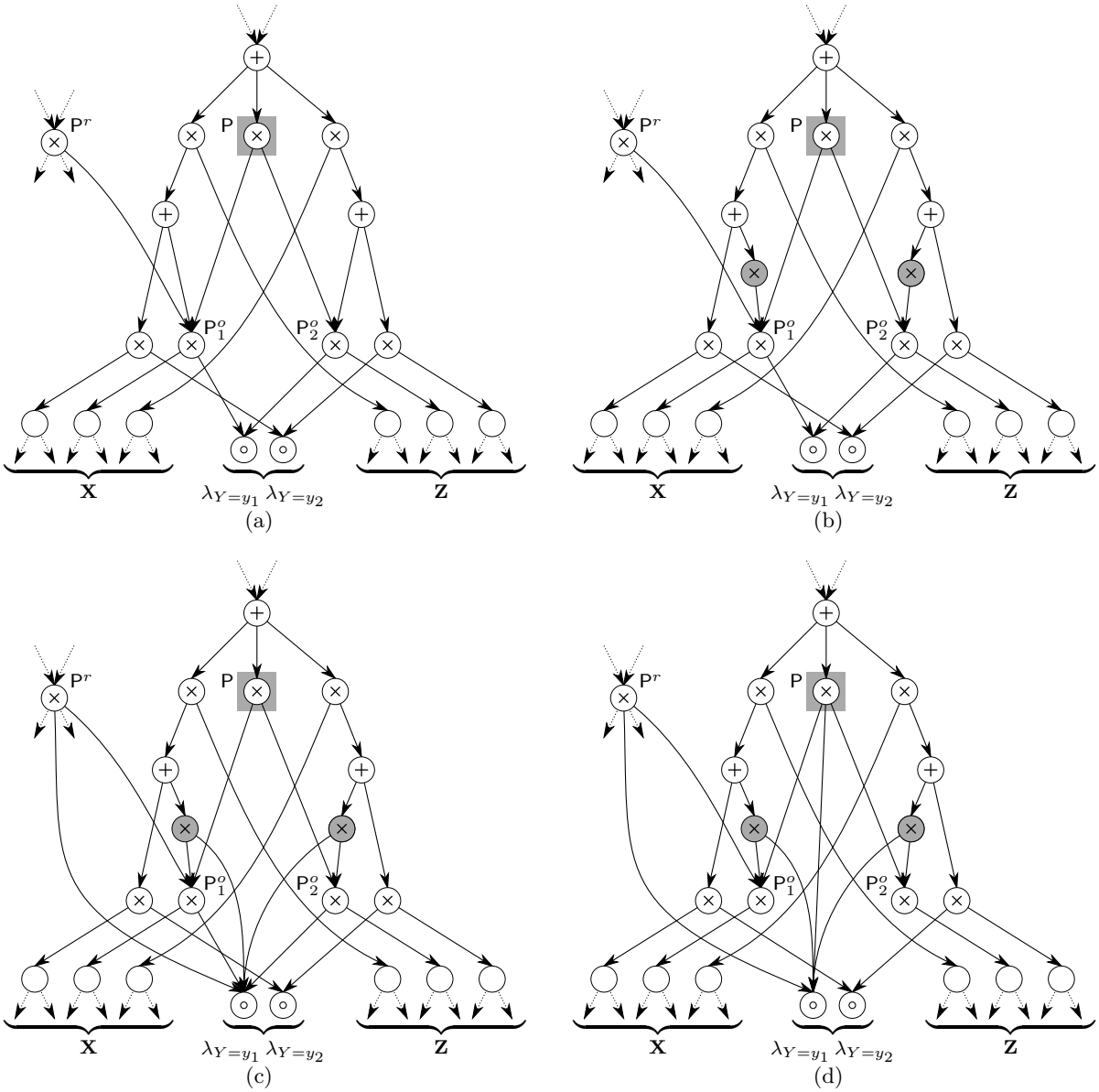


Figure 4.2: Illustration of Algorithm 2, transforming a complete and consistent SPN into a complete and decomposable one. (a): Excerpt of an SPN containing a single consistent, non-decomposable product P *with shared RVs* $\mathbf{Y} = \{Y\}$*, since* $\lambda_{Y=y_1}$ *is reached via both* P_1^o *and* P_2^o*. Dotted edges denote a continuation of the SPN outside this excerpt. The blank circles at the bottom symbolize sub-SPNs over RV sets* $\mathbf{X}$ *and* $\mathbf{Z}$*, where we assume that* $\mathbf{X} \cap \mathbf{Z} = \emptyset$*.* P*'s children* P_1^o *and* P_2^o *have both a sum node as co-parent.* P_1^o *has additionally a product* P^r *as co-parent in some remote part of the SPN. (b): Introducing links, depicted as gray product nodes, according to steps 2–7 of Algorithm 2. (c): Steps 15–20. Here, the set* $\mathbf{N}^o$ *is given as* $\mathbf{N}^o = \{\mathsf{P}, \mathsf{P}_1^o, \mathsf{P}_2^o\}$*. All parents of* $\mathbf{N}^o \setminus \{\mathsf{P}\}$ *which are not descendants of* P*, i.e. the two links and* P^r*, are connected with the IVs of the respective part of the shared RVs* $\mathbf{Y}$*, here always* $\lambda_{Y=y_1}$*. (d): Rendering* P *decomposable, steps 21–24. IV* $\lambda_{Y=y_1}$ *is disconnected from* $\mathsf{P}_1^o, \mathsf{P}_2^o$ *in steps 21–23, cutting* $\mathbf{Y}$ *from* P_1^o*,* P_2^o *and* P*. Step 24: IV* $\lambda_{Y=y_1}$ *is re-connected to* P*, which computes the same distribution as before, but is decomposable now.* P_1^o *and* P_2^o *could be short-wired and removed, since they are left with only one child. However, this does not necessarily happen and does not improve the theoretical bound of additionally required multiplications.*

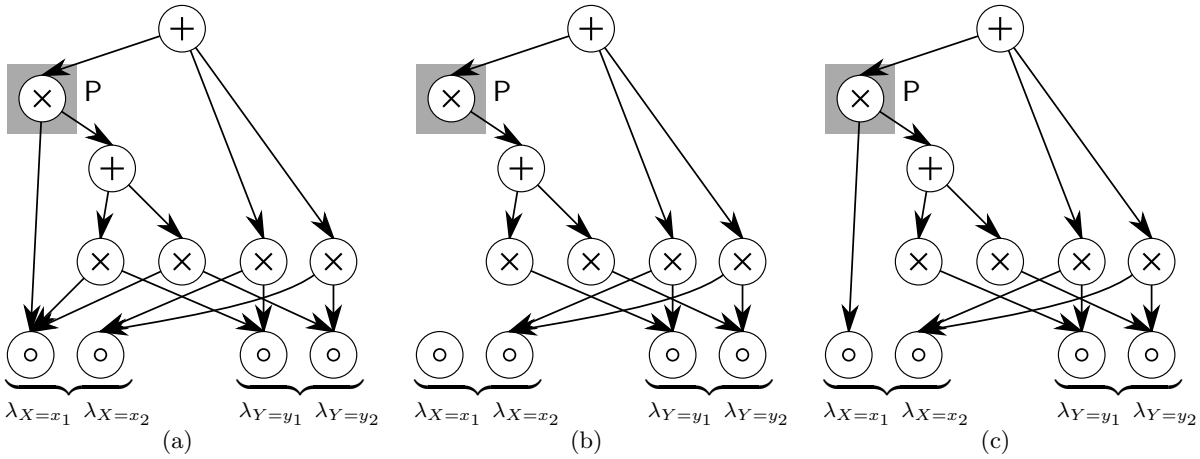


Figure 4.3: Illustration of Algorithm 2, transforming a complete and consistent, but not decomposable SPT into a complete and decomposable one. This demonstrates that using non-decomposable products is wasteful in SPTs (cf. Proposition 4.3), since after performing Algorithm 2, the SPT computes the same distribution, but requires fewer multiplications. (a): Complete and consistent SPT over two binary RVs X, Y, containing a non-decomposable product P. (b): Effect of steps 21–23 of Algorithm 2, saving 3 multiplications. (c): Effect of steps 24 of Algorithm 2, reconnecting $\lambda_{X=x_1}$ to P, re-introducing 1 multiplication.

4.4 Generalized Sum-Product Networks

So far, we considered SPNs defined as in [79], using IVs to represent the states of discrete RVs. In [40, 71], a recursive definition of SPNs was introduced, generalizing SPNs to continuous RVs and discrete RVs with infinitely many states. The basic insight is that IVs can be interpreted as discrete distribution assigning all probability mass to a single state [79]. To generalize SPNs, we replace IVs $\lambda_{X=x}$ by distributions $\mathsf{D}_{\mathbf{Y}}$, where $\mathsf{D}_{\mathbf{Y}}$ is either a PMF, a PDF or a mixed distribution over any set of RVs $\mathbf{Y}$, cf. Section 2.1. To these $\mathsf{D}_{\mathbf{Y}}$ we refer as *distribution nodes*, *input distributions* or simply as *distributions*. We define generalized SPNs as follows:

Definition 4.15 (Generalized Sum-Product Network)**.** *A sum-product network $\mathcal{S} = (\mathcal{G}, \boldsymbol{w})$ over RVs $\mathbf{X}$ is a rooted acyclic directed graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ and a set of nonnegative parameters $\boldsymbol{w}$. $\mathbf{V}$ contains three types of nodes: distribution nodes, sum nodes and product nodes. All* leaves *of $\mathcal{G}$ are distributions and all* internal *nodes are either sums or products, defined as follows:*

1. *Input distributions $\mathsf{D}_{\mathbf{Y}}$ for some $\mathbf{Y} \subseteq \mathbf{X}$. The* scope *of $\mathsf{D}_{\mathbf{Y}}$ is $\mathbf{sc}(\mathsf{D}_{\mathbf{Y}}) := \mathbf{Y}$.*

2. *Sums over nodes with identical scopes, i.e. $\mathbf{sc}(\mathsf{C}') = \mathbf{sc}(\mathsf{C}'')$, $\forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S})$:*

$$\mathsf{S} = \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \mathsf{C}, \tag{4.50}$$

   where the weight $w_{\mathsf{S},\mathsf{C}} \in \boldsymbol{w}$ is associated with the edge $(\mathsf{S} \to \mathsf{C}) \in \mathbf{E}$. The scope of a sum node is $\mathbf{sc}(\mathsf{S}) = \bigcup_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} \mathbf{sc}(\mathsf{C})$.

3. *Products over nodes with non-overlapping scopes, i.e. $\mathbf{sc}(\mathsf{C}') \cap \mathbf{sc}(\mathsf{C}'') = \emptyset$, $\forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P})$, $\mathsf{C}' \neq \mathsf{C}''$:*

$$\mathsf{P} = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathsf{C}. \tag{4.51}$$

   The scope of a product node is $\mathbf{sc}(\mathsf{P}) = \bigcup_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathbf{sc}(\mathsf{C})$.

$\boldsymbol{w}$ is the set of all weights associated with sum nodes. The distribution computed by $\mathcal{S}$ is the distribution computed by the root node.

In Definition 4.15 we already require that sums are *complete* and products are *decomposable*, see Definitions 4.9 and 4.11. Theorem 4.2 also clearly holds here, so without loss of generality we assume generalized SPNs to be locally normalized, cf. Section 4.3.3. Thus, in generalized SPNs, each node readily represents a distribution over its scope:

- Leaves are distributions per definition.
- Products are distributions assuming independence of its child distributions.
- Sums are mixture distributions.

That is, generalized SPNs represent PMFs, PDFs or mixed distributions over $\mathbf{X}$. Note that we do allow *heterogeneous* scopes of the input distributions, i.e. when we have a $\mathsf{D}_{\mathbf{Y}}$ with $Y \in \mathbf{Y}$, we can have another $\mathsf{D}_{\mathbf{Y}'}$ with $Y \in \mathbf{Y}'$ and $\mathbf{Y} \neq \mathbf{Y}'$. In the most extreme case, we can have distributions $\mathsf{D}_{\mathbf{Y}}$ for *all* possible nonempty subsets of $\mathbf{Y} \subseteq \mathbf{X}$, as long as completeness and decomposability hold. However, we require that for discrete RVs X, every $\mathsf{D}_{\mathbf{Y}}$ with $X \in \mathbf{Y}$ is a PMF with respect to X. Likewise, we require that for continuous RVs X, every $\mathsf{D}_{\mathbf{Y}}$ with $X \in \mathbf{Y}$ is a PDF with respect to X. From now on, we refer to generalized SPNs simply as SPNs. To distinguish them from SPNs according to Definition 4.4, we denote the latter as *finite-state* SPNs.

Although SPNs generalize finite-state SPNs by replacing IVs with general distributions, it will be helpful in many cases to use IVs as input distributions for finite-state RVs. This allows us to define an *extended network polynomial* and a corresponding differential approach:

Definition 4.16 (Extended network polynomial)**.** *Let $p(\mathbf{X}, \mathbf{Z})$ be a probability distribution over RVs $\mathbf{X}$, $\mathbf{Z}$, where $\mathbf{Z}$ have finitely many states. The* extended network polynomial *f_p^e is defined as*

$$f_p^e(\mathbf{X}, \boldsymbol{\lambda}) = \sum_{\mathbf{z} \in \mathbf{val}(\mathbf{Z})} p(\mathbf{X}, \mathbf{z}) \prod_{Z \in \mathbf{Z}} \lambda_{Z=\mathbf{z}[Z]}. \tag{4.52}$$

Similar as the network polynomial in (4.13), the extended network polynomial allows us to marginalize over any $\boldsymbol{\mathcal{Z}} \in \mathcal{H}_{\mathbf{Z}}$ and to apply the differential approach with respect to $\mathbf{Z}$. To see this, let us split $\mathbf{X}$ into disjoint $\mathbf{Y}$ and $\mathbf{Y}'$, where for $\mathbf{Y}$ we have partial evidence $\boldsymbol{\mathcal{Y}}$, for $\mathbf{Y}'$ we have complete evidence $\mathbf{y}'$, and evidence $\boldsymbol{\mathcal{Z}}$ for $\mathbf{Z}$, cf. Section 4.1. To evaluate this evidence for $p(\mathbf{X}, \mathbf{Z})$, we compute

$$\sum_{\mathbf{z} \in \boldsymbol{\mathcal{Z}}} \int_{\boldsymbol{\mathcal{Y}}} p(\mathbf{y}, \mathbf{y}', \mathbf{z}) \, \mathrm{d}\mathbf{y} = \int_{\boldsymbol{\mathcal{Y}}} f_p^e(\mathbf{y}, \mathbf{y}', \boldsymbol{\lambda}(\boldsymbol{\mathcal{Z}})) \, \mathrm{d}\mathbf{y}, \tag{4.53}$$

i.e. similar as the network polynomial, the extended network polynomial "takes care" about marginalization over $\boldsymbol{\mathcal{Z}}$ by simply setting the corresponding IVs. We can also apply the differential approach, e.g.

$$\frac{\partial \int_{\boldsymbol{\mathcal{Y}}} f_p^e(\mathbf{y}, \mathbf{y}', \boldsymbol{\lambda}(\boldsymbol{\mathcal{Z}})) \, \mathrm{d}\mathbf{y}}{\partial \lambda_{Z=z}} = \sum_{\mathbf{z}' \in \boldsymbol{\mathcal{Z}}[\mathbf{Z} \setminus \{Z\}]} \int_{\boldsymbol{\mathcal{Y}}} p(\mathbf{y}, \mathbf{y}', z, \mathbf{z}') \, \mathrm{d}\mathbf{y}. \tag{4.54}$$

Not surprising, when using exclusively IVs as input distributions for some set $\mathbf{Z}$ of finite-state RVs, an SPN computes an extended network polynomial:

Proposition 4.4. *Let* $\mathbf{X}$ *be a set of RVs and* $\mathbf{Z}$ *a set of finite-state RVs. An SPN* $\mathcal{S}$ *over* $\{\mathbf{X} \cup \mathbf{Z}\}$*, where for all* $Z \in \mathbf{Z}$, $z \in \mathbf{val}(Z)$ *the IVs* $\lambda_{Z=z}$ *are used as input distributions. Then* $\mathcal{S}$ *computes the extended network polynomial* $f^e_{\mathcal{S}}$*.*

Proof. The proof works similar as for Proposition 4.2, showing by induction that every node in $\mathcal{S}$ computes the extended network polynomial over the distribution represented by this node. □

We now aim to derive the same inference scenarios for SPNs as for finite-state SPNs. Inherited from the theory on network polynomials and the differential approach, there are three inference scenarios for complete and decomposable finite-state SPNs, which can be exactly solved with simple upward and/or downward passes:

1. *Evaluation of distribution* $\mathcal{S}(\mathbf{x})$ for sample $\mathbf{x} \in \mathbf{val}(\mathbf{X})$.
2. *Partial marginalization*, i.e. for $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{Y}' = \mathbf{X} \setminus \mathbf{Y}$, evaluation of
$$\int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}, \tag{4.55}$$
for partial evidence $\boldsymbol{\mathcal{Y}} \in \mathcal{H}_{\mathbf{Y}}$ and complete evidence $\mathbf{y}' \in \mathbf{val}(\mathbf{Y}')$.
3. *Evaluation of modified evidence/marginal posteriors* for all $X \in \mathbf{X}$ simultaneously, i.e. for $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{Y}' = \mathbf{X} \setminus \mathbf{Y}$, compute
$$\int_{\boldsymbol{\mathcal{Y}}[\mathbf{Y} \setminus \{X\}]} \mathcal{S}(X, \mathbf{y}, \mathbf{y}'[\mathbf{Y}' \setminus \{X\}]) \, \mathrm{d}\mathbf{y} \propto \mathcal{S}(X \,|\, \boldsymbol{\mathcal{Y}}[\mathbf{Y} \setminus \{X\}], \mathbf{y}'[\mathbf{Y}' \setminus \{X\}]), \tag{4.56}$$
for $\boldsymbol{\mathcal{Y}} \in \mathcal{H}_{\mathbf{Y}}$, $\mathbf{y}' \in \mathbf{val}(\mathbf{Y}')$.

(1.): Evaluation of $\mathcal{S}(\mathbf{x})$ works exactly the same for generalized SPNs, by simply evaluating all nodes from the leaves to the root, i.e. by one upwards pass. Note that inference scenario (2.) subsumes (1.).

(2.): Partial marginalization also works by a single upwards pass, since we can "pull" the integrals in (4.55) over all sum and product nodes, down to the input distributions. To see this, consider an arbitrary sum S for which, due to completeness, (4.55) can be written as

$$\int_{\boldsymbol{\mathcal{Y}}} \mathsf{S}(\mathbf{y}, \mathbf{y}') \, d\mathbf{y} = \int_{\boldsymbol{\mathcal{Y}}} \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} \, \mathsf{C}(\mathbf{y}, \mathbf{y}') \, d\mathbf{y} \tag{4.57}$$

$$= \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} \int_{\boldsymbol{\mathcal{Y}}} \mathsf{C}(\mathbf{y}, \mathbf{y}') \, d\mathbf{y}, \tag{4.58}$$

i.e. the integrals of a sum node equals the weighted sum of the integrals of the child nodes.

Now consider an arbitrary product node P. Due to decomposability we are allowed to swap integrals and the product, i.e. we have

$$\int_{\boldsymbol{\mathcal{Y}}} \mathsf{P}(\mathbf{y}, \mathbf{y}') \, d\mathbf{y} = \int_{\boldsymbol{\mathcal{Y}}} \prod_{k=1}^{K} \mathsf{C}_k(\mathbf{y}[\mathbf{sc}(\mathsf{C}_k)], \mathbf{y}'[\mathbf{sc}(\mathsf{C}_k)]) \, d\mathbf{y} \tag{4.59}$$

$$= \prod_{k=1}^{K} \left[\int_{\boldsymbol{\mathcal{Y}}[\mathbf{sc}(\mathsf{C}_k)]} \mathsf{C}_k(\mathbf{y}^k, \mathbf{y}'[\mathbf{sc}(\mathsf{C}_k)]) \, d\mathbf{y}^k \right] \tag{4.60}$$

We see that the integral of a product node equals the product of integrals of the child nodes, over their respective scopes. Furthermore, the same integrals are required for any co-parents of the children of S or P. Therefore, we can perform integration (4.55) by *performing the*

corresponding integrations at the input distributions over their respective scopes, and evaluating sum and product nodes in the usual way.

As a sanity check, let us see if this agrees with finite-state SPNs.

Example 4.1 (Finite-state RVs)**.** *Let* $\mathbf{X}$ *be a set of finite-state RVs and introduce for each* X*,* $x \in \mathbf{val}(X)$ *the input distributions*

$$\mathsf{D}_{X,x}(x') = \mathbb{1}(x = x'). \tag{4.61}$$

To evaluate evidence $\boldsymbol{\mathcal{Y}}$*,* $\mathbf{y}'$*, we simply perform the marginalization at the input distributions and evaluate sums and products the usual way. When* $X \in \mathbf{Y}$*, and* $\mathcal{X} = \boldsymbol{\mathcal{Y}}[X]$*, this yields*

$$\sum_{x' \in \mathcal{X}} \mathsf{D}_{X,x}(x') = \begin{cases} 1 & \text{if } x \in \mathcal{X} \\ 0 & \text{otherwise,} \end{cases} \tag{4.62}$$

which reproduces the mapping for IVs (4.11)*. When* $X \in \mathbf{Y}'$*, and* $\mathbf{y}'[X] = x'$ *we get*

$$\mathsf{D}_{X,x}(x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases} \tag{4.63}$$

which reproduces (4.9)*.*

For finite-state RVs, we yield the same result for marginalization as dictated by the theory over network polynomials. Now, let us consider continuous RVs, using single-dimensional Gaussians as input distributions.

Example 4.2 (Continuous RVs, single-dimensional Gaussians)**.** *Assume* $\mathbf{X}$ *are continuous RVs and for each* $X \in \mathbf{X}$ *we have* K *Gaussian input distributions* $\mathsf{D}_{X,1}, \ldots, \mathsf{D}_{X,K}$*, all with their individual mean and standard deviation. Let* $\mathbf{Y} \subset \mathbf{X}$ *and* $\mathbf{Y}' = \mathbf{X} \setminus \mathbf{Y}$*. To evaluate the marginal distribution of* $\mathbf{Y}'$ *for some* $\mathbf{y}'$*, we need to marginalize* $\mathbf{Y}$*, i.e. for all* $Y \in \mathbf{Y}$*,* $\boldsymbol{\mathcal{Y}}[Y] = \mathbb{R}$*. For each* Y *we get* $\int_{\boldsymbol{\mathcal{Y}}[Y]} \mathsf{D}_{Y,k} = 1$ *and each Gaussian* $\mathsf{D}_{Y',k}$ *is evaluated at* $\mathbf{y}'[Y']$*. These values serve as input to the SPN and are propagate though all sum and product nodes, yielding the desired evaluation of the marginal distribution of* $\mathbf{Y}'$*.*

Now assume more generally that we have evidence that each $Y \in \mathbf{Y}$ *takes a value in some "simple" set* $\boldsymbol{\mathcal{Y}}[Y]$*, say an interval, or the union of a few intervals. In this case we still can easily compute* $\int_{\boldsymbol{\mathcal{Y}}[Y]} \mathsf{D}_{Y,k} \leq 1$ *using the Gaussian CDF. Again evaluating* $\mathsf{D}_{Y',k}$ *for* $\mathbf{y}'[Y']$*, and propagating these values through the SPN, we get*

$$\int_{\mathbf{y} \in \boldsymbol{\mathcal{Y}}} \mathcal{S}(\mathbf{y}', \mathbf{y}) \, \mathrm{d}\mathbf{y} = \mathcal{S}(\mathbf{y}' \,|\, \mathbf{y} \in \boldsymbol{\mathcal{Y}}) \, P_{\mathcal{S}}(\mathbf{y} \in \boldsymbol{\mathcal{Y}}), \tag{4.64}$$

where P_{S} *denotes the probability measure defined by PDF* $\mathcal{S}$*, cf. Section 2.1. In particular, when* $\mathbf{Y}' = \emptyset$*, the SPN evaluates* $P_{\mathcal{S}}(\mathbf{y} \in \boldsymbol{\mathcal{Y}})$*.*

These two examples used only distributions over single RVs. We want to stress again, that this is not necessary: as long as marginalizations according to 4.55 can computed at the input distributions, the overall marginalization results from normal evaluation of the sum and product nodes. We can also be agnostic how these input distributions are represented. They might be represented as BNs, MNs, ACs [84], or again as SPNs. The latter suggests a recursive way to construct SPNs, a view which is followed in [40, 71].

(3.): Simultaneous evaluation of modified evidence/marginal posteriors for all X can be tackled using the differential approach in extended network polynomials. Let $\mathbf{X}$ be the scope of the SPN and $\mathbf{D}^X$ be the set of input distributions which have X in their scope. We define an arbitrary but fixed ordering of $\mathbf{D}^X = \{\mathsf{D}^{X,1}, \ldots, \mathsf{D}^{X,|\mathbf{D}^X|}\}$ and define $[\mathsf{D}]^X = k$ if D is the k^{th}

distribution in $\mathbf{D}^X$. For each X, we introduce a latent RV Z_X with $\mathbf{val}(Z_X) = \{1, \ldots, |\mathbf{D}^X|\}$. We denote these RVs as *distribution selectors* (DS) and let $\mathbf{Z} = \{Z_X \mid X \in \mathbf{X}\}$ be the set of all DS. Now we replace each distribution node D in the SPN by

$$\mathsf{D} \to \mathsf{D} \times \prod_{X \in \mathbf{sc}(\mathsf{D})} \lambda_{Z_X = [\mathsf{D}]^X}, \tag{4.65}$$

and denote the result as *gated* SPN. To the product introduced in (4.65) we refer as *gate* $\mathsf{P}^{X,[\mathsf{D}]^X}$. An example of constructing a gated SPN is shown in Figure 4.4.[5] In the following proposition

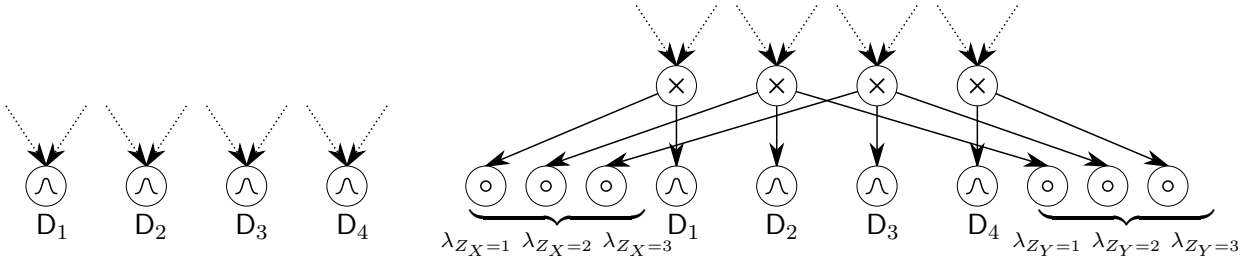


Figure 4.4: Constructing a gated SPN by introducing distribution selectors. Here we assume that $\mathbf{sc}(\mathsf{D}_1) = \{X\}$, $\mathbf{sc}(\mathsf{D}_2) = \mathbf{sc}(\mathsf{D}_3) = \{X, Y\}$ *and* $\mathbf{sc}(\mathsf{D}_4) = \{Y\}$, *and that they are the only distributions with* X *or* Y *in their scope. (a): Excerpt of an SPN showing the input distributions over* X *and* Y. *(b): Same excerpt with introduced DSs* $\lambda_{Z_X=1}, \lambda_{Z_X=2}, \lambda_{Z_X=3}, \lambda_{Z_Y=1}, \lambda_{Z_Y=2}, \lambda_{Z_Y=3}$. *The single parents of* D_1, D_2, D_3, D_4 *are the* gates $\mathsf{P}^{X,1}$, $\mathsf{P}^{X,2} = \mathsf{P}^{Y,1}$, $\mathsf{P}^{X,3} = \mathsf{P}^{Y,2}$, $\mathsf{P}^{Y,3}$, *respectively.*

we note some properties about gated SPNs.

Proposition 4.5. *Let* $\mathcal{S}$ *be an SPN and* $\mathcal{S}^g$ *be its gated SPN. Then*

1. $\mathcal{S}^g$ *is an SPN over* $\mathbf{X} \cup \mathbf{Z}$.
2. $\mathcal{S}(\mathbf{X}) = \mathcal{S}^g(\mathbf{X}) = \sum_{\mathbf{z} \in \mathbf{val}(\mathbf{Z})} \mathcal{S}^g(\mathbf{X}, \mathbf{z})$.
3. *For any* $X \in \mathbf{X}$ *and* $k \in \{1, \ldots, |\mathbf{D}^X|\}$, *let* $\mathbf{X}^D = \mathbf{sc}(\mathsf{D}^{X,k})$ *and* $\mathbf{X}^R = \mathbf{X} \setminus \mathbf{X}^D$. *It holds that*

$$\mathcal{S}^g(\mathbf{X}, Z_X = k, \mathbf{Z} \setminus \{Z_X\}) = \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathcal{S}^g(\mathbf{X}^R, Z_X = k, \mathbf{Z} \setminus \{Z_X\}), \tag{4.66}$$

i.e.

$$\mathbf{X}^D \perp\!\!\!\perp \mathbf{X}^R \cup \mathbf{Z} \setminus \{Z_X\} \mid Z_X \tag{4.67}$$

In particular

$$\mathcal{S}^g(\mathbf{X}, Z_X = k) = \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathcal{S}^g(\mathbf{X}^R, Z_X = k), \tag{4.68}$$

For a fixed X, we split the sets $\mathbf{Y}$ and $\mathbf{Y}'$ from (4.56) according to the distribution $\mathsf{D}^{X,k}$:

$$\begin{aligned} \mathbf{Y}_k &= \mathbf{Y} \cap \mathbf{sc}(\mathsf{D}^{X,k}), \quad & \mathbf{Y}'_k &= \mathbf{Y}' \cap \mathbf{sc}(\mathsf{D}^{X,k}) \\ \bar{\mathbf{Y}}_k &= \mathbf{Y} \setminus \mathbf{sc}(\mathsf{D}^{X,k}), \quad & \bar{\mathbf{Y}}'_k &= \mathbf{Y}' \setminus \mathbf{sc}(\mathsf{D}^{X,k}) \end{aligned} \tag{4.69}$$

[5] For representing distribution nodes, we use symbol ⓐ.

Then, using Proposition 4.5, we get

$$\int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y} = \sum_{k=1}^{|\mathbf{D}^X|} \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}', Z_X = k) \, \mathrm{d}\mathbf{y} \tag{4.70}$$

$$= \sum_{k=1}^{|\mathbf{D}^X|} \left(\int_{\boldsymbol{\mathcal{Y}}[\mathbf{Y}_k]} \mathsf{D}^{X,k}(\mathbf{y}^k, \mathbf{y}'[\mathbf{Y}'_k]) \, \mathrm{d}\mathbf{y}^k \right) \underbrace{\left(\int_{\boldsymbol{\mathcal{Y}}[\bar{\mathbf{Y}}_k]} \mathcal{S}^g(\bar{\mathbf{y}}^k, \mathbf{y}'[\bar{\mathbf{Y}}'_k], Z_X = k) \, \mathrm{d}\bar{\mathbf{y}}^k \right)}_{g_{X,k}} \tag{4.71}$$

To find the factors $g_{X,k}$, note that the k^{th} term in (4.70) can also be computed by the differential approach, setting all $\lambda_{Z_X=k} = 1$ and computing

$$\int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}', Z_X = k) \, \mathrm{d}\mathbf{y} = \frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \lambda_{Z_X=k}}. \tag{4.72}$$

This derivative is given according to (4.42) and (4.46) as

$$\frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \lambda_{Z_X=k}} = \frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \mathsf{P}^{X,k}} \int_{\boldsymbol{\mathcal{Y}}[\mathbf{Y}_k]} \mathsf{D}^{X,k}(\mathbf{y}^k, \mathbf{y}'[\mathbf{Y}'_k]) \, \mathrm{d}\mathbf{y}^k \tag{4.73}$$

Comparing (4.71) and (4.73), we find that

$$g_{X,k} = \frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}^g(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \mathsf{P}^{X,k}} = \frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \mathsf{D}^{X,k}} \tag{4.74}$$

The right hand side in (4.74) holds since all $\lambda_{Z_X=k} = 1$ and the structure above all $\mathsf{D}^{X,k}$ in $\mathcal{S}$ and above all $\mathsf{P}^{X,k}$ in $\mathcal{S}^g$ is identical. Since for all k, X is contained either in $\mathbf{Y}_k$ or in $\bar{\mathbf{Y}}_k$, we find the distribution over X with modified evidence by

$$\int_{\boldsymbol{\mathcal{Y}}[\mathbf{Y}\setminus\{X\}]} \mathcal{S}(X, \mathbf{y}, \mathbf{y}'[\mathbf{Y}' \setminus \{X\}]) \, \mathrm{d}\mathbf{y} = \tag{4.75}$$

$$\sum_{k=1}^{|\mathbf{D}^X|} \int_{\boldsymbol{\mathcal{Y}}[\mathbf{Y}_k\setminus\{X\}]} \mathsf{D}^{X,k}(X, \mathbf{y}^k, \mathbf{y}'[\mathbf{Y}'_k \setminus \{X\}]) \, \mathrm{d}\mathbf{y}^k \; \frac{\partial \int_{\boldsymbol{\mathcal{Y}}} \mathcal{S}(\mathbf{y}, \mathbf{y}') \, \mathrm{d}\mathbf{y}}{\partial \mathsf{D}^{X,k}}, \tag{4.76}$$

i.e. we input evidence as in inference scenario (2.), and find the derivatives $\frac{\partial \mathcal{S}}{\partial \mathsf{D}^{X,k}}$ at each input distribution using backpropagation. These derivatives are used as weights of a linear combination of distributions $\mathbf{D}^X$, evaluated anew for evidence in $\mathbf{sc}(\mathsf{D}^{X,k}) \setminus \{X\}$. Similar as inference scenario (2.), inference scenario (3.) reduces to the *corresponding inference scenario at the input distributions*. Note, that although we used gated SPNs and extended network polynomials for deriving this result, all required terms can be computed in the *original SPN* $\mathcal{S}$. However, introducing the DS, which are artificially latent RVs, helped us in our approach. In the next chapter, we go one step further and introduce a latent RV for each sum node in the SPN.

5 Sum-Product Networks as Latent Variable Models

In [79] a latent RV interpretation for SPNs was introduced, associating a latent RV with each sum node, where each child of the sum corresponds to a state of the RV. In that way, the sum operation can be interpreted as marginalization of this latent RV. This technique is often applied to *mixture distributions*, as illustrated in the following example.

Example 5.1. *Consider a mixture of Gaussians over RV X, whose PDF p is defined as:*

$$p(X) = \sum_{k=1}^{K} w_k \, \mathcal{N}(X \mid \mu_k, \sigma_k), \tag{5.1}$$

where K is the number of components, μ_k and σ_k are the mean and the standard deviation of the k^{th} component, and w_k are the mixture weights, satisfying

$$w_k \geq 0, \ \sum_k w_k = 1. \tag{5.2}$$

Eq. (5.1) *has two interpretations:*

1. *It is a convex combination of K PDFs $\mathcal{N}(\cdot \mid \mu_k, \sigma_k)$ and thus a PDF.*
2. *It is the marginal distribution of a joint distribution over X, Z, defined as*

$$p(X, Z) = p(Z) \, p(X \mid Z), \tag{5.3}$$

where $|\mathbf{val}(Z)| = K$, $p(Z = z_k) = w_k$ and $p(X \mid Z = z_k) = \mathcal{N}(X \mid \mu_k, \sigma_k)$.

The second interpretation in this example has several advantages, e.g. that the EM algorithm can be applied for ML learning. In this chapter, we investigate in detail the latent RV interpretation of sum nodes in SPNs.

5.1 Augmentation of SPNs

In [79], it was proposed to make the latent RVs explicit as depicted in Figure 5.1. For each sum

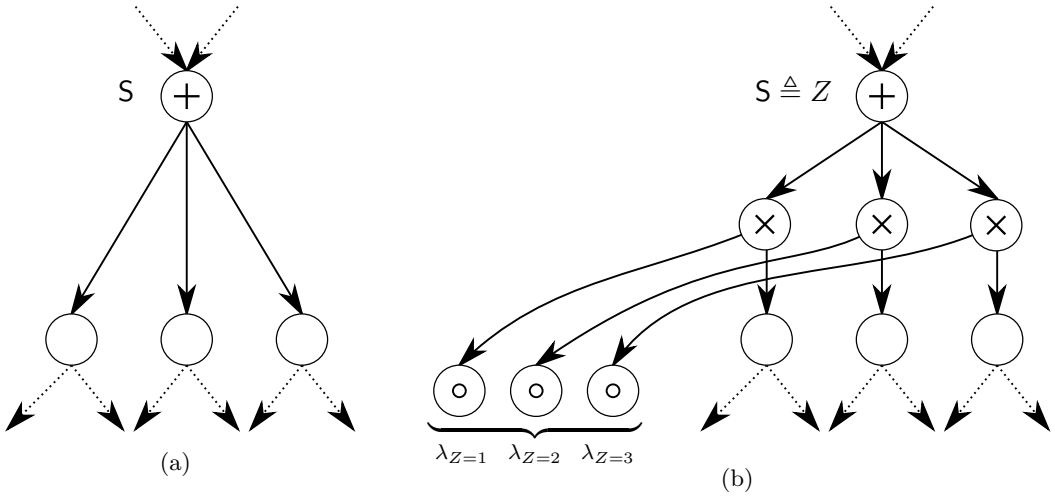


Figure 5.1: Augmenting SPN structure to explicitly introduce IVs representing states of a latent RV associated with sum node S*. (a): Part of an SPN containing a sum node* S *with three children. (b): Augmented structure explicitly containing IVs for each sum child.*

node S, one introduces a latent RV Z whose states correspond with the children of S. The SPN structure is augmented by introducing one product and one IV for each child, such that the sub-SPN rooted at this child is turned on/off by setting/unsetting its corresponding IV, as shown in Figure 5.1 (b). When all IVs are set to 1 the sum node S in Figure 5.1 (b) computes the same value as in Figure 5.1 (a). Since setting all IVs of Z to 1 corresponds to marginalizing Z, each sum node can be interpreted as a latent, marginalized RV. Furthermore, the weights associated to S should be interpretable as CPT of Z, conditioned on a certain context represented by the ancestor sum nodes of S.

However, taking a closer look, we recognize that augmenting the SPN in this way is too simplistic. Consider the augmentation of the SPN structure from Figure 5.1(a) within a larger structural context, shown in Figure 5.2(a). Explicitly introducing the IVs corresponding to Z in Figure 5.2 (b) renders node S' *incomplete*, since S is no descendant of N and therefore Z is not in the scope of N. Since we require completeness of the SPN in order that setting all IVs of Z corresponds the marginalization, this approach for augmenting the SPN is flawed.

Furthermore, note that S' also corresponds to a latent RV Z'. Although we know the probability distribution of Z if Z' is in the state corresponding to P, namely the weights associated to S, we do not know this distribution when Z' is in the state corresponding to N. Intuitively, we recognize that the state of Z is "irrelevant" when Z' is in the state corresponding to N, since it does not influence the resulting distribution over the model RVs $\mathbf{X}$. Nevertheless, the probabilistic model is *not completely specified*, which is unsatisfying.

A remedy for these problems is shown in Figure 5.2 (c), by augmenting the SPN structure further. We introduce the *twin* sum node $\bar{\mathsf{S}}$ of S whose children are the IVs corresponding to Z. The twin $\bar{\mathsf{S}}$ is connected as child of an additional product node which is interconnected between S' and N. Since this new product node has scope $\mathbf{sc}(\mathsf{N}) \cup \{Z\}$, S' is rendered complete now. If Z' takes the state corresponding to N (or actually the state corresponding to the new product node), we now have a specified conditional distribution of Z, namely the weights of the twin sum node $\bar{\mathsf{S}}$. Furthermore, given that all IVs of Z are set to 1, the network depicted in Figure 5.2 (c) still computes the same function as the network in Figure 5.2 (a), since $\bar{\mathsf{S}}$ constantly outputs 1.

The question raises, which weights should be used for the twin sum node $\bar{\mathsf{S}}$. Basically, we can assume arbitrary normalized weights, which will cause $\bar{\mathsf{S}}$ to constantly output 1. Following an esthetic and maximum entropy argument, however, we would assume uniform weights for $\bar{\mathsf{S}}$. While the choice of weights is not crucial for *evaluating evidence* in the SPN, it will however

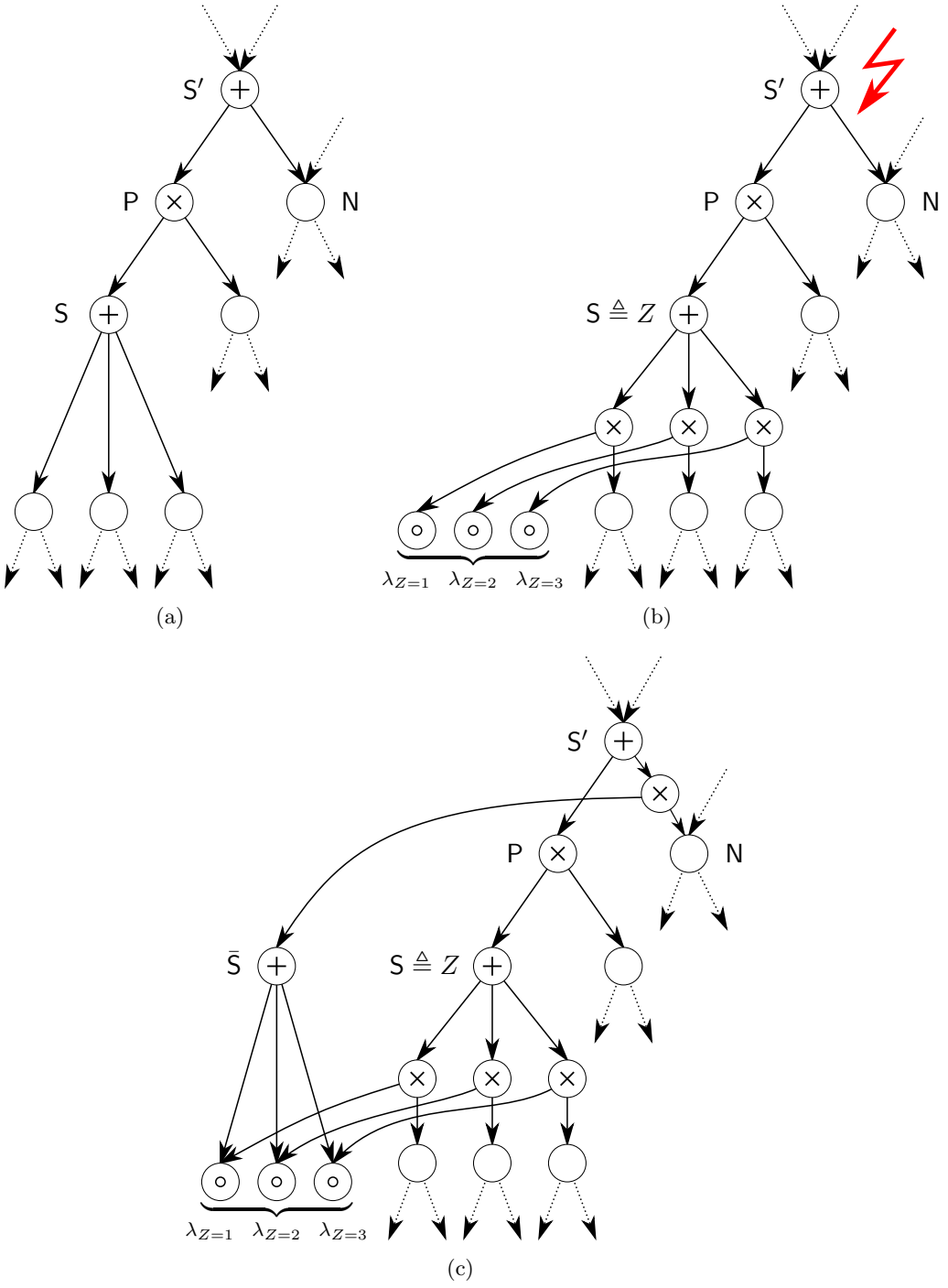


Figure 5.2: Problems occurring when IVs of latent variables are introduced. (a): Larger part of the SPN depicted in Figure 5.1 (a). In this example we assume that $\mathsf{S} \notin \mathbf{desc}(\mathsf{N})$. *(b): Introducing IVs for* Z *renders* S' *incomplete. (c): Remedy by extending SPN structure further, introducing twin sum node* $\bar{\mathsf{S}}$.

play a role when we discuss *MPE inference* in Section 5.3. For now, let us formally introduce the latent RV interpretation of sum nodes.

Definition 5.1 (Associated Latent RVs)**.** *Let* $\mathcal{S}$ *be an SPN over* $\mathbf{X}$. *For each* $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ *we assume an arbitrary but fixed ordering of its children* $\mathbf{ch}(\mathsf{S}) = \{\mathsf{C}_\mathsf{S}^1, \ldots, \mathsf{C}_\mathsf{S}^{K_\mathsf{S}}\}$, *where* $K_\mathsf{S} = |\mathbf{ch}(\mathsf{S})|$. *Let* Z_S *be an RV on the same probability space as* $\mathbf{X}$, *with* $\mathbf{val}(Z_\mathsf{S}) = \{1, \ldots, K_\mathsf{S}\}$,[6]

[6] For ease of discussion, we use the first K_S integers as states of Z_S.

where state k corresponds to child C_S^k. We call Z_S the latent RV associated *with S. For sets of sum nodes $\mathbf{S}$ we define $\mathbf{Z}_\mathbf{S} = \{Z_\mathsf{S} \mid \mathsf{S} \in \mathbf{S}\}$.*

To distinguish $\mathbf{X}$ from the latent RVs we will refer to the former as *model RVs*. Definition 5.1 postulates latent RVs associated with sum nodes, defined on the same probability space as $\mathbf{X}$, and where each state corresponds to a sum child. We will complete this definition by specifying the *augmented* SPN, defining the joint distribution of $\mathbf{X}$ and the associated RVs. First we need some more definitions.

Definition 5.2 (Sum Ancestors/Descendants)**.** *Let $\mathcal{S}$ be an SPN and N a node in $\mathcal{S}$. We define sum ancestors/descendants as*

$$\mathbf{anc}_\mathbf{S}(\mathsf{N}) := \mathbf{anc}(\mathsf{N}) \cap \mathbf{S}(\mathcal{S}) \tag{5.4}$$

$$\mathbf{desc}_\mathbf{S}(\mathsf{N}) := \mathbf{desc}(\mathsf{N}) \cap \mathbf{S}(\mathcal{S}) \tag{5.5}$$

Definition 5.3 (Conditioning sums)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$. For each sum node S we define the* conditioning sums *of S as*

$$\mathbf{S}^c(\mathsf{S}) := \{\mathsf{S}^c \in \mathbf{anc}_\mathbf{S}(\mathsf{S}) \setminus \{\mathsf{S}\} \mid \exists \mathsf{C} \in \mathbf{ch}(\mathsf{S}^c) \colon \mathsf{S} \notin \mathbf{desc}(\mathsf{C})\}. \tag{5.6}$$

Definition 5.4 (Twin weights)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$. Let $\bar{\boldsymbol{w}}$ be a set of nonnegative weights containing a weight $\bar{w}_{\mathsf{S},\mathsf{C}}$ for each $w_{\mathsf{S},\mathsf{C}}$, where $\bar{w}_{\mathsf{S},\mathsf{C}} \geq 0$, $\sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} \bar{w}_{\mathsf{S},\mathsf{C}} = 1$. We call $\bar{\boldsymbol{w}}$ a* set of twin weights *and $\bar{w}_{\mathsf{S},\mathsf{C}}$ is the* twin weight *of $w_{\mathsf{S},\mathsf{C}}$.*

We now define *augmented* SPNs, explicitly introducing the associated RVs in the SPN and specifying the completed probabilistic model.

Definition 5.5 (Augmentation of SPN)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, $\bar{\boldsymbol{w}}$ be a set of twin weights and $\mathcal{S}'$ be the result of Algorithm 3. $\mathcal{S}'$ is called the* augmented *SPN, denoted as $\mathcal{S}' =: \mathbf{aug}(\mathcal{S})$. Within the context of $\mathcal{S}'$, is C_S^k is called the k^{th}* former child *of S. The introduced product node P_S^k is called* link *of S, C_S^k and $\lambda_{Z_\mathsf{S}=k}$, respectively. The sum node $\bar{\mathsf{S}}$, if introduced, is called the* twin *sum node of S.*

In the first part of Algorithm 3 (1–8) we introduce the links P_S^k which are interconnected between the sum node S and its k^{th} child. Each link P_S^k has a single parent, namely S, and simply copies the former child C_S^k, which might have more than one parent. Note that we already used links in Algorithm 2 for the proof for Theorem 4.3. In steps 10–12, we introduce IVs corresponding to the associated RV Z_S, which is the augmentation according to [79]. As illustrated in Figure 5.2, this can render other sum nodes incomplete. Thus, when necessary, we introduce a twin sum node in steps 14–20, to treat this problem. Note that we assume a full set of twin weights, even when for some sum nodes no twin node is introduced. This will ease discussion in Section 5.2. The following proposition states the soundness of augmentation.

Proposition 5.1. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$ and $\mathbf{Z} := \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$. Then $\mathcal{S}'$ is a complete and decomposable SPN over $\mathbf{X} \cup \mathbf{Z}$ with $\mathcal{S}'(\mathbf{X}) \equiv \mathcal{S}(\mathbf{X})$.*

Proposition 5.1 states that the augmented SPN represents the same distribution over $\mathbf{X}$ as the original SPN when the latent RVs $\mathbf{Z}$ are marginalized, while being a *completely specified probabilistic model* over $\mathbf{X}$ and $\mathbf{Z}$. Clearly, the augmented SPN will usually not be generated for practical purposes, but we will mainly consider it as a theoretical tool. The following example illustrates the process of augmentation.

Algorithm 3 Augment SPN $\mathcal{S}$

$\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}),\ \forall k \in \{1, \dots, K_{\mathsf{S}}\}$ let $w_{\mathsf{S},k} = w_{\mathsf{S},\mathsf{C}^k_{\mathsf{S}}}$, $\bar{w}_{\mathsf{S},k} = \bar{w}_{\mathsf{S},\mathsf{C}^k_{\mathsf{S}}}$

Introduce links:

for $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
  for $k = 1 \dots K_{\mathsf{S}}$ **do**
    Introduce a new product node $\mathsf{P}^k_{\mathsf{S}}$ in $\mathcal{S}$
    Disconnect $\mathsf{C}^k_{\mathsf{S}}$ from S
    Connect $\mathsf{C}^k_{\mathsf{S}}$ as child of $\mathsf{P}^k_{\mathsf{S}}$
    Connect $\mathsf{P}^k_{\mathsf{S}}$ as child of S with weight $w_{\mathsf{S},k}$
  end for
end for

Introduce IVs and twins:

for $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
  for $k \in \{1, \dots, K_{\mathsf{S}}\}$ **do**
    Connect new IV $\lambda_{Z_{\mathsf{S}}=k}$ as child of $\mathsf{P}^k_{\mathsf{S}}$
  end for
  if $\mathbf{S}^c(\mathsf{S}) \neq \emptyset$ **then**
    Introduce a twin sum node $\bar{\mathsf{S}}$ in $\mathcal{S}$
    $\forall k \in \{1, \dots, K_{\mathsf{S}}\}$: connect $\lambda_{Z_{\mathsf{S}}=k}$ as child of $\bar{\mathsf{S}}$, and let $w_{\bar{\mathsf{S}},\lambda_{Z_{\mathsf{S}}=k}} = \bar{w}_{\mathsf{S},k}$
    for $\mathsf{S}^c \in \mathbf{S}^c(\mathsf{S})$ **do**
      for $k \in \{k \mid \mathsf{S} \notin \mathbf{desc}(\mathsf{P}^k_{\mathsf{S}^c})\}$ **do**
        Connect $\bar{\mathsf{S}}$ as child of $\mathsf{P}^k_{\mathsf{S}^c}$
      end for
    end for
  end if
end for

Example 5.2 (Augmentation of SPN)**.** *Figure 5.3(a) shows an example SPN over three RVs X_1, X_2, X_3. Figure 5.3(b) shows the corresponding augmented SPN.*

It should be noted that augmentation presented here is not necessarily the only way to introduce latent RVs, and, that sum nodes have sometimes a more natural interpretation. For some examples see Section 6.1, where sum nodes, or more precisely sum-weights, have a natural interpretation of CPTs of model RVs. However, due to Proposition 5.1 we have nevertheless always the option to perform augmentation and interpret sum nodes as latent RVs. We could also construct augmented SPNs of augmented SPNs, i.e. consider $\mathbf{aug}(\mathbf{aug}(\mathcal{S}))$, since $\mathbf{aug}(\mathcal{S})$ is simply a complete and decomposable SPN over $\mathbf{X}$, $\mathbf{Z}$. This would blow up the RV set further and introduce redundant copies of the latent RVs introduced in the first augmentation.

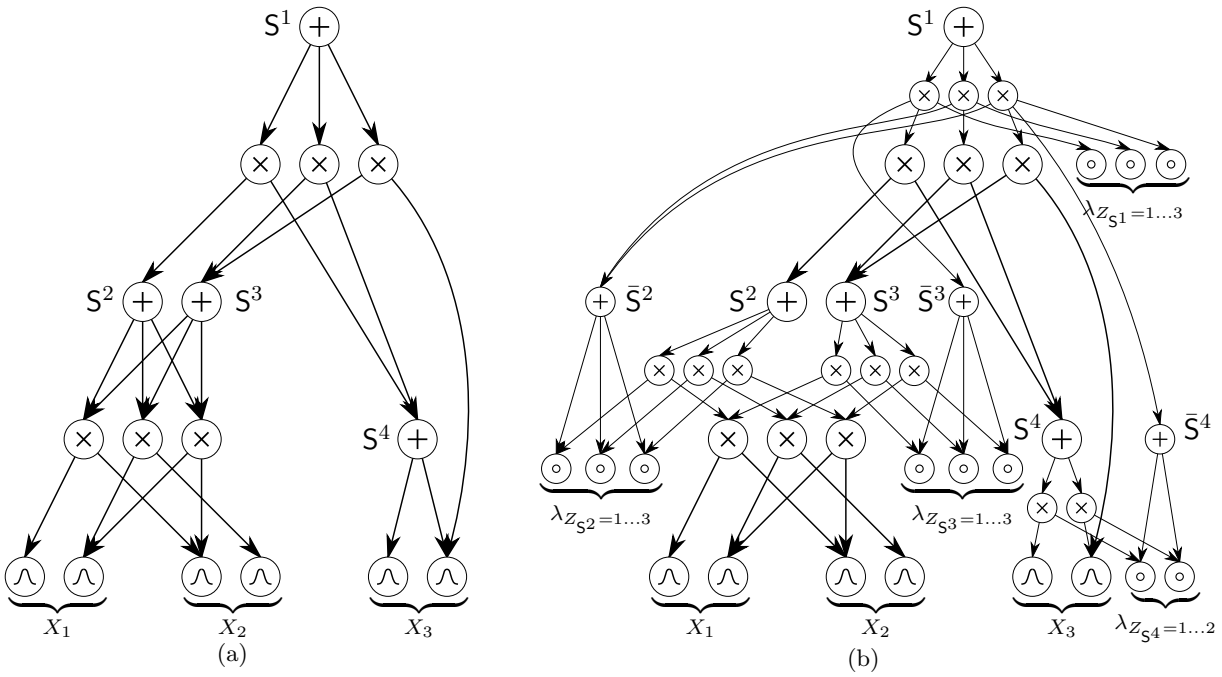


Figure 5.3: Augmentation of an SPN. (a): Example SPN over $\mathbf{X} = \{X_1, X_2, X_3\}$, containing sum nodes S^1, S^2, S^3 and S^4. (b): Augmented SPN, containing IVs corresponding to Z_{S^1}, Z_{S^2}, Z_{S^3}, Z_{S^4}, links and twin sum nodes $\bar{\mathsf{S}}^2$, $\bar{\mathsf{S}}^3$, $\bar{\mathsf{S}}^4$. For nodes introduced by augmentation smaller circles are used.

5.2 Independencies and Interpretation of Sum Weights

We now investigate certain conditional independencies which hold in the augmented SPN, allowing us to interpret the augmented SPN as BN. Furthermore, we also formalize the interpretation of sum weights as CPTs in the corresponding BN. To this end, we introduce the notion of *configured SPNs*.

Definition 5.6 (Configured SPN)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, and let $\mathbf{Y} \subseteq \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$ and $\mathbf{y} \in \mathbf{val}(\mathbf{Y})$. The* configured SPN *$\mathcal{S}^{\mathbf{y}}$ is obtained by Algorithm 4.*

Algorithm 4 Configure SPN

```
1: S^y ← aug(S)
2: for Y ∈ Y do
3:     for y ∈ val(Y), y ≠ y[Y] do
4:         Delete λ_{Y=y} and its corresponding link
5:     end for
6: end for
7: Delete all nodes in S^y which are not reachable from the root
```

Example 5.3 (Configured SPN)**.** *Two examples of configured SPNs of the SPN in Example 5.2, using two different $\mathbf{y}$, are shown in Figure 5.4.*

Intuitively, the configured SPN isolates the computational structure selected by $\mathbf{y}$. All sum edges which "survive" Algorithm 4 are equipped with the same weights as in the augmented SPN. Therefore, a configured SPN is in general *not* locally normalized. We note the following properties of configured SPNs.

Proposition 5.2. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, $\mathbf{Y} \subseteq \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$ and $\mathbf{Z} = \mathbf{Z}_{\mathbf{S}(\mathcal{S})} \setminus \mathbf{Y}$. Let $\mathbf{y} \in \mathbf{val}(\mathbf{Y})$ and let $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$. It holds that*

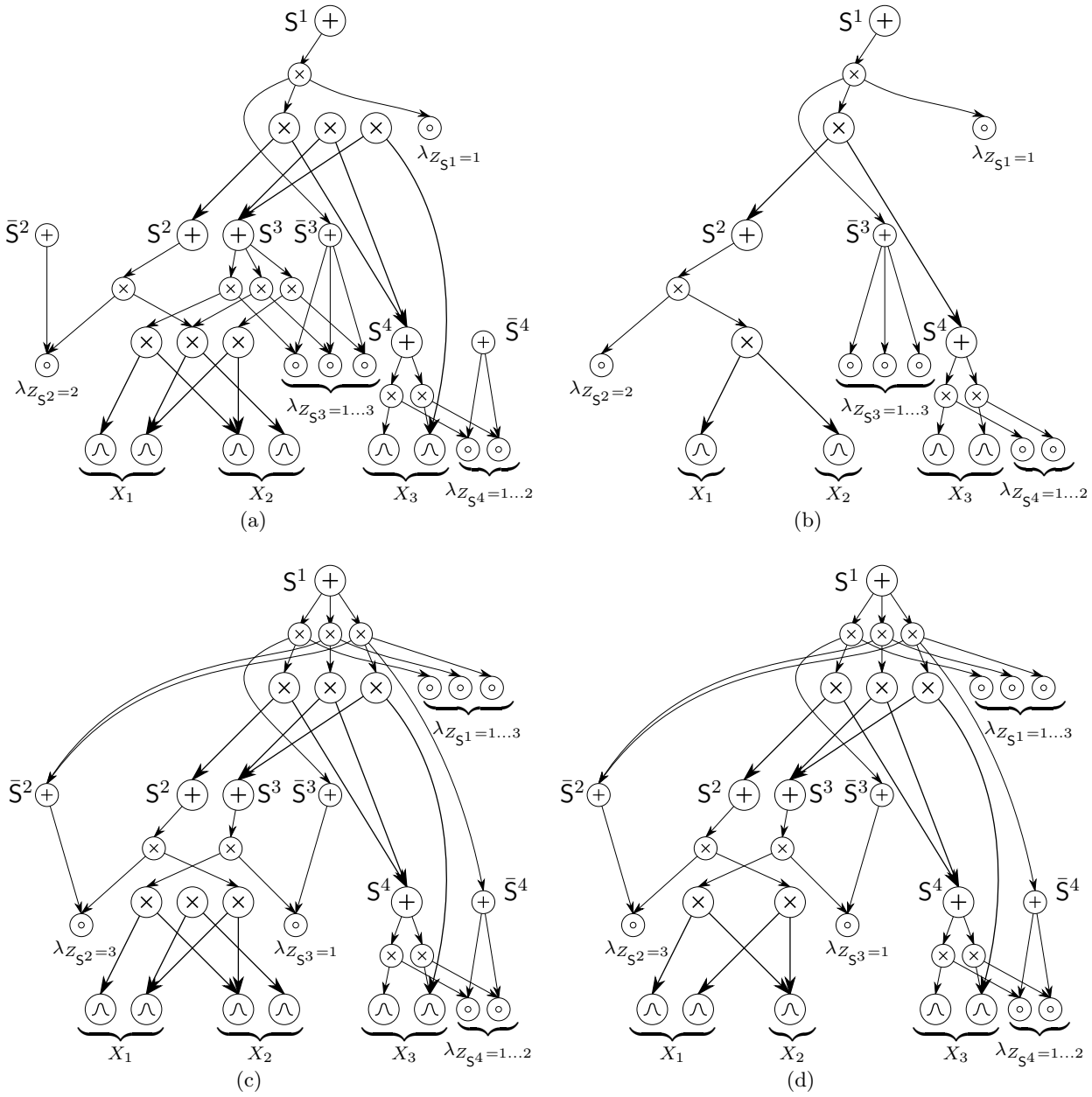


Figure 5.4: Configured SPNs of SPN shown in Figure 5.3. (a): Augmented SPN before performing step 7 of Algorithm 4 for $\mathbf{Y} = \{Z_{\mathsf{S}^1}, Z_{\mathsf{S}^2}\}$ *and configuration* $\mathbf{y} = (1, 2)$*. (b): Configured SPN resulting from removing unreachable nodes from (a). (c), (d): Similar to (a), (b), for for* $\mathbf{Y} = \{Z_{\mathsf{S}^2}, Z_{\mathsf{S}^3}\}$ *and configuration* $\mathbf{y} = (3, 1)$*.*

1. *Each node in* $\mathcal{S}^{\mathbf{y}}$ *has the same scope as its corresponding node in* $\mathcal{S}'$*.*
2. $\mathcal{S}^{\mathbf{y}}$ *is a complete and decomposable SPN over* $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$*.*
3. *For any node* N *in* $\mathcal{S}^{\mathbf{y}}$ *with* $\mathbf{sc}(\mathsf{N}) \cap \mathbf{Y} = \emptyset$*, we have that* $\mathcal{S}^{\mathbf{y}}_{\mathsf{N}} = \mathcal{S}'_{\mathsf{N}}$*.*
4. *For* $\mathbf{y}' \in \mathbf{val}(\mathbf{Y})$ *it holds that*

$$\mathcal{S}^{\mathbf{y}}(\mathbf{X}, \mathbf{Z}, \mathbf{y}') = \begin{cases} \mathcal{S}'(\mathbf{X}, \mathbf{Z}, \mathbf{y}') & \text{if } \mathbf{y}' = \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The following theorem shows certain conditional independencies which hold in the augmented

SPN. For the proof, we need the following lemma.

Lemma 5.1. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, let S be a sum node in $\mathcal{S}$ and $\mathbf{Z} = \mathbf{Z}_{\mathbf{ancs}(\mathsf{S})\setminus\{\mathsf{S}\}}$. For any $\mathbf{z} \in \mathbf{val}(\mathbf{Z})$, the configured SPN $\mathcal{S}^{\mathbf{z}}$ contains either S or its twin $\bar{\mathsf{S}}$, but not both.*

Theorem 5.1. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, S be an arbitrary sum in $\mathcal{S}$ and $w_k = w_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$, $\bar{w}_k = \bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$, $k = 1, \ldots, |\mathbf{ch}(\mathsf{S})|$. Let*

$$Z := Z_{\mathsf{S}}, \tag{5.8}$$
$$\mathbf{Z} := \mathbf{Z}_{\mathbf{ancs}(\mathsf{S})} \setminus \{Z\}, \tag{5.9}$$
$$\mathbf{Y}^d := \mathbf{sc}(\mathsf{S}) \cup \mathbf{Z}_{\mathbf{descs}(\mathsf{S})} \setminus \{Z\}, \tag{5.10}$$
$$\mathbf{Y}^n := (\mathbf{X} \cup \mathbf{Z}_{\mathbf{S}(\mathcal{S})}) \setminus (\mathbf{Y}^d \cup \mathbf{Z} \cup \{Z\}). \tag{5.11}$$

Furthermore, let $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$. There exists a two-partition of $\mathbf{val}(\mathbf{Z})$, i.e. $\boldsymbol{\mathcal{Z}}$, $\bar{\boldsymbol{\mathcal{Z}}}$: $\boldsymbol{\mathcal{Z}} \cup \bar{\boldsymbol{\mathcal{Z}}} = \mathbf{val}(\mathbf{Z})$, $\boldsymbol{\mathcal{Z}} \cap \bar{\boldsymbol{\mathcal{Z}}} = \emptyset$,[7] such that

$$\forall \mathbf{z} \in \boldsymbol{\mathcal{Z}} : \ \mathcal{S}'(Z = k, \mathbf{Y}^n, \mathbf{z}) = w_k \, \mathcal{S}'(\mathbf{Y}^n, \mathbf{z}), \ \textit{and} \tag{5.12}$$
$$\forall \mathbf{z} \in \bar{\boldsymbol{\mathcal{Z}}} : \ \mathcal{S}'(Z = k, \mathbf{Y}^n, \mathbf{z}) = \bar{w}_k \, \mathcal{S}'(\mathbf{Y}^n, \mathbf{z}). \tag{5.13}$$

Proof.

By Lemma 5.1, for each $\mathbf{z} \in \mathbf{val}(\mathbf{Z})$ the configured SPN $\mathcal{S}^{\mathbf{z}}$ contains either S or $\bar{\mathsf{S}}$, but not both. Let $\boldsymbol{\mathcal{Z}}$ be the subset of $\mathbf{val}(\mathbf{Z})$ such that S is in $\mathcal{S}^{\mathbf{z}}$ and $\bar{\boldsymbol{\mathcal{Z}}}$ be the subset of $\mathbf{val}(\mathbf{Z})$ such that $\bar{\mathsf{S}}$ is in $\mathcal{S}^{\mathbf{z}}$.

Fix $Z = k$ and $\mathbf{z} \in \boldsymbol{\mathcal{Z}}$. We want to compute $\mathcal{S}'(Z = k, \mathbf{Y}^n, \mathbf{z})$, i.e. we marginalize $\mathbf{Y}^d$. According to Proposition 5.2 (*4.*), this equals $\mathcal{S}^{\mathbf{z}}(Z = k, \mathbf{Y}^n, \mathbf{z})$. According to Proposition 5.2 (*3.*), the sub-SPN rooted at former child C_S^k is the same in $\mathcal{S}'$ and $\mathcal{S}^{\mathbf{z}}$. Since $\mathcal{S}'$ is locally normalized, this sub-SPNs is also locally normalized in $\mathcal{S}^{\mathbf{z}}$. Since the scope of the former child C_S^k is a sub-set of $\mathbf{Y}^d$ and $\lambda_{Z=k} = 1$, the link P_S^k outputs 1. Since $\lambda_{Z=k'} = 0$ for $k' \neq k$, the sum S outputs w_k.

Now consider the set of nodes in $\mathcal{S}^{\mathbf{z}}$ which have Z in their scope, not including $\lambda_{Z=k}$ and P_S^k. Clearly, since $\bar{\mathsf{S}}$ is not in $\mathcal{S}^{\mathbf{z}}$, this set must be $\mathbf{anc}(\mathsf{S})$. Let $\mathsf{N}_1, \ldots, \mathsf{N}_L$ be a topologically ordered list of $\mathbf{anc}(\mathsf{S})$, where S is N_1 and N_L is the root. Let $\mathbf{Y}_l^n := \mathbf{sc}(\mathsf{N}_l) \cap \mathbf{Y}^n$ and $\mathbf{Z}_l := \mathbf{sc}(\mathsf{N}_l) \cap \mathbf{Z}$. We show by induction that for $l = 1, \ldots, L$, we have

$$\mathsf{N}_l(Z = k, \mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]) = w_k \, \mathsf{N}_l(\mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]). \tag{5.14}$$

Since $\mathbf{Y}_1^n = \emptyset$ and $\mathbf{Z}_1 = \emptyset$, and $\mathsf{N}_1(Z = k) = w_k$, the induction basis holds. Assume that (5.14) holds for all $\mathsf{N}_1, \ldots, \mathsf{N}_{l-1}$. If N_l is a sum, we have due to completeness

$$\begin{aligned}
\mathsf{N}_l(Z = k, \mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]) &= \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{N}_l)} w_{\mathsf{N}_l,\mathsf{C}} \, \mathsf{C}(Z = k, \mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]) && (5.15)\\
&= \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{N}_l)} w_{\mathsf{N}_l,\mathsf{C}} \, w_k \, \mathsf{C}(\mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]) && (5.16)\\
&= w_k \, \mathsf{N}_l(\mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]), && (5.17)
\end{aligned}$$

i.e. the induction step holds for sums. When N_l is a product, due to decomposability, it must

[7] Here, $\boldsymbol{\mathcal{Z}}$ and $\bar{\boldsymbol{\mathcal{Z}}}$ are not necessarily elements from $\mathcal{H}_{\mathbf{Z}}$.

have a single child C' with $Z \in \mathbf{sc}(\mathsf{C}')$. We have

$$\mathsf{N}_l(Z=k, \mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]) = \mathsf{C}'(Z=k, \mathbf{sc}(\mathsf{C}') \cap \mathbf{Y}_l^n, \mathbf{z}[\mathbf{sc}(\mathsf{C}') \cap \mathbf{Z}]) \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{N}_l) \setminus \{\mathsf{C}'\}} \mathsf{C}(\cdot) \tag{5.18}$$

$$= w_k \, \mathsf{C}'(\mathbf{sc}(\mathsf{C}') \cap \mathbf{Y}^n, \mathbf{z}[\mathbf{sc}(\mathsf{C}') \cap \mathbf{Z}]) \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{N}_l) \setminus \{\mathsf{C}'\}} \mathsf{C}(\cdot) \tag{5.19}$$

$$= w_k \, \mathsf{N}_l(\mathbf{Y}_l^n, \mathbf{z}[\mathbf{Z}_l]), \tag{5.20}$$

i.e. the induction step holds for products. Therefore, by induction, (5.14) also holds for the root, and (5.12) follows.

Now fix the input to arbitrary $Z = k$ and $\mathbf{z} \in \bar{\boldsymbol{\mathcal{Z}}}$. Clearly, $\bar{\mathsf{S}}$ outputs $\bar{w}_k$ and (5.13) can be shown in similar way as (5.12). □

Corollary 5.1. *Make the same assumptions as in Theorem 5.1. In $\mathcal{S}'$, we have*

$$Z \perp\!\!\!\perp \mathbf{Y}^n \,|\, \mathbf{Z}. \tag{5.21}$$

Furthermore we have

$$\mathcal{S}'(Z=k \,|\, \mathbf{y}^n, \mathbf{z}) = \mathcal{S}'(Z=k \,|\, \mathbf{z}) = \begin{cases} w_k & \text{if } \mathbf{z} \in \boldsymbol{\mathcal{Z}} \\ \bar{w}_k & \text{if } \mathbf{z} \in \bar{\boldsymbol{\mathcal{Z}}} \end{cases} \tag{5.22}$$

Theorem 5.1 and Corollary 5.1 show that the weights and twin weights of a sum node S can be interpreted as a conditional probability table (CPT) of the associated RV Z_S. More precisely, given that the sum node ancestors "select a path" to S, i.e. $\mathbf{z} \in \boldsymbol{\mathcal{Z}}$, then $w_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$ is the probability that Z_S is in its k^{th} state. Given the complementary event $\mathbf{z} \in \bar{\boldsymbol{\mathcal{Z}}}$, the corresponding twin weight $\bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$ is the corresponding conditional probability.

We now also see an independency structure in augmented SPNs: a latent RV corresponding to S is conditional independent of all model RVs which are not in the scope of S, when the RVs corresponding to its sum ancestors are observed. It is also independent from all other latent RVs corresponding to sum nodes which are neither ancestors nor descendants of S. This is the correspondent to the *local conditional independence* statements BNs, cf. Theorem 3.1:

> An RV is independent from its non-descendants, given its parents.

Given an SPN $\mathcal{S}$, we can now construct an IMAP of $\mathbf{aug}(\mathcal{S})$ expressed via a BN. In the IMAP, we connect Z_S as child of all $\mathbf{Z}_{\mathbf{anc_S}(\mathsf{S})} \setminus \{Z_\mathsf{S}\}$ and connect all model RVs which are in the scope of S as child of Z_S. As an example, Figure 5.5 (a) shows the augmented SPN from Figure 5.3 and Figure 5.5 (b) its corresponding IMAP.

Furthermore, when all input distributions of the SPN are over single RVs, we can conclude that all model RVs are independent given all the latent RVs. In this case, we can interpret the augmented SPN as a naive Bayes mixture model with a structured latent RV $\mathbf{Z}_{\mathbf{S}(\mathcal{S})}$, whose state-space grows exponentially in the number of sum nodes in the SPN. This is illustrated in Figure 5.5 (c). In the naive Bayes interpretation, the input distributions take the role of conditional distributions over the model RVs, which are *shared* among the exponentially many latent states of $\mathbf{Z}_{\mathbf{S}(\mathcal{S})}$.

The IMAP representation shows that an SPN can be interpreted as *hierarchical mixture model*. However, the representation of an SPN as BN obscures the highly constrained structure of the CPTs of the augmented SPN. Consider an arbitrary sum node S and let $\mathbf{Z}_a = \mathbf{Z}_{\mathbf{anc_S}(\mathsf{S}) \setminus \{\mathsf{S}\}}$ in some SPN. In general, the associated RV Z_S can depend on the state of all RVs $\mathbf{Z}_a$. Thus, in a general BN the CPT for Z_S would require $|\mathbf{val}(\mathbf{Z}_a)| \, (|\mathbf{val}(Z_\mathsf{S})| - 1)$ parameters. This parameterization grows exponentially in $|\mathbf{Z}_a|$.

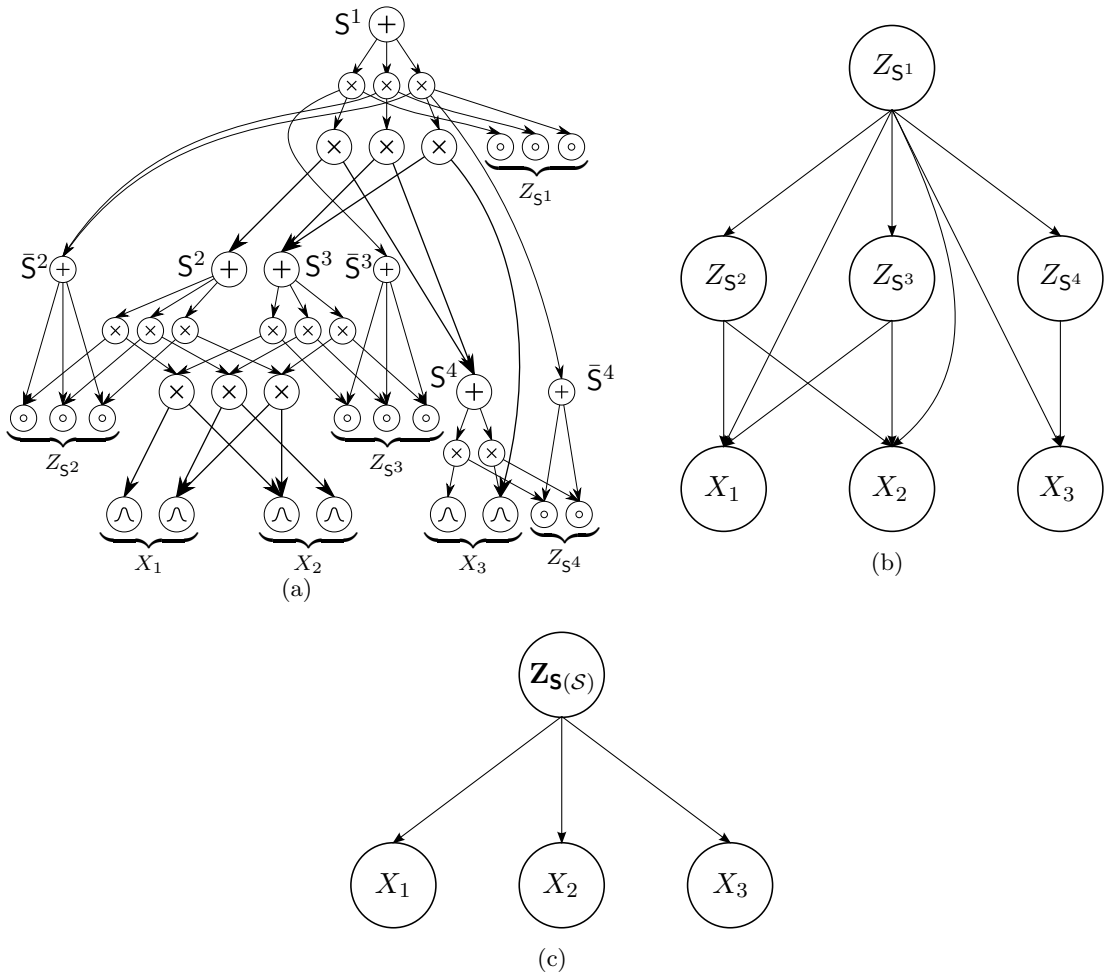


Figure 5.5: Dependency structure of an augmented SPN. (a): Augmented SPN from Figure 5.3. (b): Corresponding IMAP represented as BN. (c): Naive Bayes interpretation of augmented SPN.

However, the state space of $\mathbf{Z}_a$ is two-partitioned into $\boldsymbol{\mathcal{Z}}$ and $\bar{\boldsymbol{\mathcal{Z}}}$. For all states in $\boldsymbol{\mathcal{Z}}$, the conditional distribution of Z_S is constant and given by the weights of S, and for $\bar{\boldsymbol{\mathcal{Z}}}$ the conditional distribution is given by the twin weights, which are the same as the weights of the twin sum node $\bar{\mathsf{S}}$, if it exists. Thus, the CPT of Z_S is highly constrained and can be represented by $2\,(|\mathbf{val}(Z_\mathsf{S})|-1)$ parameters in the augmented SPN. Furthermore, the twin weights will typically be set to the uniform distribution, so that we actually have $|\mathbf{val}(Z_\mathsf{S})|-1$ free parameters for Z_S, i.e. the original SPN parameters.

The two-partition of $\mathbf{val}(\mathbf{Z}_a)$ can be made explicit by further introducing a binary RV Y_S with $\mathbf{val}(Y_\mathsf{S}) = \{y_\mathsf{S}, y_{\bar{\mathsf{S}}}\}$ where $Y_\mathsf{S} = y_\mathsf{S} \Leftrightarrow \mathbf{Z}_a \in \boldsymbol{\mathcal{Z}}$ and $Y_\mathsf{S} = y_{\bar{\mathsf{S}}} \Leftrightarrow \mathbf{Z}_a \in \bar{\boldsymbol{\mathcal{Z}}}$. Y_S can be seen as *switching parent* of Z_S which, when observed, renders Z_S *independent* from $\mathbf{Z}_a$. This switching parent can also be explicitly introduced in the augmented SPN as depicted in Figure 5.6. Here we simply introduce two new IVs $\lambda_{Y_\mathsf{S}=y_\mathsf{S}}$ and $\lambda_{Y_\mathsf{S}=y_{\bar{\mathsf{S}}}}$, which switch on/off the output of S and $\bar{\mathsf{S}}$, respectively. It is easy to see that when these IV are constantly set to 1, i.e. when Y_S is marginalized, the augmented SPN performs exactly the same computations as before. It is furthermore easy to show that completeness and decomposability of the augmented SPN are maintained when the switching parent is introduced. Introducing the switching parent allows to derive the EM algorithm for SPN-weights, cf. Section 6.3.

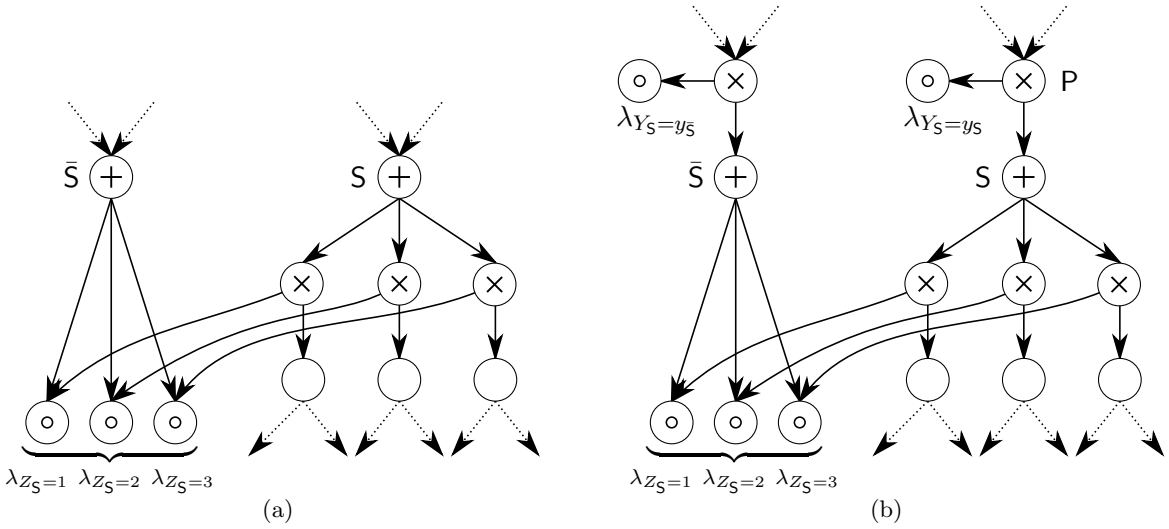


Figure 5.6: Explicitly introducing a switching parent Y_S in an augmented SPN. (a): Part of an augmented SPN containing a sum node with three children and its twin. (b): Explicitly introduced switching parent Y_S using IVs $\lambda_{Y_S=y_S}$ and $\lambda_{Y_S=y_{\bar{S}}}$.

5.3 Most Probable Explanation and Maximum A-posterior Hypothesis

As discussed in Section 4.4, SPNs allow to perform several inference scenarios efficiently, like evaluating the SPN distribution for complete evidence, partial marginalization and computing the marginal posteriors given some evidence. In [79], SPNs were applied for reconstructing data using MPE inference. Given some distribution p over $\mathbf{X}$ and partial evidence $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$, $\boldsymbol{\mathcal{X}} \neq \emptyset$, MPE can be formalized as finding

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} p(\mathbf{x}). \tag{5.23}$$

We assume here that p has a maximum in $\boldsymbol{\mathcal{X}}$, which, unless $\boldsymbol{\mathcal{X}}$ is closed, does not necessarily exist in the case of continuous RVs. MPE inference according to (5.23) generalizes MPE inference as defined in Section 3.4, i.e. finding

$$\arg\max_{\mathbf{x}^q \in \mathbf{val}(\mathbf{X}^q)} p(\mathbf{x}^q, \mathbf{x}^o). \tag{5.24}$$

This scenario is a special case of (5.23), where $\forall X \in \mathbf{X}^q \colon \boldsymbol{\mathcal{X}}[X] = \mathbf{val}(X)$ and $\forall X \in \mathbf{X}^o \colon \boldsymbol{\mathcal{X}}[X] = \{\mathbf{x}^o[X]\}$. MPE defined as in (5.23) allows more generally to incorporate constraints on the domain of the RVs, as illustrated in the following example.

Example 5.4. *Let p be a distribution over X_1, X_2, X_3 and we have evidence that $X_1 = x_1$ and that $X_2 \leq 42$. Using $\boldsymbol{\mathcal{X}} = \{x_1\} \times [-\infty, 42] \times \mathbb{R}$ and solving (5.23), we find the most probable solution $\mathbf{x}^*$ consistent with our evidence.*

MPE is a special case of MAP, defined as finding

$$\arg\max_{\mathbf{y} \in \boldsymbol{\mathcal{Y}}} \int_{\boldsymbol{\mathcal{Z}}} p(\mathbf{y}, \mathbf{z}) \, \mathrm{d}\mathbf{z}, \tag{5.25}$$

for some two-partition of $\mathbf{X}$, i.e. $\mathbf{X} = \mathbf{Y} \cup \mathbf{Z}$. Similar as for MPE, this generalizes MAP inference as defined in Section 3.4. MAP reduces to MPE when $\mathbf{Y} = \mathbf{X}$. Both MPE and MAP are NP-hard in general BNs [8, 54, 68, 69], and MAP is inherently harder than MPE [54, 69]. Theoretically,

this is reflected that the decision versions of MPE is NP-complete, while the decision version of MAP is $\mathrm{NP}^{\mathrm{PP}}$-complete [54, 69]. In practice this means that in model classes which allow to solve MPE efficiently, the MAP problem typically remains hard to solve.

In [79] it is stated that the MPE solution can be found efficiently in SPNs, where maximization is over the model RVs $\mathbf{X}$ and the latent RVs $\mathbf{Z}$ represented by the sum nodes. For this purpose, a Viterbi-style algorithm is used: each sum node is replaced by a *max* node, i.e. a node which outputs the maximum of its child nodes. To find the MPE solution, back-tracking from the root to the leave nodes is performed, where each sum node selects a maximizing child and each product nodes selects all of its children. However, as discussed at the beginning of this chapter, the latent RV interpretation in [79] is incomplete, since the latent RV model in [79] does not specify the distribution of "irrelevant" latent RVs. We proposed the augmented SPN as a remedy for this issue. In this section we show that the algorithm proposed in [79] indeed finds an MPE solution when *applied to the augmented SPN*. However, applying this algorithm to the non-augmented SPN corresponds to an unfortunate implicit parameter selection for the twin sum nodes and leads to a phenomenon we call *low-depth bias*.

5.3.1 MPE Inference in Augmented SPNs

We formally investigate MPE using *max-product networks*.

Definition 5.7 (Max-Product Network)**.** *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$. The* max-product network *(MPN) $\hat{\mathcal{S}}$ of $\mathcal{S}$ is a network obtained as follows:*

- *Replace each distribution node D by a* maximizing *distribution node $\hat{\mathsf{D}}\colon \mathcal{H}_{\mathbf{sc}(\mathsf{D})} \mapsto [0, \infty]$, defined as*

$$\hat{\mathsf{D}}(\boldsymbol{\mathcal{Y}}) := \max_{\mathbf{y} \in \boldsymbol{\mathcal{Y}}} \mathsf{D}(\mathbf{y}) \tag{5.26}$$

- *Replace each sum node S in $\mathcal{S}$ by a* max *node $\hat{\mathsf{S}}$. The outgoing edges of a max node are equipped with the same weights as for its corresponding sum node. A max node is defined as*

$$\hat{\mathsf{S}} := \max_{\mathsf{C} \in \mathbf{ch}(\hat{\mathsf{S}})} w_{\hat{\mathsf{S}},\mathsf{C}}\, \mathsf{C}. \tag{5.27}$$

- *Each product P in $\mathcal{S}$ corresponds to a product $\hat{\mathsf{P}}$ in $\hat{\mathcal{S}}$.*

The scope in MPNs is defined as in SPNs.

The following theorem shows that for *augmented* SPNs the MPN computes the MPE probability.

Theorem 5.2. *Let $\mathcal{S}$ be an* augmented *SPN over $\mathbf{X}$, where $\mathbf{X}$ already comprises the model RVs of the original SPN and the latent RVs introduced by augmentation, and let $\hat{\mathcal{S}}$ the corresponding MPN. Let N be some node in $\mathcal{S}$ and $\hat{\mathsf{N}}$ its corresponding node in $\hat{\mathcal{S}}$. For every $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{sc}(\mathsf{N})}$ we have*

$$\hat{\mathsf{N}}(\boldsymbol{\mathcal{X}}) = \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathsf{N}(\mathbf{x}). \tag{5.28}$$

Proof. We proof the theorem using an inductive argument. The theorem clearly holds for any $\hat{\mathsf{D}}$ by definition. Consider a product $\hat{\mathsf{P}}$ and assume the theorem holds for all $\mathbf{ch}(\hat{\mathsf{P}})$. Then the

theorem also holds for $\hat{\mathsf{P}}$, since

$$\hat{\mathsf{P}}(\boldsymbol{\mathcal{X}}) = \prod_{\hat{\mathsf{C}} \in \mathbf{ch}(\hat{\mathsf{P}})} \hat{\mathsf{C}}(\boldsymbol{\mathcal{X}}) \tag{5.29}$$

$$= \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathsf{C}(\mathbf{x}) \tag{5.30}$$

$$= \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathsf{C}(\mathbf{x}) \tag{5.31}$$

$$= \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathsf{P}(\mathbf{x}), \tag{5.32}$$

where (5.31) holds due to decomposability. Now consider a max node $\hat{\mathsf{S}}$. Since $\mathcal{S}$ is an augmented SPN, $\hat{\mathsf{S}}$ corresponds to either a sum node S or to a twin sum node $\bar{\mathsf{S}}$. In either case we have an associated RV Z_{S} with IVs $\lambda_{Z_{\mathsf{S}}=1}, \ldots, \lambda_{Z_{\mathsf{S}}=K}$, which are connected directly or via the links to the sum node. In the MPN, the IVs are replaced by maximizing distributions $\hat{\lambda}_{Z_{\mathsf{S}}=1}, \ldots, \hat{\lambda}_{Z_{\mathsf{S}}=K}$, where

$$\hat{\lambda}_{Z_{\mathsf{S}}=k}(\mathcal{Z}) = \max_{z_{\mathsf{S}} \in \mathcal{Z}} \hat{\lambda}_{Z_{\mathsf{S}}=k}(z_{\mathsf{S}}) = \begin{cases} 1 & \text{if } k \in \mathcal{Z} \\ 0 & \text{otherwise.} \end{cases} \tag{5.33}$$

Assume first that $\hat{\mathsf{S}}$ corresponds to a twin $\bar{\mathsf{S}}$, having $\lambda_{Z_{\mathsf{S}}=k}$ as children. Let $\bar{w}_k = w_{\bar{\mathsf{S}},\lambda_{Z_{\mathsf{S}}=k}}$. We have

$$\hat{\mathsf{S}}(\mathcal{Z}) = \max_{k \in \mathbf{val}(Z_{\mathsf{S}})} \bar{w}_k \, \hat{\lambda}_{Z_{\mathsf{S}}=k}(\mathcal{Z}) \tag{5.34}$$

$$= \max_{k \in \mathbf{val}(Z_{\mathsf{S}})} \bar{w}_k \underbrace{\max_{z_{\mathsf{S}} \in \mathcal{Z}} \lambda_{Z_{\mathsf{S}}=k}(z_{\mathsf{S}})}_{=0, \text{ for } k \notin \mathcal{Z}, \ (5.33)} \tag{5.35}$$

$$= \max_{z_{\mathsf{S}} \in \mathcal{Z}} \bar{\mathsf{S}}(z_{\mathsf{S}}) \tag{5.36}$$

i.e. the theorem holds for twin sum nodes. Now assume that $\hat{\mathsf{S}}$ corresponds to a (non-twin) sum node S. The children of S are the links $\mathsf{P}^k_{\mathsf{S}} = \lambda_{Z_{\mathsf{S}}=k} \times \mathsf{C}^k_{\mathsf{S}}$, $k \in \mathbf{val}(Z_{\mathsf{S}})$. $\mathsf{P}^k_{\mathsf{S}}$ and $\mathsf{C}^k_{\mathsf{S}}$ correspond to $\hat{\mathsf{P}}^k_{\hat{\mathsf{S}}}$ and $\hat{\mathsf{C}}^k_{\hat{\mathsf{S}}}$ in $\hat{\mathcal{S}}$, respectively. Let $w_k := w_{\mathsf{S},\mathsf{P}^k_{\mathsf{S}}}$. When we assume that the theorem holds for all $\hat{\mathsf{C}}^k_{\hat{\mathsf{S}}}$, then it follows that

$$\hat{\mathsf{S}}(\boldsymbol{\mathcal{X}}) = \max_{k \in \mathbf{val}(Z_{\mathsf{S}})} w_k \, \hat{\mathsf{P}}^k_{\hat{\mathsf{S}}}(\boldsymbol{\mathcal{X}}) \tag{5.37}$$

$$= \max_{k \in \mathbf{val}(Z_{\mathsf{S}})} w_k \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathsf{P}^k_{\mathsf{S}}(\mathbf{x}) \tag{5.38}$$

$$= \max_{k \in \mathbf{val}(Z_{\mathsf{S}})} w_k \underbrace{\left(\max_{z_{\mathsf{S}} \in \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}]} \lambda_{Z_{\mathsf{S}}=k}(z_{\mathsf{S}}) \right)}_{=0, \text{ for } k \notin \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}], \ (5.33)} \left(\max_{\mathbf{y} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{C}^k_{\mathsf{S}})]} \mathsf{C}^k_{\mathsf{S}}(\mathbf{y}) \right) \tag{5.39}$$

$$= \max_{k \in \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}]} \max_{\mathbf{y} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{C}^k_{\mathsf{S}})]} w_k \, \mathsf{C}^k_{\mathsf{S}}(\mathbf{y}) \tag{5.40}$$

$$= \max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathsf{S}(\mathbf{x}), \tag{5.41}$$

i.e. the theorem also holds for S. By induction, the theorem holds for all nodes. □

Theorem 5.2 shows that the MPN maximizes the probability of the augmented SPN over $\boldsymbol{\mathcal{X}}$. The proof also shows how to actually *find* a maximizing assignment in an augmented SPN,

Algorithm 5 MPE inference in augmented SPNs

1: For all sums S, let $\mathbf{ch}'(\mathsf{S}) := \{\mathsf{C}_{\mathsf{S}}^k \mid k \in \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}]\}$
2: Evaluate $\boldsymbol{\mathcal{X}}$ in corresponding MPN $\hat{\mathcal{S}}$ (upwards pass)
3: Initialize Q as empty queue
4: $Q \twoheadleftarrow$ root node
5: **while** Q not empty **do**
6: $\quad \mathsf{N} \twoheadleftarrow Q$
7: $\quad$ **if** N is a sum node **then**
8: $\quad\quad Q \twoheadleftarrow \arg\max_{\hat{\mathsf{C}} \in \mathbf{ch}'(\mathsf{N})} \left\{ w_{\hat{\mathsf{N}},\hat{\mathsf{C}}}\, \hat{\mathsf{C}}(\boldsymbol{\mathcal{X}}[\mathbf{sc}(\hat{\mathsf{C}})]) \right\}$
9: $\quad$ **else if** N is a product node **then**
10: $\quad\quad \forall \mathsf{C} \in \mathbf{ch}(\mathsf{N}) : Q \twoheadleftarrow \mathsf{C}$
11: $\quad$ **else if** N is a distribution node **then**
12: $\quad\quad \mathbf{x}^*[\mathbf{sc}(\mathsf{N})] = \arg\max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{N})]} \mathsf{N}(\mathbf{x})$
13: $\quad$ **end if**
14: **end while**
15: **return** $\mathbf{x}^*$

i.e. how to solve the MPE problem: In (5.30) we see that a product node is maximized by independently maximizing each of its children. Therefore, finding a maximizing assignment for the product is reduced to independently finding maximizing assignments for its children. In (5.35) and (5.38), we see that a (twin) sum node is maximized by maximizing the weighted maxima of its children. Therefore, finding a maximizing assignment for a (twin) sum node is reduced to find a child with maximal weighted output and find a maximizing assignment for this child. Algorithm 5 finds a maximizing state of the augmented SPN using back-tracking. In this implementation we use a queue Q, where $Q \twoheadleftarrow \mathsf{N}$ and $\mathsf{N} \twoheadleftarrow Q$ denote the *enqueue* and *dequeue* operations, respectively. This is the same algorithm as proposed in [79], which was, however, applied to general SPNs, i.e. SPNs which are not *augmented* SPNs. Does this still correspond to an MPE solution? The answer is yes, but we implicitly use *deterministic* (zero-one) weights for the twin sum nodes.

To show this, let us modify Algorithm 5, such that it can be applied to an arbitrary (not augmented) SPN, but returning the MPE solution of the corresponding *augmented* SPN. Such an algorithm is also required for practical applications, since we usually do not want to explicitly generate the augmented SPN. Algorithm 6 is such a modification of Algorithm 5, simulating the additional structure introduced by augmentation. In steps 2–9 we construct the weights $\tilde{w}_{\mathsf{S},\mathsf{C}}$ which are used to compensate the twin max nodes. For a particular S and C, $\tilde{w}_{\mathsf{S},\mathsf{C}}$ is the product over the maxima of all twin max nodes which would be connected to the link of C in the augmented SPN. Thus, $\tilde{w}_{\mathsf{S},\mathsf{C}}$ is the result of "simulating" the SPN structure corresponding to the twin sums. By equipping the corresponding MPN with weights $w_{\hat{\mathsf{S}},\hat{\mathsf{C}}} \leftarrow \tilde{w}_{\mathsf{S},\mathsf{C}} \times w_{\mathsf{S},\mathsf{C}}$, every max node in the MPN gets the same inputs as in the augmented version.

Since the latent RVs corresponding to the sum nodes are not explicitly introduced, we need to assign their MPE state "by hand". In step 20, the maximizing state of the latent RV Z_{S} is assigned. In steps 28–30, the maximizing states of those latent RVs are assigned, which were not visited during back-tracking.

Algorithm 6 is essentially equivalent to the back-tracking algorithm in [79] when

1. $\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}) : \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}] = \mathbf{val}(Z_{\mathsf{S}})$, i.e. Z_{S} is maximized over all its possible states.
2. The states of the RV corresponding to not visited sum nodes are not assigned, i.e. steps 28–30 are not performed.
3. All $\tilde{w}_{\mathsf{S},\mathsf{C}} \equiv 1$, which is the case when all twin weights are deterministic, i.e. for each sum

Algorithm 6 MPE inference

For all sums S, let $\mathbf{ch}'(\mathsf{S}) := \{\mathsf{C}_\mathsf{S}^k \,|\, k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]\}$

$\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}) : \forall \mathsf{C} \in \mathbf{ch}(\mathsf{S}) : \ \tilde{w}_{\mathsf{S},\mathsf{C}} \leftarrow 1$
for all sum nodes $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
    for $\mathsf{S}^c \in \mathbf{S}^c(\mathsf{S})$ **do**
        for $\mathsf{C} \in \{\mathsf{C} \in \mathbf{ch}(\mathsf{S}^c) \,|\, \mathsf{S} \notin \mathbf{desc}(\mathsf{C})\}$ **do**
            $\tilde{w}_{\mathsf{S}^c,\mathsf{C}} \leftarrow \tilde{w}_{\mathsf{S}^c,\mathsf{C}} \times \max\limits_{k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]} \bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$
        end for
    end for
end for
Equip corresponding MPN $\hat{\mathcal{S}}$ with weights $w_{\hat{\mathsf{S}},\hat{\mathsf{C}}} \leftarrow \tilde{w}_{\mathsf{S},\mathsf{C}} \times w_{\mathsf{S},\mathsf{C}}$
Evaluate $\boldsymbol{\mathcal{X}}$ in MPN $\hat{\mathcal{S}}$ (upwards pass)

$\mathbf{S} \leftarrow \mathbf{S}(\mathcal{S})$
Initialize Q as empty queue
$Q \twoheadleftarrow$ root node
while Q not empty **do**
    $\mathsf{N} \twoheadleftarrow Q$
    if N is a sum node **then**
        $k^* \leftarrow \operatorname*{arg\,max}\limits_{k:\, \mathsf{C}_\mathsf{N}^k \in \mathbf{ch}'(\mathsf{N})} \left\{ w_{\hat{\mathsf{N}},\hat{\mathsf{C}}_\mathsf{S}^k} \hat{\mathsf{C}}_\mathsf{S}^k(\boldsymbol{\mathcal{X}}[\mathbf{sc}(\hat{\mathsf{C}}_\mathsf{S}^k)]) \right\}$
        $Q \leftarrow \mathsf{C}_\mathsf{N}^{k^*}$
        $z_\mathsf{S}^* = k^*$
        $\mathbf{S} \leftarrow \mathbf{S} \setminus \{\mathsf{S}\}$
    else if N is a product node **then**
        $\forall \mathsf{C} \in \mathbf{ch}(\mathsf{N}) : Q \twoheadleftarrow \mathsf{C}$
    else if N is a distribution node **then**
        $\mathbf{x}^*[\mathbf{sc}(\mathsf{N})] = \operatorname*{arg\,max}\limits_{\mathbf{x} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{N})]} \mathsf{N}(\mathbf{x})$
    end if
end while

for $\mathsf{S} \in \mathbf{S}$ **do**
    $z_\mathsf{S}^* = \operatorname*{arg\,max}\limits_{k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]} \bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$
end for

return $\mathbf{x}^*, \mathbf{z}^*$

node, one twin weight is 1 and all others are 0.

In practice we will rarely care about point 1., since maximizing over all possible states of the latent RVs is the typical scenario. Also point 2. can typically be neglected, since we are usually not interested in these "irrelevant" latent RVs. However, point 3. deserves more attention: First, it is an unnatural choice to use 0-1 weights, i.e. to prefer one state of a latent RV over the others, even when this latent RV is rendered irrelevant. The maximum entropy principle would dictate to use uniform parameters for the twin nodes. Second, using 0-1 weights introduces a bias in MPE inference towards less structured models, which we call the *low-depth bias*. This is illustrated in Figure 5.7, which shows an SPN over three RVs X_1, X_2, X_3. The augmented SPN has two twin sum nodes, corresponding to S^2 and S^3. When their twin weights are 0-1 weights,

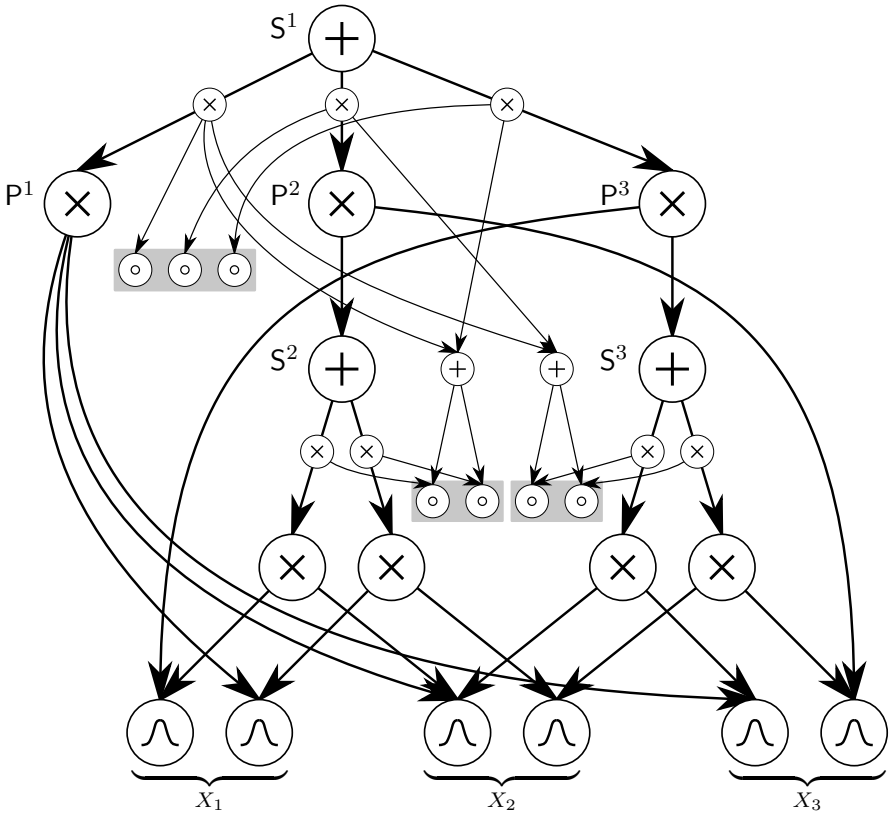


Figure 5.7: Low-depth bias using a simple SPN over $\{X_1, X_2, X_3\}$. The structure introduced by augmentation is depicted by small nodes and edges. When 0-1 weights are used as twin weights, the state of Z_{S^1} corresponding to P^1 is preferred over P^2 and P^3, since their probabilities are "dampened" by the weights of S^2 and S^3.

the selection of the state of Z_{S^1} is *biased* towards the state corresponding to P^1, which is a distribution assuming independence among X_1, X_2 and X_3. This comes from the fact, that the distributions P^2 and P^3 are dampened by the weights of S^2 and S^3, respectively. Therefore, when using 0-1 weights for twin sum nodes, we introduce a bias in MPE selection towards children whose sub-SPNs are less deep and less structured. The example in Figure 5.7 can be made arbitrarily extreme, using more than three RVs: when using 0-1 weights, MPE inference will prefer distributions assuming independence over finely structured SPNs. Clearly, this behavior is in general not desired. Using uniform weights for twin sum nodes is much "fairer", since in this case P^1 gets dampened by $\bar{\mathsf{S}}^2$ and $\bar{\mathsf{S}}^3$, P^2 by S^2 and $\bar{\mathsf{S}}^3$, and P^3 by $\bar{\mathsf{S}}^2$ and S^3. Uniform weights are actually the opposite choice to 0-1 weights: the former represent the strongest dampening via twin weights and therefore actually *penalize* less structured distributions. Note that regarding MPE inference, in [79], Section 5, it is noted that:

> The best results were obtained using sums on the upward pass and maxes on the downward pass (i.e., the MPE value of each hidden variable is computed conditioning on the MPE values of the hidden variables above it and summing out the ones below).

Presumably, the mentioned good results of this method stem from the fact that it does not suffer from the low-depth bias, since in this case the damping factor of each sum node is 1. However, this method does in general *not* deliver an MPE solution of the augmented SPN. We call this method the *sum approximation* of MPE. In Section 7.2, we test our results about MPE inference using toy examples and on the task of face image reconstruction [79].

5.3.2 MPE Inference in General SPNs

We see that MPE inference is accomplished efficiently in *augmented* SPNs, given that the maximization of the input distributions is efficient. The question rises if we can also efficiently perform MPE inference in general SPNs, i.e. SPNs which are not augmented SPNs. This scenario is desired in the task of *data completion*, where SPNs showed convincing results [31, 71, 73, 79]. Data completion is essentially the MPE inference scenario described in Section 3.4: We aim to predict the values of query RVs $\mathbf{X}^q$ given the values of observed RVs $\mathbf{X}^o$ using the joint distribution $\mathcal{S}(\mathbf{X}^o, \mathbf{X}^q)$.

However, in [31,71,73,79] (the sum approximation of) MPE inference in *augmented* SPNs was used, i.e.

$$\arg\max_{\mathbf{x}^q, \mathbf{z}} p_{\mathcal{S}'}(\mathbf{x}^o, \mathbf{x}^q, \mathbf{z}), \tag{5.42}$$

and the values of the latent RVs $\mathbf{Z}$ discarded. That is, MPE inference

$$\arg\max_{\mathbf{x}^q} p_{\mathcal{S}}(\mathbf{x}^o, \mathbf{x}^q), \tag{5.43}$$

or equivalently MAP inference

$$\arg\max_{\mathbf{x}^q} \sum_{\mathbf{z}} p_{\mathcal{S}'}(\mathbf{x}^o, \mathbf{x}^q, \mathbf{z}), \tag{5.44}$$

in the augmented SPN is approximated using MPE inference in the augmented SPN. However, MPE can be a poor approximation of MAP [68]. Thus we are interested in an MPE solution for general SPNs.

Unfortunately, this problem is generally NP-hard, even when restricted to binary RVs. This is shown in following theorem.

Theorem 5.3 (NP-completeness of MPE in SPNs)**.** *Consider SPNs over binary RVs* $\mathbf{X}$*, where* $\forall X \in \mathbf{X}\colon \mathbf{val}(X) = \{x, \bar{x}\}$*, using* $\lambda_{X=x}$*,* $\lambda_{X=\bar{x}}$ *as leaves and rational sum weights. The decision problem*

> ***SPN-MPE****: Given* $q \in \mathbb{Q}$*, is there an* $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ *such that* $\mathcal{S}(\mathbf{x}) \geq q$*?*

is NP-complete.

Proof. Membership in NP is immediate. Given an assignment $\mathbf{x} \in \mathbf{val}(\mathbf{X})$, we can decide $\mathcal{S}(\mathbf{x}) \geq q$ in polynomial time, assuming that the SPN is coded adequately, for instance as topologically sorted list of nodes.

To proof hardness, we reduce **MAX-SAT** to **SPN-MPE**. The NP-complete **MAX-SAT** problem is defined as [66]:

> **MAX-SAT**: Given a Boolean formula in conjunctive normal form (CNF), is there a truth assignment which make at least K clauses true?

For ease of discussion, we denote also the Boolean variables as $\mathbf{X} = \{X_1, \ldots, X_N\}$, with x and $\bar{x}$ denoting **true** and **false**, respectively. Assume M clauses and let L^m be the number of literals in the the m^{th} clause. We construct an SPN solving **MAX-SAT** as follows: For each X we introduce a sum node S^X with $\lambda_{X=x}$ and $\lambda_{X=\bar{x}}$ as children, and $w_{\mathsf{S}^X, \lambda_{X=x}} = w_{\mathsf{S}^X, \lambda_{X=\bar{x}}} = 0.5$, i.e. S^X represents the uniform distribution over X. For the m^{th} clause we introduce a product node P^m. For all X which have a positive literal in C^m, we connect $\lambda_{X=x}$, and for all X which have a negative literal in C^m, we connect $\lambda_{X=\bar{x}}$ as child of P^m. For all X which do not have a literal in C^m, we connect S^X as child of P^m. Therefore, $\mathsf{P}^m(\mathbf{x}) = 0.5^{(N-L^m)}$ for assignments $\mathbf{x}$ which make C^m **true**, and $\mathsf{P}^m(\mathbf{x}) = 0$, otherwise. Furthermore, we introduce a root sum node

S, with all P^m as children and $w_{\mathsf{S},\mathsf{P}^m} = \frac{2^{(N-L^m)}}{\mathscr{Z}}$, with $\mathscr{Z} = \sum_{m'=1}^{M} 2^{(N-L^{m'})}$. Therefore, for assignment $\mathbf{x}$, the m^{th} input to S is

$$w_{\mathsf{S},\mathsf{P}^m}\,\mathsf{P}^m(\mathbf{x}) = \begin{cases} \frac{1}{\mathscr{Z}} & \text{when } \mathbf{x} \text{ makes } C^m \textbf{ true} \\ 0 & \text{otherwise.} \end{cases} \tag{5.45}$$

Therefore, since S sums over (5.45), there exist an assignment $\mathbf{x}$ which make at least K clauses true, if and only if for such an assignment $\mathcal{S}(\mathbf{x}) \geq \frac{K}{\mathscr{Z}}$, i.e. our SPN solves the **MAX-SAT** problem.

Finally, we have to show that the reduction works in polynomial time. Without loss of generality, we assume that each variable appears in at least one clause. Therefore, generating $\lambda_{X=x}$, $\lambda_{X=\bar{x}}$ and S^X for all X works in linear time. For the m^{th} clause, we generate a P^m. Since each clause has to be mentioned in the input, this also works in linear time. S is generated in constant time. The weights $w_{\mathsf{S},\mathsf{P}^m} = \frac{2^{(N-L^m)}}{\mathscr{Z}}$, with $\mathscr{Z} = \sum_{m'=1}^{M} 2^{(N-L^{m'})}$ grow exponentially with N, i.e. their numeric representation grows linearly with N and can be easily generated using bit-shifts. Therefore, the reduction works in polynomial time. □

6 Learning Sum-Product Networks

As discussed in the last chapters, SPNs are a promising avenue for probabilistic modeling. Their main advantage is that many inference scenarios are stated as simple network passes. Under this perspective, SPNs readily represent *a probabilistic model and a corresponding inference machine of this model*. Using the latent RV interpretation of Chapter 5, we can always think of an SPN as a BN with a rich and potentially deep latent RV structure, which exhibits a high degree of structure in its CPDs. This structure can in principle capture complex dependencies among the model RVs.

The model power of SPNs and their conceptually simple and usually feasible inference mechanisms are the most appealing advantages of SPNs. Moreover, the fact that inference cost is directly related with the network size introduces *inference-aware* learning. On the one hand, it is easy to explicitly incorporate the inference cost during learning, as in [64, 72]. On the other hand, SPNs also naturally enforce inference-aware learning, by the mere fact that we have to hold some representation of SPNs in computer memory. Furthermore, a learning algorithm has to manipulate this representation typically many times, so that, informally stated, the "effort of learning" usually upper-bounds the "effort of inference" in SPNs. Thus, when the cost of learning an SPN is reasonable to us, we will probably also get a model with reasonable inference cost. On the other hand, one can easily design and represent a classical PGM where inference is intractable.

Their flexible definition leaves open how to *learn* SPNs. There have been proposed several general-purpose approaches to learn SPNs [31, 40, 71, 84], as well as domain-specific approaches [4, 40, 73, 79]. In this chapter, we collect and discuss some available approaches to learning. Similar as in classical PGMs it sometimes make sense to distinguish between structure and parameter learning. For instance, the algorithm used in [79] predefines a general structure based on locality in images; the final structure results from parameter learning with enforced sparseness. Furthermore, many classical PGMs can be readily represented as SPNs. We discuss some well-known examples in Section 6.1. Using these examples as building templates, one can sometimes design certain SPNs by hand, or in a semi-automatic manner.

For a given SPN structure, we need to learn the SPN parameters, i.e. the sum weights and possibly the parameters of the input distributions. As discussed in Section 3.3, the basic generative approach is to maximize the data likelihood. A basic approach using projected gradient ascend is discussed in Section 6.2. On the other hand, the latent RV interpretation of sum nodes opens the door for the EM algorithm. This was already considered in [79], where the provided EM update, however, is incorrect. A corrected EM-algorithm for SPNs is presented in Section 6.3. Both, gradient ascend and the EM-algorithm can be applied for maximizing likelihood. Since

SPNs can be seen as potentially deep hierarchical mixture models, they will generally yield local optima only. An interesting question is, if there is a sub-class of SPNs where ML parameter learning is easy. In Section 6.4 we discuss such a class, which we call *selective SPNs*. Section 6.5 shortly review some approaches to SPN structure learning.

6.1 Hall of Fame: Some Classical Probabilistic Models as SPNs

The simplest probabilistic model assumes independence among a set of RVs $\mathbf{X}$. An example of this distribution over three binary RVs X, Y, Z, together with the corresponding BN, is shown in Figure 6.1. Here the marginal distributions are computed by sums over the IVs of the respective RV. The marginals are further the children of a single product, yielding the product of marginals.

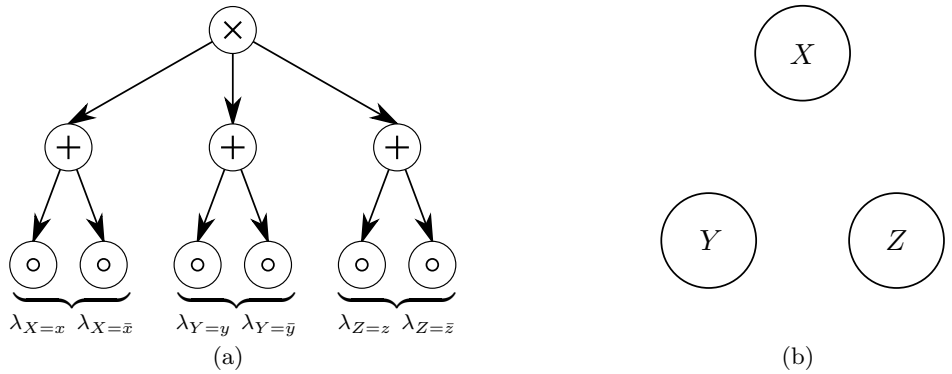
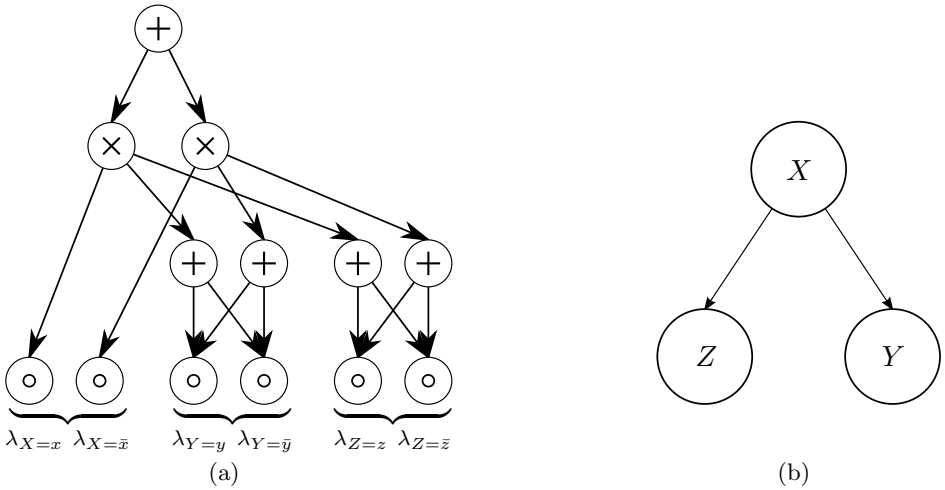


Figure 6.1: Independence among RVs. (a): SPN over three independent binary RVs X, Y, Z, where the joint distribution factorizes into product of marginals. (b): corresponding BN with no edges.

For finite-state RVs, it is also rather easy to derive SPNs for the three canonical examples of BNs discussed for d-separation, cf. Section 3.1.2. For the *diverging connection*, where two RV Y and Z have a common parent X, we use the insight that Y and Z are conditionally independent when X is set to a particular value. We can use a sum node whose weights represent the marginal distribution over X, having one child per state of X. For a particular state, we can re-use the pattern of Figure 6.1 designing two sub-SPNs assuming independence among Y and Z. These sub-SPNs are switched on/off by the corresponding IV of X. An example of an SPN representing a diverging connection is shown in Figure 6.2.

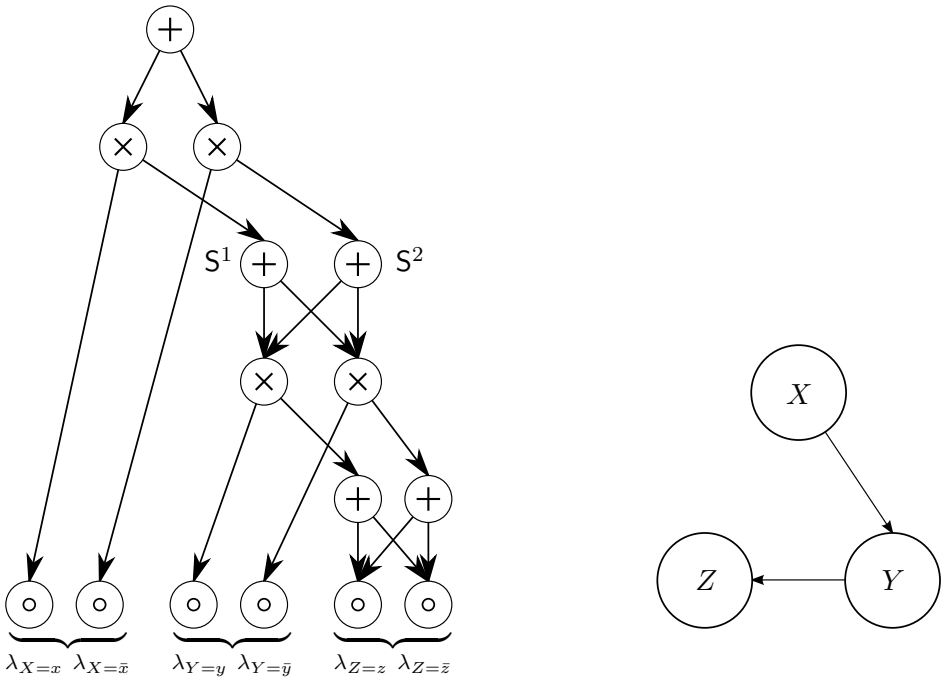
Similarly, we can derive an SPN representing a *serial connection*, as shown in Figure 6.3. As in Figure 6.2 we use a sum node to represent the marginal distribution over X and use the IVs corresponding to X to switch on/off sub-SPNs rooted at S^1 and S^2, representing conditional distributions over Y and Z, depending on X's states. Note that since S^1 and S^2 share their children, the distribution over Z depends merely on the state of Y, when set to a particular value, yielding the conditional independence assertion $X \perp\!\!\!\perp Z \,|\, Y$. By duplicating the shared sub-SPNs in Figure 6.3, we yield a SPN representing a *fully connected BN*, i.e. having no conditional independencies, in Figure 6.4.

However, it is not possible to directly reflect a *converging connection* in the *SPN structure*. It can be implemented in the SPN in Figure 6.4(a) by imposing equality constraints on the weights associated with edges which are connected by dashed lines: In this case the distribution of Y does not depend on the state of X. It is not surprising that a converging connection can not be reflected in the SPN structure, since an SPN is a general purpose inference machine, similar as the junction tree algorithm, cf. Section 3.4, which also moralizes converging connections in BNs as a first step. To exploit a-priori independence of X and Y, one would have to use *specialized* inference methods assuming that Z, or any of its descendant, is never observed.



(a) (b)

Figure 6.2: Diverging connection. (a): SPN over three binary RVs X, Y, Z, where Y and Z depend directly on X. (b): corresponding BN.



(a) (b)

Figure 6.3: Serial connection. (a): SPN over three binary RVs X, Y, Z, where Y depends directly on X and Z depends directly on Y. (b): corresponding BN.

Using these canonical BN structures as templates, it is easy to find SPNs for further classical models. By extending the serial connection in Figure 6.3 to a *Markov chain* over state RVs Y^t for time t, and attaching state-dependent SPNs over $\mathbf{X}^t$, we yield a *hidden Markov model* (HMM), shown in Figure 6.5.

Similarly, we can design a *Bayes classifier* whose class conditional models over *features* $\mathbf{X}$ are represented by SPNs, as shown in Figure 6.6. Naive Bayes, a special case of the Bayes classifier assuming independence among the features when the class is observed, is shown in Figure 6.7. Furthermore, as already mentioned, a single sum node represents a mixture distribution, shown in Figure 6.8(a). When the input distributions are Gaussian, and assuming independence for each component, Figure 6.8(b) shows a Gaussian mixture model over 4 RVs and having three components, using diagonal covariance matrices.

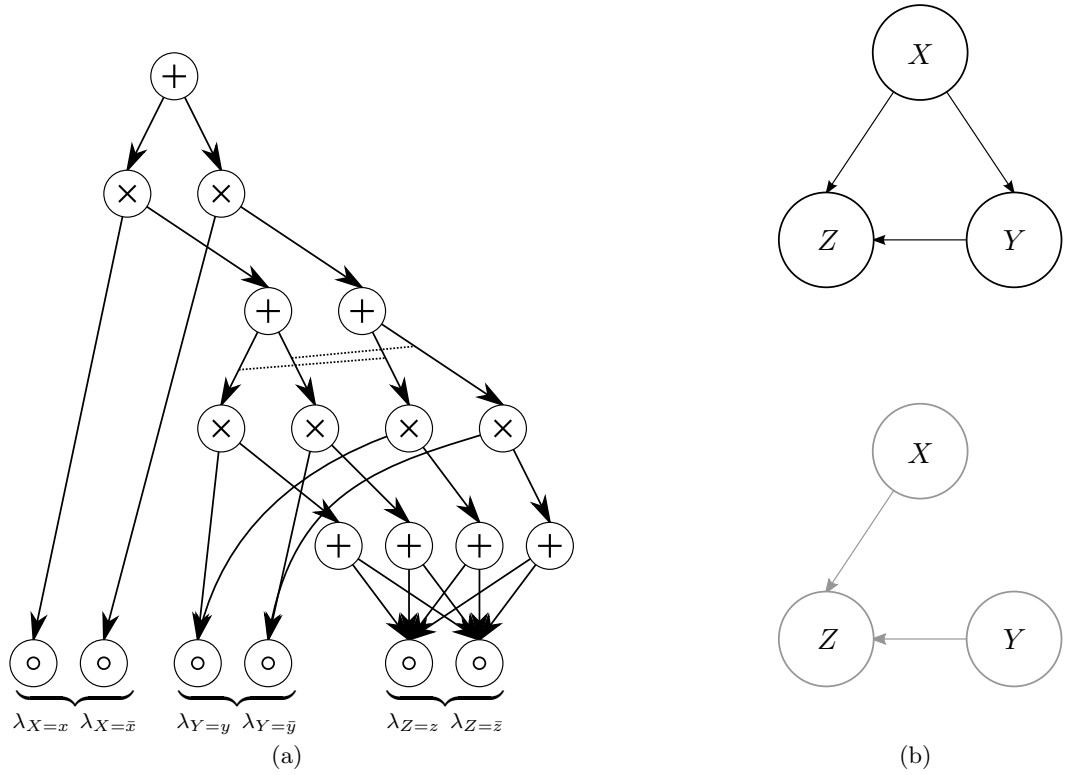


Figure 6.4: Fully connected network. (a): SPN over three binary RVs X, Y, Z, representing the fully connected BN shown in (b), top. When sum-weights associated with edges connected by dashed lines are constrained to be equal, the SPN represents the converging connection shown in (b), bottom.

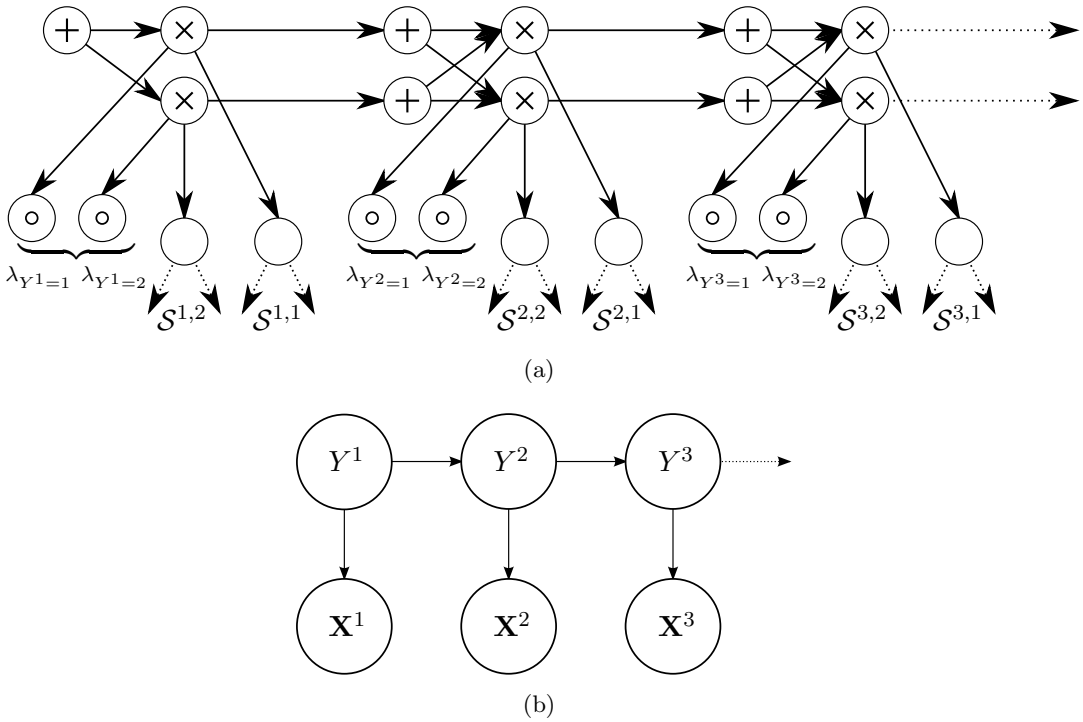


Figure 6.5: HMM with hidden states Y^t and observables $\mathbf{X}^t$ for time t. The SPN shown in (a) represents the HMM shown in (b), here assuming $|\mathbf{val}(\mathbf{Y}^t)| = 2$. $\mathcal{S}^{t,k}$ is the SPN representing the conditional distribution over observables $\mathbf{X}^t$, where according to the definition of HMMs, for all t, $\mathcal{S}^{t,k}$ represent the same distribution.

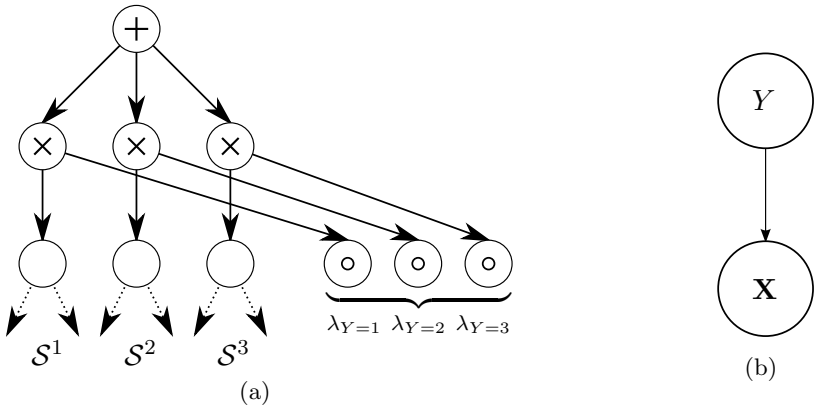


Figure 6.6: Bayes classifier with class RV Y and features $\mathbf{X}$. The SPN shown in (a) represents the Bayes classifier shown in (b), where the number of classes is $|\mathbf{val}(\mathbf{Y})| = 3$. $\mathcal{S}^k$ is the SPN representing the conditional distribution over $\mathbf{X}$ for class k.

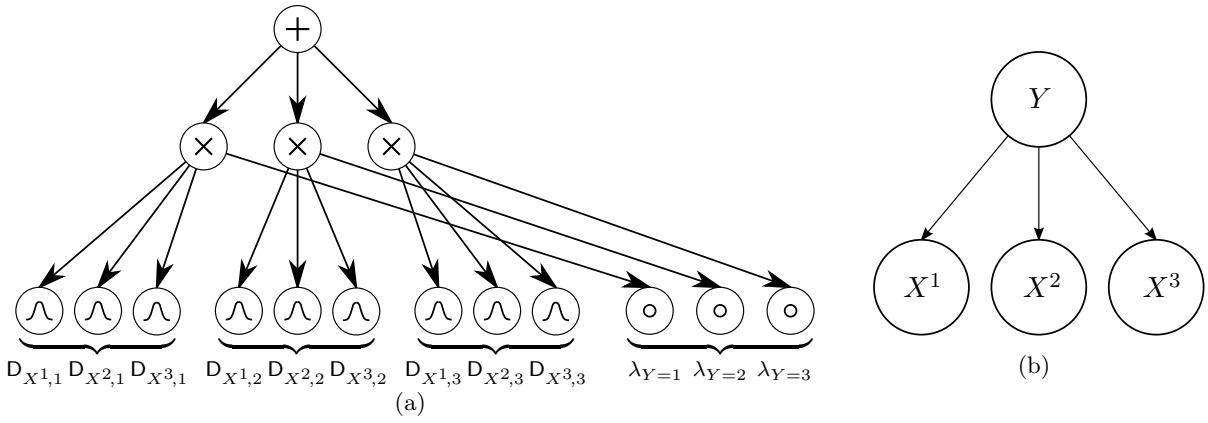


Figure 6.7: Naive Bayes classifier with class RV Y and features $\mathbf{X}$. The SPN shown in (a) represents the Naive Bayes classifier shown in (b), where the number of classes is $|\mathbf{val}(\mathbf{Y})| = 3$ and the observables are X^1, X^2, X^3.

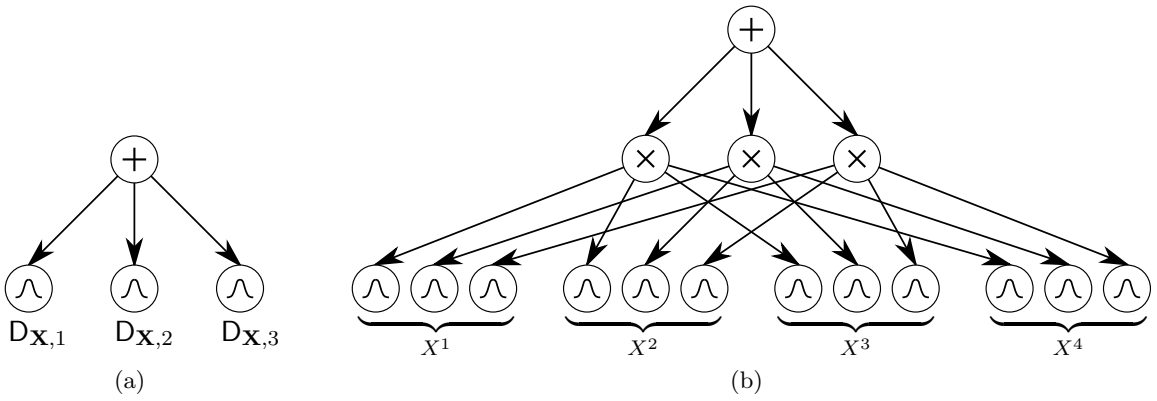


Figure 6.8: Mixture model. (a): SPN with a single sum node, representing a mixture of distributions $\mathsf{D}_{\mathbf{X},1}, \mathsf{D}_{\mathbf{X},2}, \mathsf{D}_{\mathbf{X},3}$ over $\mathbf{X}$. (b): Mixture model, for each component assuming independence among $\mathbf{X} = \{X_1, X_2, X_3, X_3\}$.

6.2 ML Parameter Learning using Gradient Methods

As discussed in Section 4.3.2, finding derivatives in SPNs is easily accomplished using backpropagation. Assume a fixed structure $\mathcal{G}$ for an SPN over $\mathbf{X}$ and a data set $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}\}$ of L i.i.d. samples. The task is to maximize the log-likelihood

$$\max \quad \log \mathcal{L} = \sum_{l=1}^{L} \log \mathcal{S}(\mathbf{x}^{(l)}) \tag{6.1}$$

$$\text{s.t.} \quad \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} = 1, \ \forall \mathsf{S} \tag{6.2}$$

$$w_{\mathsf{S},\mathsf{C}} \geq 0, \qquad \forall \mathsf{S}, \mathsf{C} \in \mathbf{ch}(\mathsf{S}) \tag{6.3}$$

For any node N, the derivative of the log-likelihood of the l^{th} sample is

$$\frac{\partial \log \mathcal{S}}{\partial \mathsf{N}}(\mathbf{x}^{(l)}) = \frac{1}{\mathcal{S}(\mathbf{x}^{(l)})} \frac{\partial \mathcal{S}}{\partial \mathsf{N}}(\mathbf{x}^{(l)}). \tag{6.4}$$

The derivative $\frac{\partial \mathcal{S}}{\partial \mathsf{N}}$ of the log-likelihood of the l^{th} sample can be found by backpropagation, cf. Section 4.3.2. The derivative with respect to the sum weights is given as

$$\frac{\partial \log \mathcal{S}}{\partial w_{\mathsf{S},\mathsf{C}}}(\mathbf{x}^{(l)}) = \frac{\partial \log \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)}) \, \mathsf{C}(\mathbf{x}^{(l)}) = \frac{1}{\mathcal{S}(\mathbf{x}^{(l)})} \frac{\partial \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)}) \, \mathsf{C}(\mathbf{x}^{(l)}), \tag{6.5}$$

and the derivative of the log-likelihood (6.1) is given by

$$\frac{\partial \log \mathcal{L}}{\partial w_{\mathsf{S},\mathsf{C}}} = \sum_{l=1}^{L} \frac{\partial \log \mathcal{S}}{\partial w_{\mathsf{S},\mathsf{C}}}(\mathbf{x}^{(l)}). \tag{6.6}$$

Steepest ascend updates are given as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \nabla \log \mathcal{L}, \tag{6.7}$$

where η is a suitable step-size.

An application of (6.7) will generally not yield locally normalized parameters, i.e. violate constraints (6.2), (6.3). A common way to treat this, is to use *projected gradient methods*, i.e. to project the parameters $\boldsymbol{w}$ back to the feasible set after each gradient step. An efficient algorithm for projecting on the feasible set (6.2), (6.3) is given in [33].

6.3 The EM Algorithm for Sum Weights

The latent RV interpretation of SPNs allows us to use the EM algorithm [30,65] for learning the SPN parameters. Consider an SPN $\mathcal{S}$ over RVs $\mathbf{X}$ with fixed structure and its augmented SPN $\mathcal{S}'$. The augmented SPN represents the same distribution over $\mathbf{X}$ when all latent RVs, i.e. the RV associated with sum nodes, are marginalized. The twin weights are fixed and not optimized. Therefore, in both the original and the augmented SPN the same parameters $\boldsymbol{w}$ are subject to optimization. Since EM is a general method of learning ML parameters when some RVs are unobserved, the augmented SPN is amenable for EM.

This approach was already pointed out in [79], where it was suggested that for RV associated

with S, the marginal posteriors is given as[8]

$$p(Z_{\mathsf{S}} = k \,|\, e) \propto w_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k} \frac{\partial \mathcal{S}(e)}{\partial \mathsf{S}(e)}, \tag{6.8}$$

which should be used for EM updates. However, note (6.8) can not be the claimed marginal posterior, since $\frac{\partial \mathcal{S}(e)}{\partial \mathsf{S}(e)}$ is constant for all k, yielding

$$p(Z_{\mathsf{S}} = k \,|\, e) \propto w_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k}, \tag{6.9}$$

i.e. "prior $w_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k}$ equals posterior after observing evidence e". Furthermore, for EM we do not need posteriors $p(Z_{\mathsf{S}} = k \,|\, e)$, but rather posteriors of Z_{S} and its parents. Therefore, the EM method illustrated in [79] will not work; we derive EM for the SPN sum weights in the following.

Assume a data set $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}\}$ of L i.i.d. samples, i.e. we have complete evidence about all model RVs $\mathbf{X}$. Let $\mathbf{Z} = \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$ be the latent RVs, for which we have no evidence. In this case, an EM update is given as, cf. Section 3.3,

$$\boldsymbol{w}^{\mathrm{t}+1} = \arg\max_{\boldsymbol{w}} \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}^t) \tag{6.10}$$

$$\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}^t) = \sum_{l=1}^{L} \sum_{\mathbf{z} \in \mathbf{val}(\mathbf{Z})} \mathcal{S}(\mathbf{z} \,|\, \mathbf{x}^{(l)}; \boldsymbol{w}^t) \log \mathcal{S}(\mathbf{z}, \mathbf{x}^{(l)}; \boldsymbol{w}). \tag{6.11}$$

To solve the EM iteration, we can use the BN interpretation of augmented SPNs introduced in Section 5.2: Consider a certain sum node S and recall that the latent RV Z_{S} has $\mathbf{Z}_a := \mathbf{Z}_{\mathbf{ancs}(\mathsf{S})\setminus\{\mathsf{S}\}}$ as parents in the corresponding BN. Furthermore, recall that the state space $\mathbf{val}(\mathbf{Z}_a)$ is two-partitioned into $\boldsymbol{\mathcal{Z}}$ and $\bar{\boldsymbol{\mathcal{Z}}}$ and the CPT of Z_{S} is

$$p(Z_{\mathsf{S}} = k \,|\, \mathbf{Z}_a) = \begin{cases} w_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k} & \text{if } \mathbf{Z}_a \in \boldsymbol{\mathcal{Z}} \\ \bar{w}_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k} & \text{if } \mathbf{Z}_a \in \bar{\boldsymbol{\mathcal{Z}}} \end{cases} \tag{6.12}$$

As pointed out, explicitly storing this extremely redundant CPT would be wasteful, so we introduced the binary switching parents Y_{S} which render Z_{S} independent from $\mathbf{Z}_a$, yielding the CPT

$$p(Z_{\mathsf{S}} = k \,|\, Y_{\mathsf{S}}) = \begin{cases} w_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k} & \text{if } Y_{\mathsf{S}} = y_{\mathsf{S}} \\ \bar{w}_{\mathsf{S},\mathsf{C}_{\mathsf{S}}^k} & \text{if } Y_{\mathsf{S}} = y_{\bar{\mathsf{S}}} \end{cases} \tag{6.13}$$

In the BN interpretation, since Y is deterministic with respect to $\mathbf{Z}_a$, the CPT of Y is given as

$$\begin{aligned} p(Y_{\mathsf{S}} = y_{\mathsf{S}} \,|\, \mathbf{Z}_a) &= \begin{cases} 1 & \text{if } \mathbf{Z}_a \in \boldsymbol{\mathcal{Z}} \\ 0 & \text{if } \mathbf{Z}_a \in \bar{\boldsymbol{\mathcal{Z}}} \end{cases} \\ p(Y_{\mathsf{S}} = y_{\bar{\mathsf{S}}} \,|\, \mathbf{Z}_a) &= \begin{cases} 0 & \text{if } \mathbf{Z}_a \in \boldsymbol{\mathcal{Z}} \\ 1 & \text{if } \mathbf{Z}_a \in \bar{\boldsymbol{\mathcal{Z}}} \end{cases} \end{aligned} \tag{6.14}$$

and is not subject to learning.

As pointed out in Section 3.3, for BNs over finite-state RVs, the EM algorithm takes a simple form: In the E-step, we compute the expected sufficient statistics

$$\bar{n}_{x|\mathbf{x_{pa}}}^{X} = \sum_{1=l}^{L} p_{\mathscr{B}}(X, \mathbf{pa}_X \,|\, \mathbf{x}^{(l)}; \boldsymbol{\Theta}^t). \tag{6.15}$$

[8] In (6.8) we use the symbol e to symbolize evidence as in [79].

In the M-step we use the expected sufficient statistics for parameter updates as in (3.46). Thus, all we require for updating the weights of S are the posteriors

$$\mathcal{S}'(Z_\mathsf{S}, Y_\mathsf{S} \,|\, \mathbf{x}^{(l)}), \tag{6.16}$$

where $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$. Moreover, since we keep the twin weights fixed, we just need the posteriors $\mathcal{S}'(Z_\mathsf{S}, Y_\mathsf{S} = y_\mathsf{S} \,|\, \mathbf{x}^{(l)}; \boldsymbol{w})$.

To compute these we apply the differential approach. First, we evaluate the network for sample $\mathbf{x}^{(l)}$. Using back-propagation, we find the derivative

$$\frac{\partial \mathcal{S}'}{\partial \lambda_{Y_\mathsf{S}=y_\mathsf{S}}} = \frac{\partial \mathcal{S}'}{\partial \mathsf{P}}\,\mathsf{S}, \tag{6.17}$$

where P is the common product parent of S and $\lambda_{Y_\mathsf{S}=y_\mathsf{S}}$ in the augmented SPN (see Figure 5.6(b)). Note that $\frac{\partial \mathcal{S}'}{\partial \mathsf{P}}$ equals $\frac{\partial \mathcal{S}}{\partial \mathsf{S}}$ in the original SPN and therefore

$$\mathcal{S}(Y_\mathsf{S} = y_\mathsf{S}, \mathbf{x}^{(l)}) = \frac{\partial \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)})\,\mathsf{S}(\mathbf{x}^{(l)}) \tag{6.18}$$

$$= \frac{\partial \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)}) \sum_{k=1}^{K} \lambda_{Z_\mathsf{S}=k}\, w_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}\, \mathsf{C}_\mathsf{S}^k(\mathbf{x}^{(l)}) \tag{6.19}$$

Equation (6.19) is an extended network polynomial evaluated for $\mathbf{x}^{(l)}$ and differentiated after $\lambda_{Y_\mathsf{S}=y_\mathsf{S}}$. Differentiating it a second time after $\lambda_{Z_\mathsf{S}=k}$ yields

$$\mathcal{S}(Z_\mathsf{S} = k, Y_\mathsf{S} = y_\mathsf{S}, \mathbf{x}^{(l)}) = \frac{\partial \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)})\,\mathsf{C}_\mathsf{S}^k(\mathbf{x}^{(l)})\, w_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}, \tag{6.20}$$

delivering the required posteriors

$$\mathcal{S}(Z_\mathsf{S} = k, Y_\mathsf{S} = y_\mathsf{S} \,|\, \mathbf{x}^{(l)}) = \frac{1}{\mathcal{S}(\mathbf{x}^{(l)})}\,\frac{\partial \mathcal{S}}{\partial \mathsf{S}}(\mathbf{x}^{(l)})\,\mathsf{C}_\mathsf{S}^k(\mathbf{x}^{(l)})\, w_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}. \tag{6.21}$$

Although we derived (6.21) using the augmented SPN, the necessary quantities can be obtained in the *original* SPN. The EM algorithm for SPN weights is shown in Algorithm 7. Here $n_{\mathsf{S},\mathsf{C}}$ are accumulator variables for the expected counts for sum-weights $w_{\mathsf{S},\mathsf{C}}$. An important property of the EM algorithm is *monotonicity*, i.e. that log-likelihood is non-decreasing in each iteration. In Section 7.1 we apply Algorithm 7 to toy data, demonstrating its monotonicity.

Algorithm 7 EM for SPN Weights

1: Initialize $\boldsymbol{w}$
2: **while** not converged **do**
3: $\quad\forall \mathsf{S} \in \mathbf{S}, \forall \mathsf{C} \in \mathbf{ch}(\mathsf{S})\colon n_{\mathsf{S},\mathsf{C}} \leftarrow 0$
4: $\quad$**for** $l = 1 \ldots L$ **do**
5: $\quad\quad$Input $\mathbf{x}^{(l)}$ to $\mathcal{S}$
6: $\quad\quad$Evaluate $\mathcal{S}$ (upward-pass)
7: $\quad\quad$Backprop $\mathcal{S}$ (backward-pass)
8: $\quad\quad\forall \mathsf{S} \in \mathbf{S}, \forall \mathsf{C} \in \mathbf{ch}(\mathsf{S}) : n_{\mathsf{S},\mathsf{C}} \leftarrow n_{\mathsf{S},\mathsf{C}} + \frac{1}{\mathcal{S}}\,\frac{\partial \mathcal{S}}{\partial \mathsf{S}}\,\mathsf{C}\, w_{\mathsf{S},\mathsf{C}}$
9: $\quad$**end for**
10: $\quad\forall \mathsf{S} \in \mathbf{S}, \forall \mathsf{C} \in \mathbf{ch}(\mathsf{S}) : w_{\mathsf{S},\mathsf{C}} \leftarrow \frac{n_{\mathsf{S},\mathsf{C}}}{\sum_{\mathsf{C}' \in \mathbf{ch}(\mathsf{S})} n_{\mathsf{S},\mathsf{C}'}}$
11: **end while**

So far we considered complete evidence, i.e. complete data cases $\mathbf{x}^{(l)}$. Algorithm 7 is easily adapted to partial evidence, e.g. when we have samples with missing values. To this end split $\mathbf{X}$

according the l^{th} data case into $\mathbf{Y}^{(l)}$, $\mathbf{Y}^{(l)'}$, where for $\mathbf{Y}^{(l)'}$ we have complete evidence $\mathbf{y}^{(l)'}$ and for $\mathbf{Y}^{(l)}$ we have partial evidence $\boldsymbol{\mathcal{Y}}^{(l)}$. In step 5, we simply input the evidence $\mathbf{y}^{(l)'}, \boldsymbol{\mathcal{Y}}^{(l)}$ to the SPN, as illustrated in Section 4.4, using inference scenario (2.). The other steps of Algorithm 7 remain the same.

Both projected gradient and EM aim to maximize the data-likelihood. Since SPNs can be seen as deep hierarchical latent RV models, the log-likelihood is generally non-convex, i.e. we can only expect to find a local maximum. As discussed in Section 3.3, for finite-state BNs we can obtain globally optimal ML parameters in closed form. Furthermore, as discussed in Section 6.1, SPNs can represent many classical models, as for instance finite-state BNs. Since the weights of these SPNs directly correspond to the BN parameters, there must exist a class of SPNs for which ML parameter learning is easy. In the following section, we discuss such a class.

6.4 Selective Sum-Product Networks

Since all nodes in an SPN represent distributions, it is natural to define the support $\mathbf{sup}(\mathsf{N})$, i.e. the largest subset of $\mathbf{val}(\mathbf{sc}(\mathsf{N}))$ such that $\mathcal{S}_{\mathsf{N}}$ has positive output for each element in this set. The support of a node depends on the *structure* and the *parameters* of the SPN. We want to introduce a modified notion of support in SPNs which does not depend on its parametrization. It is easy to see that if $\mathbf{y} \in \mathbf{sup}(\mathsf{N})$ for some parametrization $\boldsymbol{w}$, then also $\mathbf{y} \in \mathbf{sup}(\mathsf{N})$ for some other parametrization $\boldsymbol{w}'$ with *strictly* positive parameters, e.g. uniform parameters for all sum nodes. Therefore, we define the *inherent support* of N, denoted as $\mathbf{isup}(\mathsf{N})$, as the support when uniform parameters are used for all sum nodes. The inherent support depends only on the used input distributions and the SPN structure.

We define selective sum nodes and selective SPNs as follows.

Definition 6.1. *A sum node* S *is* selective *if*

$$\mathbf{isup}(\mathsf{C}') \cap \mathbf{isup}(\mathsf{C}'') = \emptyset, \forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S}), \mathsf{C}' \neq \mathsf{C}''. \tag{6.22}$$

An SPN is selective *if every sum node in the SPN is selective.*

This notion was actually introduced in the context of arithmetic circuits [27, 64] under the term *determinism*. However, the term "deterministic SPN" is somewhat misleading, since it suggests that these SPNs partly model deterministic relations among the model RVs. This is in general not the case, so we deliberately use the term selective SPN here. In words, an SPN is selective when for each possible input and each possible parametrization, each sum node has at most one child with positive output. When using the latent RV interpretation of sum nodes (cf. Chapter 5), we already see that selective SPNs allow us to obtain closed form ML parameters: Since for an arbitrary sample and for each sum node at most one child is non-zero, the state of the corresponding latent RVs is deterministic with respect to the model RVs. Thus, in selective SPNs, the latent RVs are actually *observed*, which allows us to obtain closed form ML parameters.

In [64], a less general notion of selectivity (actually determinism) was used, which we call *regular selectivity*. To define this notion, recall that $\mathbf{D}^X$ is the set of distributions $\mathsf{D}_{\mathbf{Y}}$ with $X \in \mathbf{Y}$, cf. Section 4.4. Let $I_X(\mathsf{N}) \subseteq \{1, \dots, |\mathbf{D}^X|\}$ denote the indices of distribution nodes reachable by N, i.e. $\mathsf{D}_{X,k} \in \mathbf{desc}(\mathsf{N}) \Leftrightarrow k \in I_X(\mathsf{N})$.

Definition 6.2. *An SPN* $\mathcal{S}$ *is* regular selective *if the following two conditions are fulfilled:*

1. *The distribution nodes have non-overlapping support, i.e.* $\forall X \in \mathbf{X} : \forall i \neq j : \mathbf{sup}(\mathsf{D}_{X,i}) \cap \mathbf{sup}(\mathsf{D}_{X,j}) = \emptyset$.

2. *Each sum node* S *is* regular selective *with respect to some* $X \in \mathbf{sc}(\mathsf{S})$*, denoted as* $\mathsf{S} \rightsquigarrow X$ *and defined as* $\forall \mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{S}), \mathsf{C}' \neq \mathsf{C}''$*:*

   a) $I_X(\mathsf{C}') \cap I_X(\mathsf{C}'') = \emptyset$

   b) $\forall Y \in \mathbf{sc}(\mathsf{S}), Y \neq X : I_Y(\mathsf{C}') = I_Y(\mathsf{C}'')$

Regular selectivity implies selectivity but not vice versa. Selectivity or regular selectivity is, besides completeness and decomposability, a further constraint on the SPN structure. This raises the question which distributions can be modeled with selective SPNs. In the next section, we discuss several examples of regular selective SPNs and show that they are a rather flexible model class. Selectivity allows us to find globally optimal ML parameters in *closed form*, as discussed in Section 6.4.2. Furthermore, the log-likelihood function can be evaluated efficiently, since it decomposes into a sum of terms associated with sum nodes. This is useful when considering structure learning of selective SPNs [72].

6.4.1 Regular Selective SPNs

The SPN representations of the canonical examples of BNs over three binary RVs, shown in Figures 6.1, 6.2, 6.3 and 6.4, are example of regular selective SPNs. Therefore, regular selective SPNs can represent any distribution over finite-state RVs, since BNs can represent any distributions over finite-state RVs.

However, regular selective SPNs are capable to represent more advanced models than BNs. In Figure 6.9 we see a regular selective SPN representing a *BN with context-specific independence* (CSI): When $X = x$ the conditional distribution is a BN $Y \to Z$; for $X = \bar{x}$, the conditional distribution is a product of marginals. Therefore, Y and Z are independent in the context $X = \bar{x}$. BNs with CSI are not new and discussed e.g. in [10, 19]. What is remarkable, however, is that when represented as regular selective SPNs, inference for BNs with CSI is treated exactly the same way as for classic BNs. Typically, classic inference methods for BNs, like the junction tree algorithm, can not be automatically used for BNs with CSI.

Furthermore, regular selective SPNs can represent *nested multi-nets*. Figure 6.10 shows a regular selective SPN which uses a BN $Y \to Z$ for the context $X = x$ and a BN $Y \leftarrow Z$ for the context $X = \bar{x}$. Since the network structure of Y and Z depend on the state of X, this SPN represents a multi-net. When considering more than 3 variables, the context-specific models are not restricted to be BNs but can themselves be multi-nets. In that way one can build more or less arbitrarily nested multi-nets. Note that BNs with CSI are actually a special case of nested multi-nets. The concept of nested multi-nets is potentially powerful and can be advantageous to condition on different variables in different contexts.

As discussed in Chapter 5, we can associate a latent RV Z_{S} with each sum node S. In the context of selective SPNs, the states of Z_{S} represent events of the form $\mathbf{sc}(\mathsf{S}) \in \mathbf{isup}(\mathsf{C})$. Therefore, the latent RVs are *functions* of model RVs and thus observed, when all model RVs are observed. In particular for *regular* selective SPNs, they represent a partition of (a subset of) the state space of a single model RV. For binary RVs, as in the examples considered so far, there is only one partition of the state space and the RVs associated with sum nodes simply 'imitate' the binary RVs. In Figure 6.11 we see an example of a regular selective SPN containing an RV X with three states. The topmost sum node is regular selective with respect to X and its two children represent the events $X \in \{x_1\}$ and $X \in \{x_2, x_3\}$. The next sum node represents a partition of the two states of RV Y. Finally, in the branch $X \in \{x_2, x_3\}, Y = y_2$ we find a sum node which is again regular selective with respect to X and further splits the two states x_2 and x_3. This example shows that regular selective SPNs can be interpreted like a CPT organized as decision graph [10, 19]. However, in the classical work on CPTs represented as decision graphs, each RV is allowed to appear only *once* in the diagram.

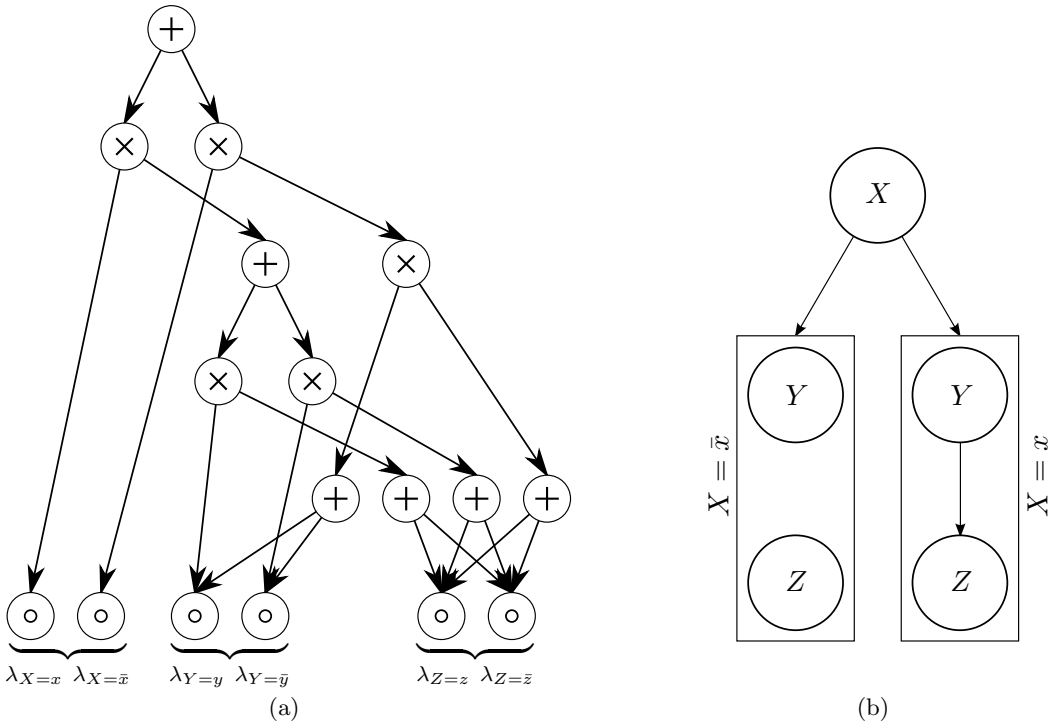


Figure 6.9: Regular selective SPN with CSI. (a): Regular selective SPN over RVs X, Y, Z, representing the BN with CSI in (b). When $X = \bar{x}$, the conditional distribution of Y, Z is a product of marginals, i.e. Y and Z are independent in the context $X = \bar{x}$. When $X = x$, the conditional distribution of Y, Z corresponds to a BN $Y \to Z$.

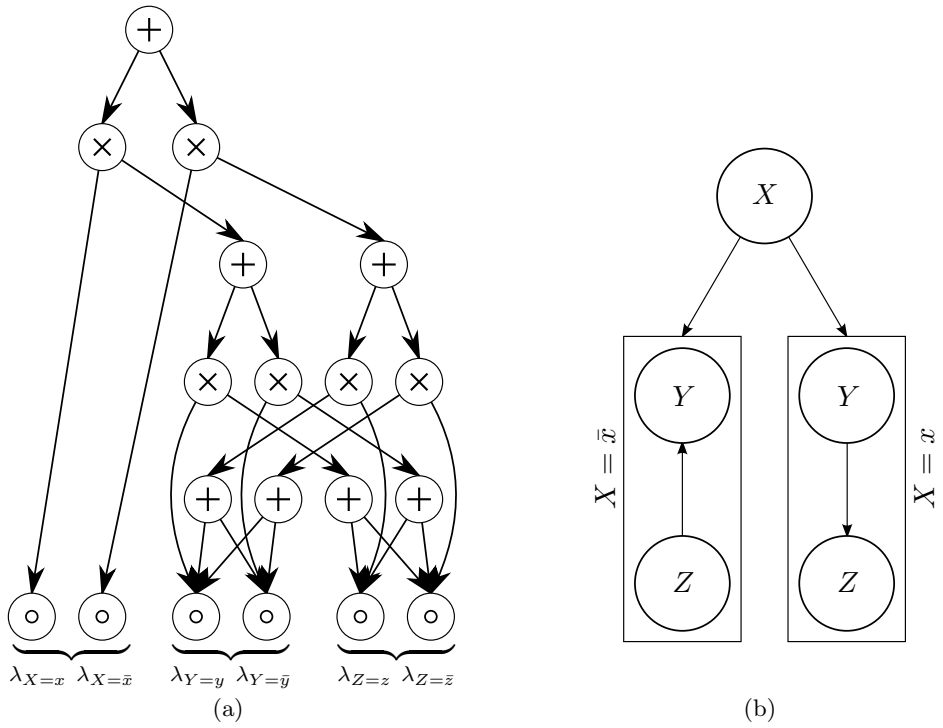


Figure 6.10: Regular selective SPN representing a multi-net. (a): Regular selective SPN over RVs X, Y, Z, representing the multi-net in (b). When $X = x$, the conditional distribution over Y, Z is a BN $Y \to Z$. When $X = \bar{x}$, the conditional distribution over Y, Z is a BN $Z \to Y$.

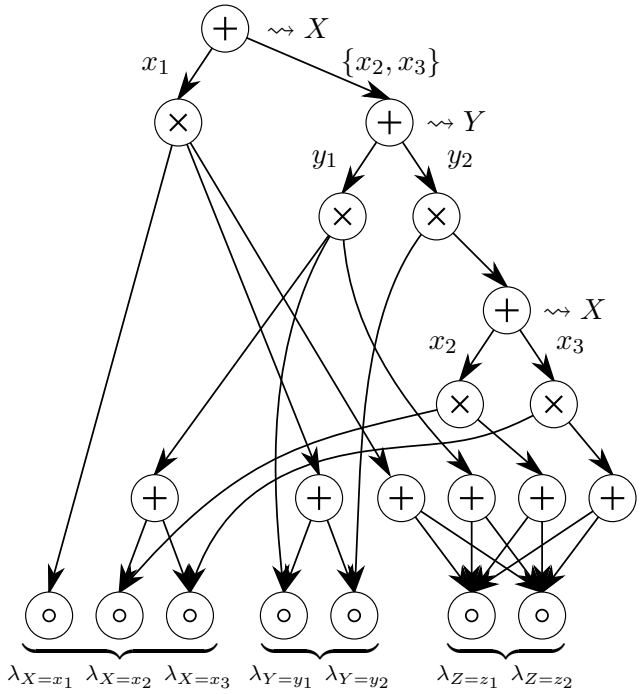


Figure 6.11: Regular selective SPN whose sum nodes represent a partition of the state space of the model RVs. The top-most sum node is regular selective with respect to X, and its children represent the partition $\{x_1\}$ and $\{x_2, x_3\}$ of the state space of X. Further below we see another sum node, splitting $\{x_2, x_3\}$ further into $\{x_2\}$ and $\{x_3\}$.

Furthermore, regular selective SPNs naturally allow to share components among different parts of the network. In Figure 6.12 we see a simple example adapted from Figure 6.9 where the two conditional models over $\{Y, Z\}$ share a sum representing a marginal over Z. This model uses one marginal over Z for the context $X = x, Y = y$ and another marginal for all other contexts. This implements a form of *parameter tying*, a technique which is widely used in classical PGMs, e.g. for HMM in automatic speech recognition.

Selective SPNs are highly related to probabilistic decision graphs (PDGs) [46]. The main difference is that PDGs, similar as BNs, are restricted to a fixed variable order. Probabilistic Sentential Decision Diagrams (PSDD) [50] are another concept related to selective SPNs.

We see that regular selective SPNs, although being a restricted class of SPNs, still generalize BNs over finite-state RVs. We want to point out again to the fact that the concepts presented here – CSI, nested multi-nets, decision diagrams with multiple RV appearance, and component sharing – do not need any extra treatment in the inference phase, but are naturally treated within the SPN framework. The examples discussed in this section are all SPNs over finite-state SPNs and used IVs as input distributions. Similarly, for continuous RVs we can use any input distributions with non-overlapping support, e.g. uniform distributions over non-overlapping intervals.

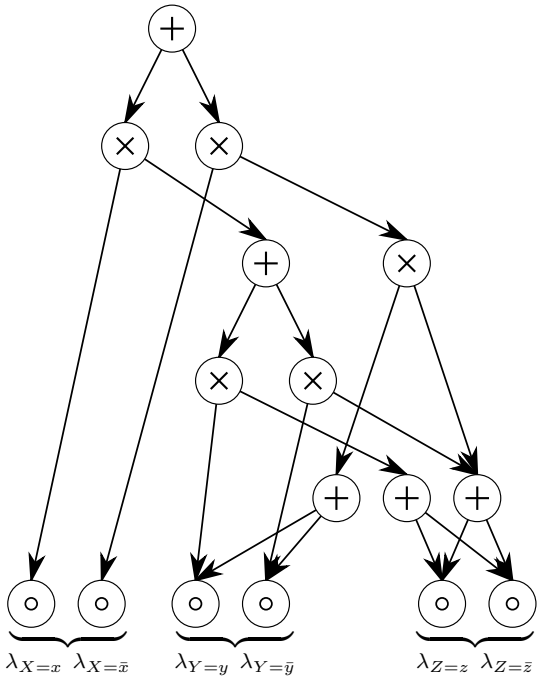


Figure 6.12: Regular selective SPN with shared parts. This SPN is an adapted version from Figure 6.9, where the conditional models for x and $\bar{x}$ share a sum node representing a marginal over Z.

6.4.2 Maximum Likelihood Parameters

We saw that regular selective SPNs can represent BNs over finite-state RVs. In this case, we can easily obtain closed form ML parameters, cf. Section 3.3. However, as discussed in the last section, regular selective SPNs can represent more advanced models than BNs. In these cases, it is not clear how to obtain ML parameters. In this section we derive a closed form solution for the ML parameters of the more general class of selective SPNs. It turns out that ML parameters are obtained in very similar way as for BNs over finite-state RVs.

Assume that we have a given selective SPN structure $\mathcal{G}$ and a set of completely observed i.i.d. samples $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}\}$. We wish to maximize the likelihood:

$$\mathcal{L}(\mathcal{D}; \boldsymbol{w}) = \prod_{l=1}^{L} \mathcal{S}(\mathbf{x}^{(l)}). \tag{6.23}$$

We assume that there *exists* a set of SPN parameters $\boldsymbol{w}$ such that $\mathcal{S}(\mathbf{x}^{(l)}) > 0$, $l = 1, \ldots, L$. Otherwise, the likelihood would be zero for all parameter sets, i.e. every parameter set would be optimal. The following definitions will help us to derive the ML parameters.

Definition 6.3. *Let $\mathcal{S}$ be an SPN and N be a node in $\mathcal{S}$. A* calculation path $\Pi_{\mathbf{x}}(\mathsf{N})$ *for node N and sample $\mathbf{x}$ is a path $\Pi_{\mathbf{x}}(\mathsf{N}) = (\mathsf{N}_1, \ldots, \mathsf{N}_J)$, where N_1 is the root node and $\mathsf{N}_J = \mathsf{N}$, with $\mathsf{N}_j(\mathbf{x}) > 0$, $j = 1, \ldots, J$.*

Definition 6.4. *Let $\mathcal{S}$ be a complete, decomposable and selective SPN. The* calculation tree $\mathcal{T}_{\mathbf{x}}$ *for sample $\mathbf{x}$ is the SPN induced by all nodes for which there exists a calculation path.*

It is easy to see that for each sample $\mathbf{x}$ we have $\mathcal{S}(\mathbf{x}) = \mathcal{T}_{\mathbf{x}}(\mathbf{x})$. The following lemma will help us in our discussion.

Lemma 6.1. *For each complete, decomposable and selective SPN $\mathcal{S}$ and each sample $\mathbf{x}$, $\mathcal{T}_{\mathbf{x}}$ is a tree.*

Calculation trees allow us to write the probability of the l^{th} sample as

$$\mathcal{S}(\mathbf{x}^{(l)}) = \left(\prod_{\mathsf{S} \in \mathbf{S}(\mathcal{S})} \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}^{u(l,\mathsf{S},\mathsf{C})} \right) \left(\prod_{X \in \mathbf{X}} \prod_{k=1}^{|\mathbf{D}^X|} \mathsf{D}_{X,k}^{u(l,X,k)} \right), \tag{6.24}$$

where $u(l, \mathsf{S}, \mathsf{C}) := \mathbb{1}(\mathsf{S} \in \mathcal{T}_{\mathbf{x}^{(l)}} \wedge \mathsf{C} \in \mathcal{T}_{\mathbf{x}^{(l)}})$ and $u(l, X, k) := \mathbb{1}(\mathsf{D}_{X,k} \in \mathcal{T}_{\mathbf{x}^{(l)}})$. Since the sample probability factorizes over sum nodes, sum children and distribution nodes, the likelihood (6.23) can be written as

$$L(\mathcal{D}; \boldsymbol{w}) = \left(\prod_{\mathsf{S} \in \mathbf{S}(\mathcal{S})} \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}^{n(\mathsf{S},\mathsf{C})} \right) \left(\prod_{X \in \mathbf{X}} \prod_{k=1}^{|\mathbf{D}^X|} \mathsf{D}_{X,k}^{n(X,k)} \right), \tag{6.25}$$

where $n(\mathsf{S}, \mathsf{C}) = \sum_{l=1}^{L} u(l, \mathsf{S}, \mathsf{C})$ and $n(X, k) = \sum_{l=1}^{L} u(l, X, k)$. Using the results for ML parameter estimation in BNs (cf. (3.46)) [52, 77], the ML parameters are given as

$$w_{\mathsf{S},\mathsf{C}} = \begin{cases} \frac{n(\mathsf{S},\mathsf{C})}{n(\mathsf{S})} & \text{if } n(\mathsf{S}) \neq 0 \\ \frac{1}{|\mathbf{ch}(\mathsf{S})|} & \text{otherwise,} \end{cases} \tag{6.26}$$

where $n(\mathsf{S}) = \sum_{\mathsf{C}' \in \mathbf{ch}(\mathsf{S})} n(\mathsf{S}, \mathsf{C}')$. Furthermore, one can apply Laplace smoothing to the ML solution and also incorporate a Dirichlet prior on the parameters.

6.5 Structure Learning

As discussed, for a fixed SPN structure we can use gradient methods and the EM-algorithm for ML parameters learning. Furthermore, in the case of selective SPNs, we can obtain closed form ML parameters. Obtaining a suitable SPN structure is more delicate. Here we summarize some approaches proposed in literature so far.

In [79] it was proposed to define a general SPN structure exploiting domain knowledge and to cast effective structure learning as sparse parameter learning. In particular, an architecture for image data was proposed, or more generally a structure for data arranged in rectangles. This architecture uses explicit knowledge about *locality* or *neighborhood* of RVs.

First, it assigns a single sum node to the overall image, which is the root of the SPN and represents a distribution over images. Then the overall image is split into all possible partitions of two sub-rectangles along the two dimensions. Every sub-rectangle is equipped with several sum nodes, representing distributions over the sub-rectangle. This process is recursively repeated, splitting each (sub-)rectangle into two sub-rectangles in all possible ways. All pairs of sum nodes from two sub-rectangles are connected as children of a product node, which in turn is connected as child of all sum nodes of the parent rectangle. Since this structure grows quickly with the size of the image – for square-shaped images of length l the number of sub-rectangles is $\sum_l l^3$ [1] – it was proposed to split rectangles in a coarser way, i.e. iterate along each dimension with a larger step size. We denote this SPN architecture as Poon-Domingos (PD) architecture.

Dennis and Ventura [31] proposed to generalize the PD architecture to be applicable to sets of RVs which are not naturally arranged in rectangles and which do not have a natural notion of locality. To this end, they organized sets of RVs, called *regions* in this context, into pairwise splits of sub-regions. This regions are not necessarily arranged as rectangles and are found in a data driven way by finding a k-means clustering of regions, i.e. applying k-means on "the transposed data-matrix". We call this approach the Dennis-Venture (DV) architecture.

In [71] it was noted that the DV architecture, although not requiring locality of RVs, still implicitly assumes that RVs are dependent by *positive correlation*. A greedy learning approach

was proposed, *merging* regions of RVs from bottom-up instead of splitting regions from top-down as proposed in [31, 79]. Here the merging process is guided by independence tests and thus not relying on positive correlation of the RVs.

In [40] a top-down scheme for learning SPN structures was proposed. They applied probabilistic clustering on data samples corresponding to sum nodes and clustering on RVs using independence tests, corresponding to product nodes. Thus, this approach also does not rely on positive correlation of RVs.

In [84], the approach in [40] was refined in order to use input distributions over larger scopes. These input distributions are MNs represented as ACs [63].

In [72], a greedy hill-climbing algorithm was proposed to learn tree-shaped regular selective SPNs.

7 Experiments and Applications

7.1 EM Algorithm

In Section 6.3 we derived the EM algorithm for sum-weights in SPNs (Algorithm 7). A correctly derived EM algorithm must exhibit monotonicity, i.e. the log-likelihood must be non-decreasing in each iteration. To demonstrate monotonicity, we applied Algorithm 7 in a toy example. We consider sets of 9, 16 and 25 RVs, and construct SPNs using the PD architecture (cf. Section 6.5) by arranging the RVs in a 3×3, 4×4 and 5×5 array, respectively. We use $K = 5$ sum nodes per rectangle and $G = 10$ Gaussian PDFs as input distributions, whose means are set by histogram quantiles and whose standard deviations are uniformly set to 1 [31, 71, 79]. For each of the used SPN structures, the number of sum nodes, sum-weights and product nodes, and the length of the longest path from the root node to some input distribution is shown in Table 7.1. As data we

Table 7.1: Statistics of toy SPN structures.

structure	# sums	#sum-weights	#products	longest path
3×3	131	11600	2400	9
4×4	416	39400	8000	10
5×5	996	104200	21000	10

used 9, 16 and 25 randomly selected PCA features (among the first 100 features) of 1000 samples from the MNIST data set, normalized to zero mean and unit variance. Furthermore, we added white Gaussian noise with standard deviation 0.1. We initialized the sum-weights randomly using a Dirichlet distributions with uniform parameters $\alpha = 1$, i.e. the uniform distribution on the standard simplex. For each structure, we executed the EM algorithm for 100 iterations and performed 10 random restarts, i.e. in total we ran the EM algorithm 30 times. Figure 7.1 shows the log-likelihood of these runs, normalized by the number of RVs. We see that Algorithm 7 indeed monotonously increases the training likelihood – in this experiment, the minimal increase of log-likelihood within one iteration was 0.2.

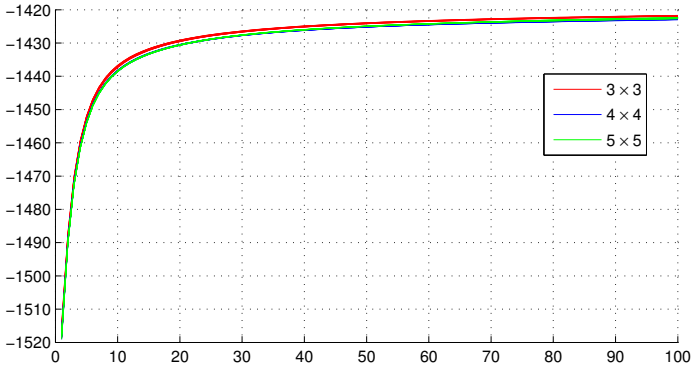


Figure 7.1: Log-likelihood as a function of the EM iteration number.

7.2 MPE Inference

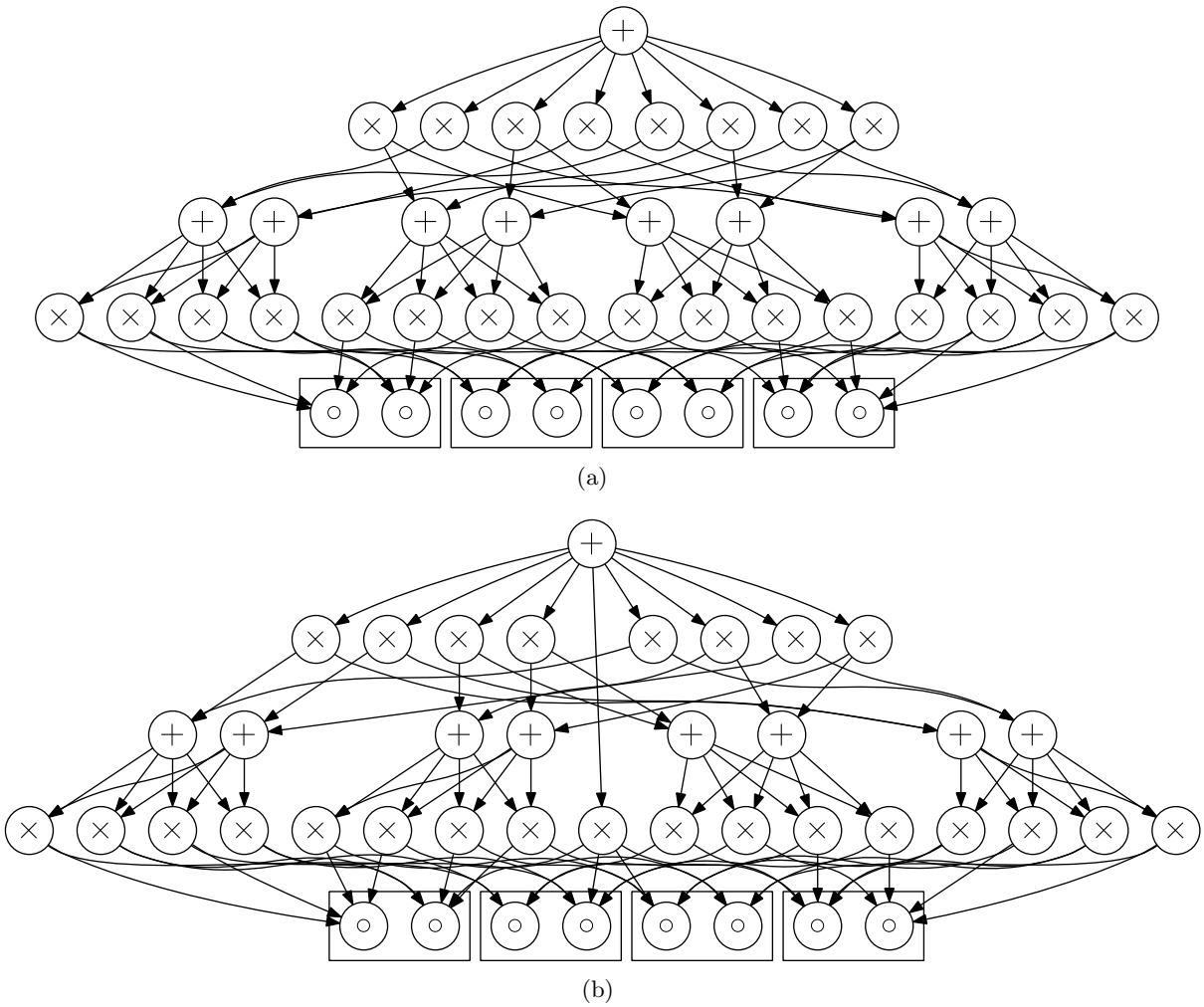
In Section 5.3, we discussed MPE inference and showed that Algorithm 5 delivers an MPE solution for augmented SPNs. We further proposed Algorithm 6 which can be applied to the original SPN but delivers an MPE solution for the augmented SPN. As a sanity check, we constructed a toy SPN over 4 binary RVs and the corresponding augmented SPN, where MPE inference by exhaustive enumeration is tractable. To define the structure of our toy SPN, we arranged the 4 RVs as a 2×2 grid and used the PD architecture using 2 sum nodes per rectangle (cf. Section 6.5). This structure is shown in Figure 7.2(a). However, since this SPN regularly structured, it does not fall prey to the low-depth bias and standard max-backtracking yields the same solution as Algorithm 6. In order to demonstrate the low-depth bias, we add an additional product node as child of the sum root node, whose children are IVs for each of the 4 RVs. The resulting structure is shown in Figure 7.2(b).

We equipped this toy SPN with random sum-weights, drawn from a Dirichlet distribution with uniform α-parameter, where $\alpha \in \{0.5, 1, 2\}$. Then we explicitly constructed the augmented SPN using Algorithm 3, where we used uniform weights for the twin sums. The augmented SPN contains 9 additional latent RVs, 8 with 4 states and 1 with 9 states.

We performed MPE inference by exhaustive enumeration in the augmented SPN, i.e. computing the SPN distribution for all 9437184 possible inputs and selecting the max. We applied the back-tracking Algorithm 5 in the augmented SPN (denoted as BACK-AUG) and Algorithm 6 in the original SPN (denoted as BACK-MPE). We also executed Algorithm 6 in the original SPN, but with correction weights constantly set to 1, which corresponds to the back-tracking algorithm in [79] (denoted as BACK-ORIG). Furthermore, we performed the sum approximation of MPE (denoted as SUM, cf. Section 5.3). For each α, we repeated this experiment 100 times. Table 7.2 shows the average/maximal deviation in log-probability from the exhaustive MPE solution. A deviation of 0 means that the algorithm delivered an optimal solution, while a negative value indicates a sub-optimal solution. We see that BACK-AUG and BACK-MPE always delivered a correct MPE solution in this toy example, while BACK-ORIG and SUM do not guarantee an optimal solution. The minimum deviation was 0 for all algorithms in this example, i.e. all algorithms delivered a correct MPE solutions in some of the 100 runs.

Table 7.2: Results for MPE inference in toy SPN, comparing to exhaustive MPE solution.

	BACK-AUG	BACK-MPE	BACK-ORIG	SUM
$\alpha = 0.5$	0.000/0.000	0.000/0.000	-0.533/-2.763	-0.183/-2.072
$\alpha = 1$	0.000/0.000	0.000/0.000	-0.951/-2.680	-0.197/-1.675
$\alpha = 2$	0.000/0.000	0.000/0.000	-1.415/-2.755	-0.168/-1.303



(a)

(b)

Figure 7.2: Toy SPN for MPE inference. (a): SPN structure defined by PD architecture for 4 binary RVs arranged as 2×2 grid. (b): Same structure as in (a), with additional product node as child of root node. (Graphs visualized using Graphviz *[38]).*

We furthermore constructed a larger toy SPN, again using the PD architecture on 16 RVs with 4 states, organized in a 4×4 grid, and using 4 sums per rectangle. We again added an additional product node as child of the sum root. In this network, exhaustive enumeration is not tractable anymore. Therefore, we compare the solutions to BACK-AUG. Table 7.3 shows the minimal/average/maximal deviation in terms of log-likelihood. We see that BACK-AUG and BACK-MPE always return a solution with equal probability, while BACK-ORGIG and SUM are evidently suboptimal. Furthermore, we see that SUM seems to be the better approximation than BACK-ORIG, since it delivers an equivalent solution to BACK-AUG in some cases. Also its average/maximal deviation is smaller than in BACK-ORIG.

Table 7.3: Results for MPE inference in larger toy SPN, comparing to BACK-AUG.

	BACK-MPE	BACK-ORIG	SUM
$\alpha = 0.5$	0.000/0.000/0.000	-24.680/-28.498/-34.760	0.000/-2.674/-24.890
$\alpha = 1$	0.000/0.000/0.000	-19.920/-22.782/-26.230	0.000/-2.055/-19.920
$\alpha = 2$	0.000/0.000/0.000	-15.930/-17.795/-22.040	0.000/-1.976/-16.180

7.3 Face Image Completion

In [79], it was demonstrated that MPE inference in SPNs achieves convincing results on the ill-posed problem of image completion, i.e. reconstructing occluded parts of face images. To this end, SPNs are trained on the ORL face image data set [86]. This data set consists of 400 face images, taken from 40 different persons, each with 10 images. The center 64×64 region of the original images was extracted[9] and split into a training set of 35 persons (total 350 images) and a test set of 5 persons (total 50 images). We transformed the training images to zero mean and unit variance and applied the same transformation to the test images. We trained SPNs with the method of Poon and Domingos (PD) [79], the method of Dennis and Ventura (DV) [31], and greedy-part-wise learning (Merge) [71]. We model single pixels with G Gaussian PDFs, where the means are set by the averages of histogram quantiles, and the standard deviation is uniformly set to 1. We set the number of Gaussians $G = 10$ for all three algorithms. All three algorithms find a recursive decomposition of the overall image into sub-regions and each region is equipped with K sum nodes, representing a dictionary of distributions over the respective region. As in [31, 79] we use $K = 20$ for all three algorithms.

In [31, 71, 79], SUM was used for reconstructing the missing image parts. As discussed in Section 5.3 and shown in the last section, this does not correspond to MPE-inference in the augmented SPN. However, it is an arguable approximation for MPE inference in the original SPN. Here, additionally to SUM we use BACK-MPE and BACK-ORIG, where the latter was proposed in [79] for MPE inference. As discussed in Section 5.3, this corresponds to MPE-inference in the augmented SPN when using deterministic twin weights, which suffers from the low-depth bias. As a base-line we replace missing pixel by the pixel values of the mean image over all training images (MEAN).

Figures 7.3, 7.4, 7.5 and 7.6 show results for face image completion, when the left, top, right and bottom halves of the images are missing, respectively. All models (PD, DV, Merge) and all inference methods (BACK-ORIG, SUM, BACK-MPE) perform reasonable for reconstructing missing image parts. The artifacts introduced by BACK-ORIG seem to be often somewhat stronger than the artifacts introduced by SUM and BACK-MPE. Table 7.5 shows reconstruction SNRs for all models and all inference methods. BACK-ORIG consistently performs worse (or as well as) the other two methods. BACK-MPE and SUM are roughly on par. This observation is consistent with the remark in [79], that SUM performed better than BACK-ORIG in the experiments. Note that MEAN consistently has the highest SNR. This probably comes from the artifacts with high energy produced by the SPN reconstructions.

Table 7.4 shows training and test log-likelihoods for models PD, DV and Merge. While Merge achieves the lowest training likelihood, it achieves the highest likelihood on the test set, stating that Merge generalized best in this task.

Table 7.4: Log-likelihoods on training and test set, normalized by number of samples.

	Train	Test
PD	-4287.82	-5068.97
DV	-4356.41	-4673.73
Merge	-4493.46	-4667.04

[9] Obtained from [78].

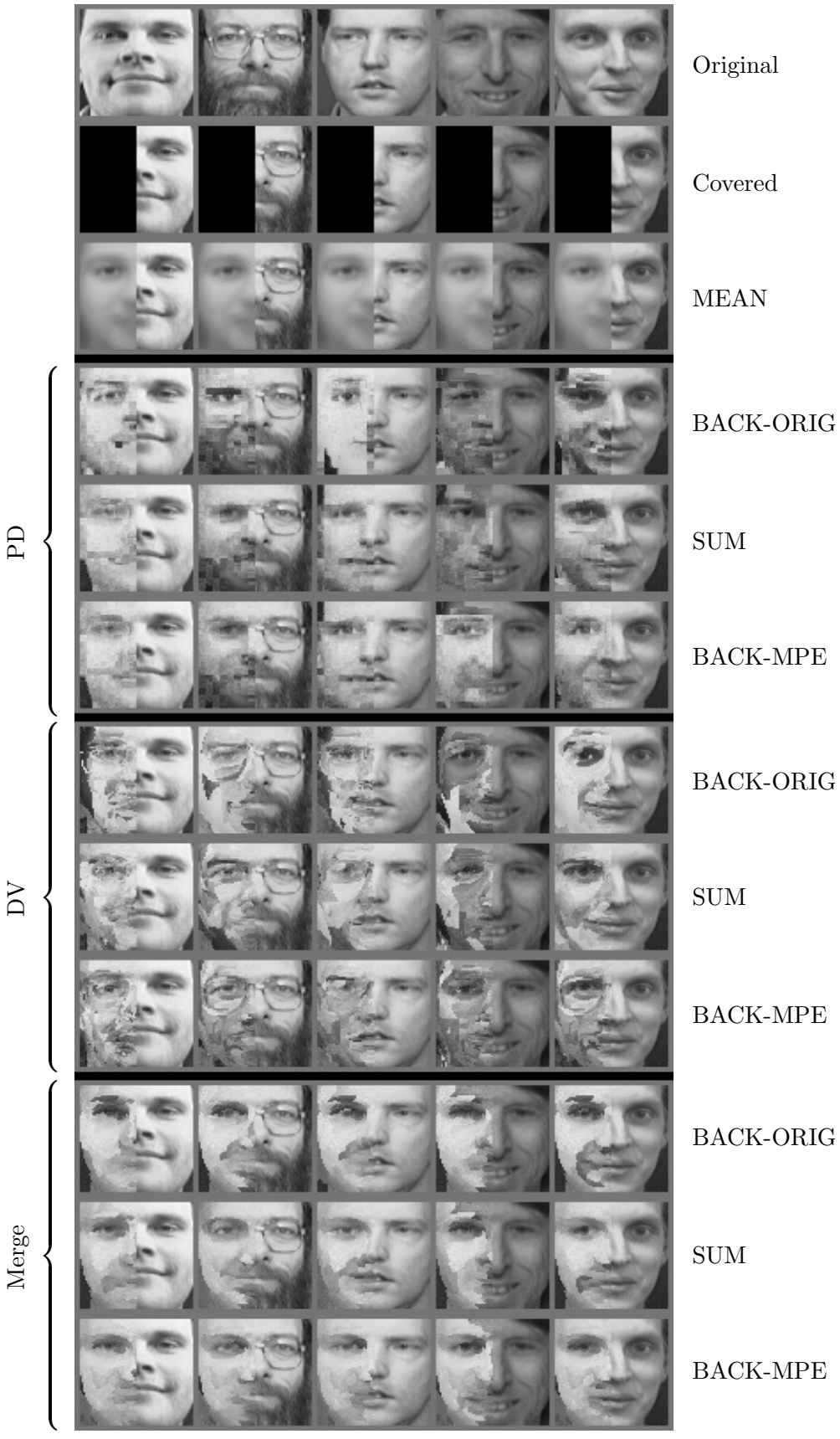


Figure 7.3: Examples of face image reconstructions, left half is covered.

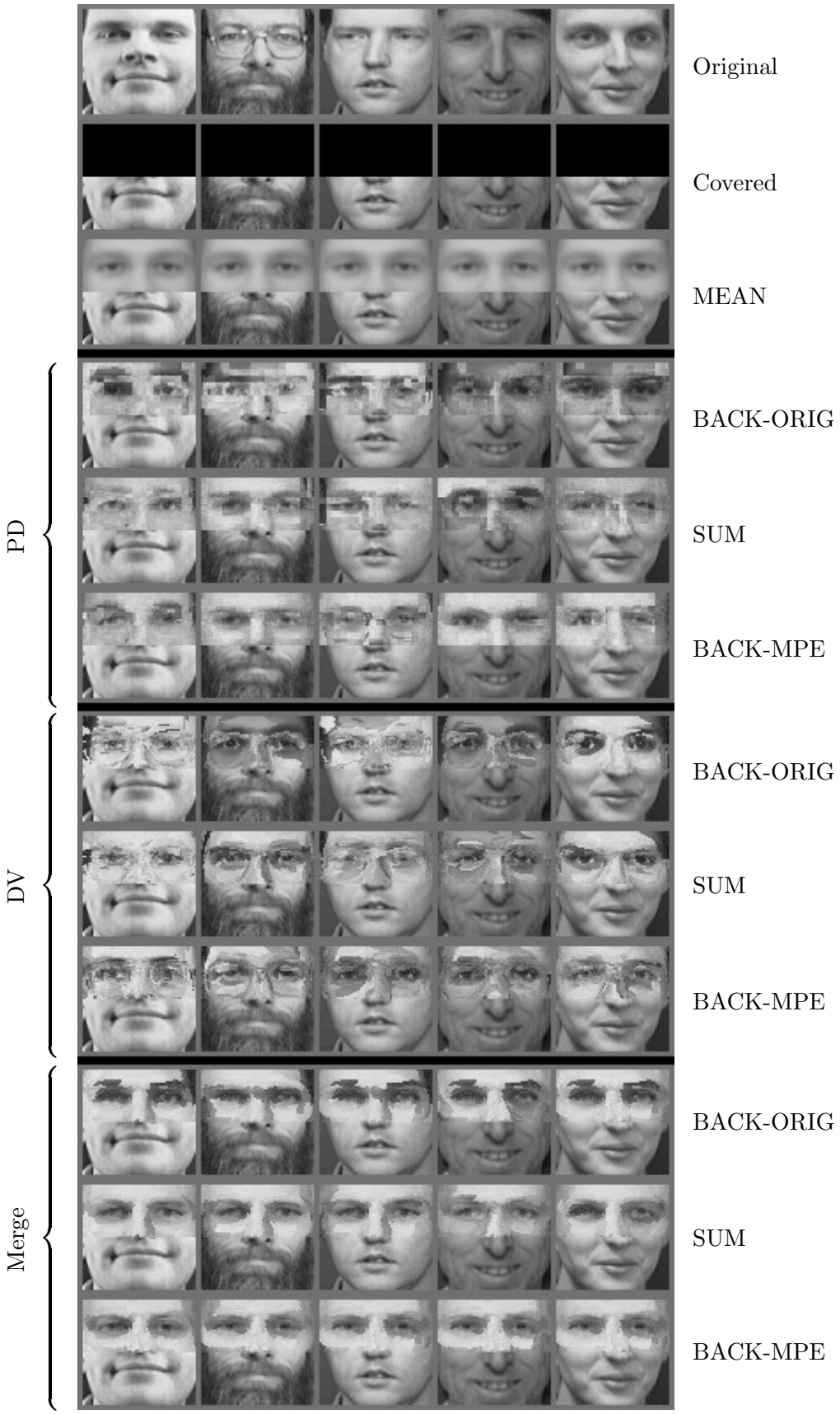


Figure 7.4: Examples of face image reconstructions, top half is covered.

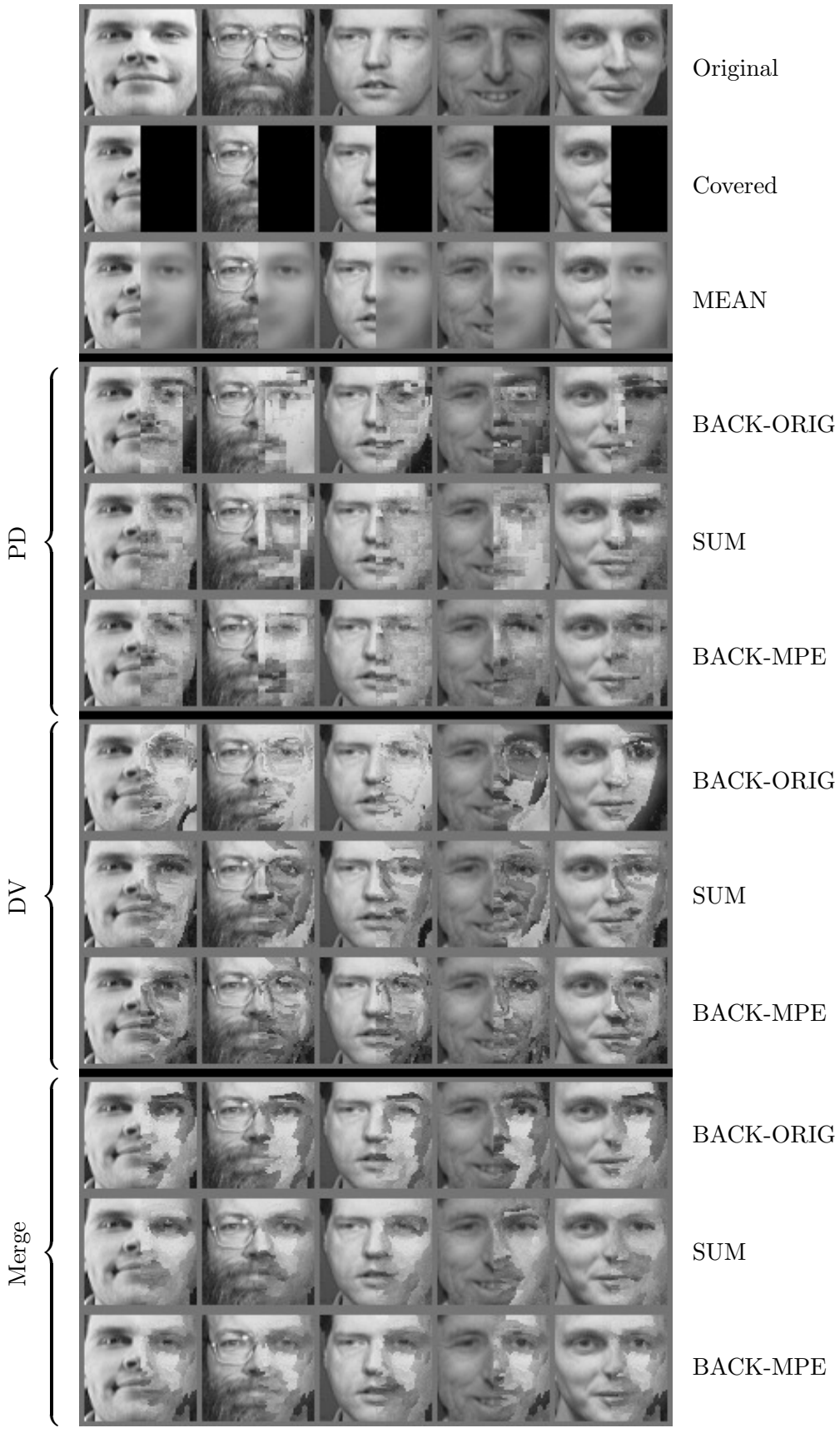


Figure 7.5: Examples of face image reconstructions, right half is covered.

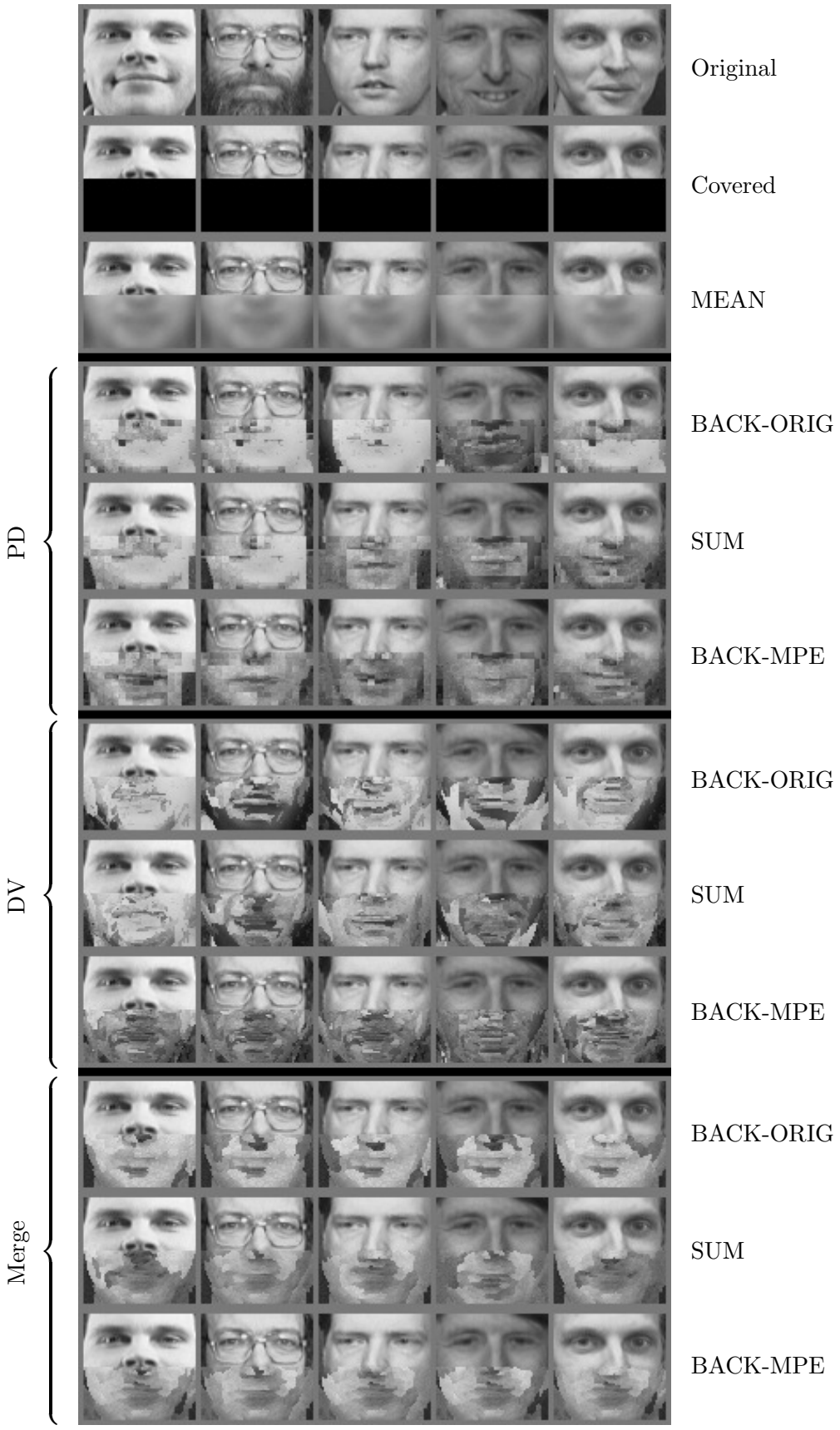


Figure 7.6: Examples of face image reconstructions, bottom half is covered.

Table 7.5: Reconstruction SNRs for the top, bottom, left, and right halves of face images covered.

		top	bottom	left	right
MEAN	–	13.16	11.25	12.72	11.83
PD	BACK-ORIG	10.45	9.29	9.83	10.49
	SUM	12.34	10.19	11.58	11.72
	BACK-MPE	12.61	10.20	11.56	11.77
DV	BACK-ORIG	11.33	8.36	9.36	10.47
	SUM	11.75	9.42	10.62	11.08
	BACK-MPE	11.33	9.58	11.29	10.70
Merge	BACK-ORIG	10.64	8.88	9.40	9.83
	SUM	12.43	9.83	10.96	11.78
	BACK-MPE	12.10	9.73	10.46	11.32

7.4 Artificial Bandwidth Extension

We can transfer their reconstruction abilities of SPNs to the audio domain, by considering an image-like representation of audio signal, e.g. the log-spectrogram. One problem which can tackled within this framework, is *artificial bandwidth extension* (ABE), i.e. to recover high frequencies which are lost in telephone transmission. In [73], the HMM-based framework for ABE [47, 90] was modified to incorporate SPNs for modeling the observations.

In the HMM-based system [47], time signals are processed in frames with some overlap, yielding a total number of T frames. For each frame, the spectral envelope of the high-band is modeled using cepstral coefficients obtained from linear prediction (LP). On a training set, these coefficients are clustered using the LBG algorithm [61]. The temporally ordered cluster indices are used as hidden state sequence of an HMM, whose prior and transition probabilities can be estimated using the observed frequency estimates. For each hidden state, an observation GMM is trained on features taken from the low-band (see [47] for details about these features). In the test phase, the high frequency components and therefore the hidden states of the HMM are missing. For each time frame, the marginal probability of the hidden state is inferred using the forward-backward algorithm [80]. For real-time capable systems, the backward-messages have to be obtained from a limited number of $\Lambda \geq 0$ look-ahead frames. Using the hidden state posterior, an MMSE estimate of the high-band cepstral coefficients is obtained [47], which together with the periodogram of the low-band yield estimates of the wide-band cepstral coefficients. To extend the excitation signal to the high-band, the low-band excitation is modulated either with a fixed frequency carrier, or with a pitch-dependent carrier. According to [47] and related ABE literature, the results are quite insensitive to the method of extending the excitation.

To incorporate SPNs in HMM-based ABE, we use the log-spectra of the time frames as observations, where redundant frequency bins are discarded. Let $S(t, f)$ be the f^{th} frequency bin of the t^{th} time-frame of the full-band signal, $t \in \{1, \ldots, T\}$, $f \in \{1, \ldots, F\}$, where F is the number of frequency bins and $\mathbf{S}_t = (S(t, 1), \ldots, S(t, F))^T$. We cluster the log-spectra $\{\mathbf{S}_{1:T}\}$ of training speech using the LBG algorithm, and use the cluster indices as hidden states of an HMM. On each cluster, we train an SPN, yielding state-dependent models over the log-spectra. For training SPNs, we use the PD algorithm [79], requiring that the data is organized as rectangular array; here the data is a $1 \times F$ rectangular array. We used $\rho = 20$ sum nodes per rectangle and $\gamma = 20$ Gaussian PDF nodes per variable. This values were chosen as an "educated guess" and *not cross-validated*. Similar as in [79], we use a *coarse resolution* of 4, i.e. rectangles of height larger than 4 are split with a stepsize of 4.

For ABE we simulate narrow-band telephone speech [2] by applying a bandpass filter with stop frequencies 50 Hz and 4000 Hz. Let $\bar{S}(t, f)$ be the time-frequency bins of the telephone filtered signal, and $\bar{\mathbf{S}}_t = (\bar{S}(t, 1), \ldots, \bar{S}(t, F))^T$. Within the telephone band, we can assume that

$S(t,f) \approx \bar{S}(t,f)$, while some of the lowest and the upper half of the frequency bins in $\bar{\mathbf{S}}_t$ are lost. To perform inference in the HMM, we marginalize missing data in the state-dependent SPNs, which can be done efficiently in SPNs, i.e. Gaussian distributions corresponding to unobserved frequency bins, constantly return value 1, cf. Section 4.4. The output probabilities serve as observation likelihoods and are processed by the forward-backward algorithm [80]. This delivers the marginals $p(Y_t \,|\, \mathbf{e}_t)$, where Y_t is the hidden HMM variable in the the t^{th} time frame, and $\mathbf{e}_t$ is denotes the observed data up to time frame t, i.e. all frequency bins in the telephone band, for all time frames $1, \ldots, (t + \Lambda)$. An illustration of the modified HMM used in this paper is given in Figure 7.7, cf. also Figure 6.5.

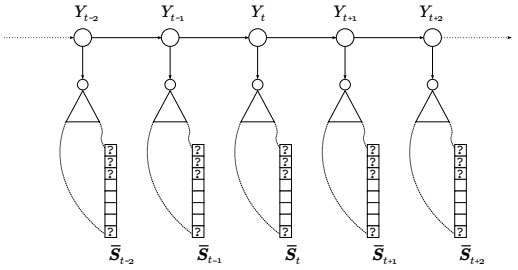


Figure 7.7: Illustration of the HMM with SPN observation models. State-dependent SPNs are symbolized by triangles with a circle on top. For the forward-backward algorithm, frequency bins marked with "?" (missing) are marginalized out by the SPNs.

We use the sum-approximation for MPE inference for recovering the missing spectrogram content, where we reconstruct the high-band only. Let $\hat{\mathbf{S}}_{t,k} = (\hat{S}_{t,k}(1), \ldots \hat{S}_{t,k}(F))^T$ be the MPE-reconstruction of the t^{th} time frame, using the SPN depending on the k^{th} HMM-state. Then we use the following bandwidth-extended log-spectrogram

$$\hat{S}(t,f) = \begin{cases} \bar{S}(t,f) & \text{if } f < f' \\ \sum_{k=1}^{K} p(Y_t = k|\mathbf{e}_t)\hat{S}_{t,k}(f) & \text{o.w.} \end{cases} \tag{7.1}$$

where f' corresponds to 4000 Hz.

7.4.1 Reconstructing Time Signals

To synthesize a time-signal from the bandwidth extended log-spectrogram, we need to associate a phase spectrum to the estimated magnitude spectrum $e^{\hat{S}(t,f)}$. The problem of recovering a time-domain signal given a modified magnitude appears in many speech applications, such as single-channel speech enhancement [60], single-channel source separation [74, 92] and speech signal modification [59,93]. These signal modifications are solely employed in spectral amplitude domain while the phase information of the desired signal is not available. A typical approach is to use the observed (noisy) phase spectrum or to replace it with an enhanced/estimated phase.

In order to recover phase information for ABE, we use the iterative algorithm proposed by Griffin and Lim (GL) [42]. Let $j \in \{0, \ldots, J\}$ be an iteration index, and $\hat{C}^{(j)}$ be a complex valued matrix generated in the j^{th} iteration. Let $|\hat{C}^{(0)}(t,f)| = e^{\hat{S}(t,f)}$ and

$$\angle\hat{C}^{(0)}(t,f) = \begin{cases} \angle\bar{C}(f,t) & 1 \le f \le f' \\ 0 & \text{o.w.,} \end{cases} \tag{7.2}$$

where $\bar{C}$ is the complex spectrogram of the bandpass filtered input signal. Within the telephone band, phase information is considered reliable and copied from the input. Outside the narrow-band phase is initialized with zero. Note that in general $\hat{C}^{(0)}$ is *not a valid* spectrogram since a time signal whose STFT equals $\hat{C}^{(0)}$ might not exist. The j^{th} iteration of the GL algorithm is

given by

$$\hat{C}^{(j)} = |\hat{C}^{(0)}| \otimes e^{i\angle\mathcal{G}(\hat{C}^{(j-1)})}, \tag{7.3}$$

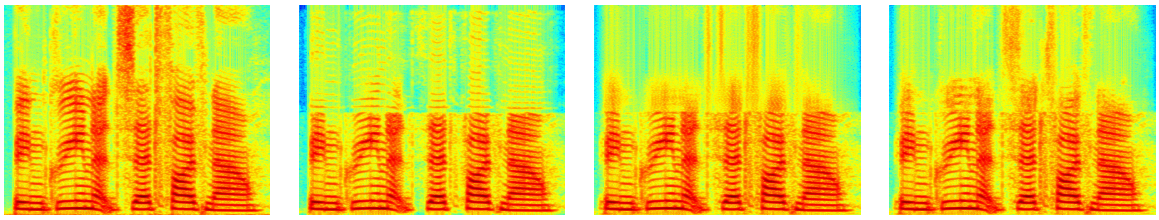
$$\mathcal{G}(C) = \mathrm{STFT}(\mathrm{STFT}^{-1}(C)), \tag{7.4}$$

where $\otimes$ denotes the Hadamard product. At each iteration, the magnitude of the approximate STFT $\hat{C}^{(j)}$ is set to the magnitude $|\hat{C}^{(0)}|$ estimated by the SPN while temporal coherence of the signal is enforced by the operator $\mathcal{G}(\cdot)$ (see e.g. [59] for more details). The estimated time signal s_j at the j^{th} iteration is given by $s_j = \mathrm{STFT}^{-1}\left(\hat{C}^{(j)}\right)$. At each iteration, the mean square error between $|\mathrm{STFT}(s_j)|$ and $|\hat{C}^{(0)}|$ is reduced [42]. In our experiments, we set the number of iterations $J = 100$, which appeared to be sufficient for convergence.

7.4.2 Experiments

We used 2 baselines in our experiments. The first baseline is the method proposed in [47], based on the vocal tract filter model using linear prediction. We used 64 HMM states and 16 components per state-dependent GMM, which performed best in [47]. We refer as HMM-LP to this baseline. The second baseline is almost identical to our method, where we replaced the SPN with a Gaussian mixture model with 256 components with diagonal covariance matrices. For training GMMs, we ran the EM algorithm for maximal 100 iterations and using 3 random restarts. Since a GMM can be represented as an SPN with a single sum node, see Figure 6.8(b), inference using the GMM model works the same way as for SPNs. We refer as HMM-GMM to this baseline. To the HMM incorporating SPNs as observation models, we refer as HMM-SPN. For HMM-GMM and HMM-SPN, we used the same clustering of log-spectra using a codebook size of 64.

We used time-frames of 512 samples length, with 75% overlap, which using a sampling frequency of 16 kHz corresponds to a frame length of 32 ms and a frame rate of 8 ms. Before applying the FFT, the frames were weighted with a Hamming window. For the forward-backward algorithm we used a look-ahead of $\Lambda = 3$ frames, which corresponds to the minimal delay introduced by the 75% frame-overlap. We performed our experiments on the GRID corpus [21], where we used the test speakers with numbers 1, 2, 18, and 20, referred to as s1, s2, s18, and s20, respectively. Speakers s1 and s2 are male, and s18 and s20 are female. We trained *speaker dependent* and *speaker independent* models. For speaker dependent models we used 10 minutes of speech of the respective speaker. For speaker independent models we used 10 minutes of speech obtained from the remaining 30 speakers of the corpus, each speaker providing approximately 20 seconds of speech. For testing we used 50 utterances per test speaker, not included in the training set.



(a) Original (b) HMM-LP (c) HMM-GMM (d) HMM-SPN

Figure 7.8: Log-spectrogram of the utterance "Bin green at zed 5 now", spoken by s18. *(a): original full bandwidth signal. (b): ABE result of HMM-LP [47]. (c): ABE result of HMM-GMM. (d): ABE results of HMM-SPN.*

Figure 7.8 shows log-spectrograms of a test utterance of speaker s18 and the bandwidth extended signals by HMM-LP, HMM-GMM and HMM-SPN, using speaker dependent models. We

see that HMM-LP succeeds in reconstructing a harmonic structure for voiced sounds. However, we see that fricative and plosive sounds are not well captured. The reconstruction by HMM-GMM is blurry and does not recover the harmonic structure of the original signal well, but partly recovers high-frequency content related to consonants. The HMM-SPN method recovers a natural high frequency structure, which largely resembles the original full-band signal: the harmonic structure appears more natural than the one delivered by HMM-LP and consonant sounds seem to be better detected and reconstructed than by HMM-GMM.

For an objective evaluation, we use the log-spectral distortion (LSD) in the high-band [47]. Given an original signal and an ABE reconstruction, we perform L^{th}-order LPC analysis for each frame, where $L = 9$. This yields $(L+1)$-dimensional coefficient vectors $\mathbf{a}_t$ and $\hat{\mathbf{a}}_t$ of the original and the reconstructed signals, respectively. The spectral envelope modeled by a generic LPC coefficient vector $\mathbf{a} = (a_0, \ldots, a_L)^t$ is given as

$$E_{\mathbf{a}}(e^{j\Omega}) = \frac{\sigma}{|\sum_{k=0}^{L} a_k e^{-jk\Omega}|}, \tag{7.5}$$

where σ is the square-root of the variance of the LPC-analyzed signal. The LSD for the τ^{th} frame, in high-band is calculated as

$$\text{LSD}_\tau = \sqrt{\frac{\int_\nu^\pi \left(20 \log E_{\mathbf{a}_\tau}(e^{j\Omega}) - 20 \log E_{\hat{\mathbf{a}}_\tau}(e^{j\Omega})\right)^2 d\Omega}{\pi - \nu}}, \tag{7.6}$$

where $\nu = \pi \frac{4000}{fs/2}$, fs being the sampling frequency. The LSD at utterance level is given as the average of LSD_τ over all frames. Tables 7.6 and 7.7 show the LSD of all three methods for the speaker dependent and speaker independent scenarios, respectively, averaged over the 50 test sentences. We see a clear ranking of the three methods, and that the HMM-SPN method always performs best. All differences are significant at a 0.95 confidence level, according to a paired one-sided t-test.

Table 7.6: Average LSD using speaker-dependent models.

	s1	s2	s18	s20
HMM-LP	7.13	7.57	6.48	6.41
HMM-GMM	3.18	2.93	2.28	2.82
HMM-SPN	**3.12**	**2.84**	**2.15**	**2.59**

Table 7.7: Average LSD using speaker-independent models.

	s1	s2	s18	s20
HMM-LP	7.12	7.66	6.60	6.34
HMM-GMM	3.62	4.46	3.82	3.60
HMM-SPN	**3.42**	**3.85**	**3.05**	**3.36**

8 Discussion and Future Work

The core part of this thesis is dedicated to investigate the foundations of SPNs and to strengthen our theoretic understanding about them. We want to summarize the most important insights and draw a conceptional picture of SPNs, yielding from the results developed in this thesis.

In the most basic definition, SPNs are feedforward neural networks containing sum and product neurons. We define a distribution by simply normalizing the output of the SPN. Although this approach is mathematically sound, we will not be able to perform inference in such a model, except in the simplest cases. This calls for considering a restricted class of SPNs, which allows at least to compute the normalization constant of the SPN. In [79], this class was called *valid SPNs*, and two sufficient conditions were given to guarantee validity: *completeness* and *consistency*. At the same time, an alternative condition to consistency, *decomposability*, was given; decomposability implies consistency, but not vice versa, i.e. consistency is the weaker condition. However, in literature mostly decomposability is used, since decomposability is conceptually easier and also easier to establish. This raises the question: Do we loose much when using decomposability? The answer we give here is no, since consistent SPNs can not model distributions significantly more concise than decomposable SPNs (Theorem 4.3). Furthermore, consistency rules out a class of inference scenarios defined via the *differential approach* [27], which *can* be applied in decomposable SPNs. These arguments give us license to consider only complete and decomposable SPNs, simplifying our conceptual image of SPNs.

SPNs with locally normalized weights are readily normalized, and in literature often locally normalized SPNs were used due to convenience. Again, this raises the question: Do we loose much by this restriction? Here the answer is an even clearer no; we can without loss of generality assume locally normalized SPN weights (Theorem 4.2), which again simplifies our conceptual image of SPNs.

Complete and decomposable SPNs can be easily generalized be replacing IVs by more general input distributions, also over larger scopes. It is immediate that generalized SPNs represent some distribution and it is clear how to evaluate this distribution for some sample. However, it was missed in literature to carry over other inference mechanism from finite-state SPNs, i.e. marginalization and the differential approach. In this thesis we derived these inference scenarios for generalized SPNs, cf. Section 4.4. When we neglect consistency, we see that finite-state SPNs are indeed just a special case of generalized SPNs. Furthermore, since we can always assume locally normalized weights, we see that SPNs can be understood as nothing but a nested arrangement of *mixtures of distributions* (sum nodes) and *factorized distributions* (product nodes). Although this description of SPNs sounds simple, one must not underestimate their model power. We saw in Section 6.4, that even a quite restricted class of SPNs, i.e. regular

selective SPNs, generalizes BNs over finite-state RVs and incorporates advanced concepts such as context-specific independence, nested multi-nets and sub-component sharing, *without need for any specialized inference mechanism.*

A central theme of SPNs is their interpretation as *hierarchical structured latent RV* model. As pointed out in Chapter 5, the original augmentation of SPNs explicitly incorporating these latent RVs is too simplistic, leading to irritating results. We completed this augmentation, by introducing the so-called *twin sum nodes*. Furthermore, we introduced an IMAP of the augmented model, represented as BN, linking SPNs back to classical PGMs. The BN interpretation of augmented SPNs helped us to derive a sound EM-algorithm for SPN weights.

The latent RV interpretation was already used in the seminal paper [79] for data reconstruction using MPE inference. The used algorithm for MPE inference was never shown to be correct, besides the fact that the latent model was not completely specified. We showed that it is indeed a correct algorithm for *augmented* SPNs, but that it needs a modification when applied to non-augmented SPNs.

Furthermore, so far it was not clear if the more desirable *MAP inference* for data reconstruction can be performed efficiently in SPNs. We gave a negative result, showing that this problem is generally NP-hard.

This thesis tries to shed some light on the foundations of SPNs. However, it is certainly not an exhaustive or completely rigorous treatment of the subject of SPNs. Many questions are open and we want to conclude the thesis with possible future research topics.

- What is the complexity of MAP inference in SPNs?
- Is there a class of SPNs for which MAP inference can efficiently be solved, i.e. its decision version is in P?
- If yes, how does it look like?
- Design an algorithm which returns a MAP solution given enough time and memory and which delivers a solution with sub-optimality bounds, i.e. an any-time solution. A possible technique could be branch-and-bound.
- What are suitable cost functions for learning SPNs?
- Given such cost function, is learning SPNs NP-hard? (Presumably yes)
- Given a budget of arithmetic operations and a cost function, design an algorithm returning a globally optimal SPN.

A Appendix

A.1 Proofs for Chapter 4

Proposition 4.1. *Let* P *be a product node and* $\mathbf{Y}$ *be its shared RVs.* P *is consistent if and only if for each* $Y \in \mathbf{Y}$ *there exists a* unique $y^* \in \mathbf{val}(Y)$ *with* $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{P})$.

Proof of Proposition (4.1).

Direction "$\Leftarrow$", i.e. suppose for each $Y \in \mathbf{Y}$ there exists a unique $y^* \in \mathbf{val}(Y)$ with $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{P})$. Consider arbitrary two distinct children $\mathsf{C}', \mathsf{C}'' \in \mathbf{ch}(\mathsf{P})$ and let $\mathbf{Y}' = \mathbf{sc}(\mathsf{C}') \cap \mathbf{sc}(\mathsf{C}'')$, where from Definition 4.12 (shared RVs) it follows that $\mathbf{Y}' \subseteq \mathbf{Y}$. First, let $X \in \mathbf{sc}(\mathsf{C}') \setminus \mathbf{Y}'$ arbitrary. Since $X \notin \mathbf{sc}(\mathsf{C}'')$, we have

$$\lambda_{X=x} \in \mathbf{desc}(\mathsf{C}') \Rightarrow \forall x' \colon \lambda_{X=x'} \notin \mathbf{desc}(\mathsf{C}''), \tag{A.1}$$

and in particular

$$\lambda_{X=x} \in \mathbf{desc}(\mathsf{C}') \Rightarrow \forall x' \neq x \colon \lambda_{X=x'} \notin \mathbf{desc}(\mathsf{C}''). \tag{A.2}$$

Now let $X \in \mathbf{Y}'$ arbitrary. Since $X \in \mathbf{Y}$, there is a unique $x^* \in \mathbf{val}(X)$ with $\lambda_{X=x^*} \in \mathbf{desc}(\mathsf{C}')$ and $\lambda_{X=x^*} \in \mathbf{desc}(\mathsf{C}'')$, and again (A.2) holds. Since C', C'' and X are arbitrary, P is consistent.

Direction "$\Rightarrow$", i.e. suppose P is consistent. If $\mathbf{Y}$ is empty, i.e. P is decomposable, then the claimed property holds trivially for all $\mathbf{Y}$. Otherwise let $Y \in \mathbf{Y}$ arbitrary and let C' and C'' be arbitrary two children having Y in their scope. There are at least two such children by Definition 4.12 (shared RVs). There must be a unique y^* with $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{C}')$ and $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{C}'')$. To see this, note that there must be at least one $y' \in \mathbf{val}(Y)$ with $\lambda_{Y=y'} \in \mathbf{desc}(\mathsf{C}')$ and at least one $y'' \in \mathbf{val}(Y)$ with $\lambda_{Y=y''} \in \mathbf{desc}(\mathsf{C}'')$, since Y is in the scope of both C' and C''. However, there can be at most one such y' and y''. Assume there were y'_1 and y'_2, $y'_1 \neq y'_2$. Then either for $y' = y'_1$ or for $y' = y'_2$ we had a contradiction to

$$\lambda_{Y=y'} \in \mathbf{desc}(\mathsf{C}') \Rightarrow \forall y'' \neq y' \colon \lambda_{Y=y'} \notin \mathbf{desc}(\mathsf{C}''). \tag{A.3}$$

i.e. P would not be consistent. By symmetry, there is also at most one y'', and (A.3) can only hold when $y' = y'' =: y^*$. Therefore, since Y, C' and C'' are arbitrary, there must be a unique y^* with $y^* \in \mathbf{desc}(\mathsf{C})$, $\forall \mathsf{C} \in \mathbf{ch}(\mathsf{P}) \colon Y \in \mathbf{sc}(\mathsf{C})$. Therefore, this y^* is also the unique value with $\lambda_{Y=y^*} \in \mathbf{desc}(\mathsf{P})$. □

Lemma 4.1. *Let* N *be a node in some complete and consistent SPN,* $X \in \mathbf{sc}(\mathsf{N})$ *and* $x \in \mathbf{val}(X)$*. When* $\lambda_{X=x} \notin \mathbf{desc}(\mathsf{N})$*, then* $\forall \mathbf{x} \in \mathbf{val}(\mathbf{X})$ *with* $\mathbf{x}[X] = x$ *we have* $\mathsf{N}(\mathbf{x}) = 0$*.*

Proof of Lemma 4.1.

When $\mathsf{N} = \lambda_{X=x'}$, for some $x' \in \mathbf{val}(X)$, the lemma clearly holds, since $x' \neq x$.

When N is a product or a sum, let $\mathbf{N} = \{\mathsf{N}' \in \mathbf{desc}(\mathsf{N}) : X \in \mathbf{sc}(\mathsf{N}')\}$. Let $K = |\mathbf{N}|$ and let $\mathsf{N}_1, \ldots, \mathsf{N}_K$ be a topologically order list of $\mathbf{N}$, i.e. $\mathsf{N}_k \notin \mathbf{desc}(\mathsf{N}_l)$ when $k > l$. We can assume that for some I, $\mathsf{N}_1, \ldots, \mathsf{N}_I$ are IVs of X, $\mathsf{N}_{I+1}, \ldots, \mathsf{N}_K$ are sum and product nodes, where $\mathsf{N}_K = \mathsf{N}$. For any $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ with $\mathbf{x}[X] = x$, we have $\mathsf{N}_1(\mathbf{x}) = \mathsf{N}_2(\mathbf{x}) = \cdots = \mathsf{N}_I(\mathbf{x}) = 0$, since $\lambda_{X=x} \notin \mathbf{desc}(\mathsf{N})$.

When all $\mathsf{N}_1(\mathbf{x}) = \mathsf{N}_2(\mathbf{x}) = \cdots = \mathsf{N}_k(\mathbf{x}) = 0$ for some $k \geq I$, then also $\mathsf{N}_{k+1}(\mathbf{x}) = 0$. When N_{k+1} is a sum node, due to completeness, all children of N_{k+1} must have X in their scope. This means that all children are in $\{\mathsf{N}_1, \ldots, \mathsf{N}_k\}$. Therefore $\mathsf{N}_{k+1}(\mathbf{x}) = 0$. When N_{k+1} is a product, at least on child has to be in $\{\mathsf{N}_1, \ldots, \mathsf{N}_k\}$. Therefore $\mathsf{N}_{k+1}(\mathbf{x}) = 0$. Thus, by induction, for all $\mathsf{N}' \in \mathbf{N}$ we have $\mathsf{N}'(\mathbf{x}) = 0$. In particular this is true for $\mathsf{N}_K = \mathsf{N}$. □

Lemma 4.2. *Let* P *be a PMF over* $\mathbf{X}$ *and* $\mathbf{Y} \subseteq \mathbf{X}$*,* $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$ *such that there exists a* $\mathbf{y}^* \in \mathbf{val}(\mathbf{Y})$ *with* $P(\mathbf{z}, \mathbf{y}) = 0$ *when* $\mathbf{y} \neq \mathbf{y}^*$*. Then we have* $P(\mathbf{z}, \mathbf{y}) = \mathbb{1}(\mathbf{y} = \mathbf{y}^*)\, P(\mathbf{z})$*.*

Proof of Lemma 4.2.

Since $P(\mathbf{z}, \mathbf{y}) = 0$ when $\mathbf{y} \neq \mathbf{y}^*$, $P(\mathbf{z}) = \sum_{\mathbf{y} \in \mathbf{val}(\mathbf{Y})} P(\mathbf{z}, \mathbf{y}) = P(\mathbf{z}, \mathbf{y}^*)$. Thus

$$P(\mathbf{z}, \mathbf{y}) = \begin{cases} 0 & \text{if } \mathbf{y} \neq \mathbf{y}^* \\ P(\mathbf{z}) & \text{if } \mathbf{y} = \mathbf{y}^* \end{cases} = \mathbb{1}(\mathbf{y} = \mathbf{y}^*)\, P(\mathbf{z}). \tag{A.4}$$

□

Proposition 4.2. *A complete and decomposable SPN computes the network polynomial of some unnormalized distribution.*

Proof of Proposition 4.2.

Recall that a network polynomial for unnormalized distribution Φ is defined as (cf. (4.13)):

$$f_\Phi(\boldsymbol{\lambda}) = \sum_{\mathbf{x} \in \mathbf{val}(\mathbf{X})} \Phi(\mathbf{x}) \prod_{X \in \mathbf{X}} \lambda_{X=\mathbf{x}[X]}. \tag{A.5}$$

Assume some topological order of the SPN nodes. We show by induction over this order, from bottom to top, that each sub-SPN $\mathcal{S}_\mathsf{N}$ computes a network polynomial over $\mathbf{sc}(\mathsf{N})$. The basis are the IVs $\lambda_{X=x}$ which compute

$$\lambda_{X=x} = \sum_{x' \in \mathbf{val}(X)} \mathbb{1}(x' = x)\, \lambda_{X=x'}, \tag{A.6}$$

which has the form (A.5), i.e. IVs compute network polynomials.

A complete sum node S computes

$$\mathsf{S}(\boldsymbol{\lambda}) = \sum_{\mathsf{C}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \mathsf{C}(\boldsymbol{\lambda}) \tag{A.7}$$

$$= \sum_{\mathsf{C}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{sc}(\mathsf{S}))} \Phi_{\mathsf{C}}(\mathbf{x}) \prod_{X\in\mathbf{sc}(\mathsf{S})} \lambda_{X=\mathbf{x}[X]} \tag{A.8}$$

$$= \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{sc}(\mathsf{S}))} \left(\sum_{\mathsf{C}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \Phi_{\mathsf{C}}(\mathbf{x}) \right) \prod_{X\in\mathbf{sc}(\mathsf{S})} \lambda_{X=\mathbf{x}[X]} \tag{A.9}$$

$$= \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{sc}(\mathsf{S}))} \Phi_{\mathsf{S}}(\mathbf{x}) \prod_{X\in\mathbf{sc}(\mathsf{S})} \lambda_{X=\mathbf{x}[X]}, \tag{A.10}$$

where Φ_{C} is the unnormalized distributions of the network polynomial of C, and $\Phi_{\mathsf{S}}(\mathbf{x}) := \sum_{\mathsf{C}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}}\, \Phi_{\mathsf{C}}(\mathbf{x})$. In (A.8) we apply the induction hypothesis and the fact that for complete sum nodes the scopes of the node and all its children are the same. We see that (A.10) has the form (A.5), i.e. the induction step holds for complete sum nodes.

Now consider some decomposable product node P with children $\mathbf{ch}(\mathsf{P}) = \{\mathsf{C}_1, \ldots, \mathsf{C}_K\}$. The product node P computes

$$\mathsf{P}(\boldsymbol{\lambda}) = \prod_{k=1}^{K} \mathsf{C}_k(\boldsymbol{\lambda}) \tag{A.11}$$

$$= \prod_{k=1}^{K} \sum_{\mathbf{x}^k\in\mathbf{val}(\mathbf{sc}(\mathsf{C}_k))} \Phi_{\mathsf{C}_k}(\mathbf{x}^k) \prod_{X\in\mathbf{sc}(\mathsf{C}_k)} \lambda_{X=\mathbf{x}^k[X]} \tag{A.12}$$

$$= \sum_{\mathbf{x}^1\in\mathbf{val}(\mathbf{sc}(\mathsf{C}_1))} \cdots \sum_{\mathbf{x}^K\in\mathbf{val}(\mathbf{sc}(\mathsf{C}_K))} \prod_{k=1}^{K} \left[\Phi_{\mathsf{C}_k}(\mathbf{x}^k) \prod_{X\in\mathbf{sc}(\mathsf{C}_k)} \lambda_{X=\mathbf{x}^k[X]} \right] \tag{A.13}$$

$$= \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{sc}(\mathsf{P}))} \left(\prod_{k=1}^{K} \Phi_{\mathsf{C}_k}(\mathbf{x}[\mathbf{sc}(\mathsf{C}_k)]) \right) \left(\prod_{X\in\mathbf{sc}(\mathsf{P})} \lambda_{X=\mathbf{x}[X]} \right) \tag{A.14}$$

$$= \sum_{\mathbf{x}\in\mathbf{val}(\mathbf{sc}(\mathsf{P}))} \Phi_{\mathsf{P}}(\mathbf{x}) \prod_{X\in\mathbf{sc}(\mathsf{P})} \lambda_{X=\mathbf{x}[X]}, \tag{A.15}$$

where Φ_{C_k} is the distributions of the network polynomial of C_k, and $\Phi_{\mathsf{P}}(\mathbf{x}) := \prod_{k=1}^{K} \Phi_{\mathsf{C}_k}(\mathbf{x}[\mathbf{sc}(\mathsf{C}_k)])$. In (A.12) we use the induction hypothesis, in (A.13) we apply the distributive law, and in (A.14) we use the fact that the scope of a decomposable product node is partitioned by the scopes of its children. We see that (A.15) has the same form as (A.5), i.e. the induction step holds for complete product nodes.

Consequently, each sub-SPN and therefore also the overall SPN computes a network polynomial of some unnormalized distribution. □

Proposition 4.3. *Every complete and consistent, but non-decomposable SPT $\mathcal{S} = ((\mathbf{V}, \mathbf{E}), \boldsymbol{w})$ over $\mathbf{X}$ can be transformed into a complete and decomposable SPT $\mathcal{S}' = ((\mathbf{V}', \mathbf{E}'), \boldsymbol{w}')$ over $\mathbf{X}$ such that $\mathcal{S} \equiv \mathcal{S}'$, and where $|\mathbf{V}'| \leq |\mathbf{V}|$, $A_{\mathcal{S}'} \leq A_{\mathcal{S}}$ and $M_{\mathcal{S}'} < M_{\mathcal{S}}$.*

Proof. For SPTs, no links are introduced in steps 2–7 of Algorithm 2. Consider a non-decomposable product P in the SPT, and let $\mathbf{Y}$ be the shared RVs. If $\mathbf{sc}(\mathsf{P}) = \mathbf{Y}$, the whole sub-SPN rooted at P will be replaced by $\prod_{Y\in\mathbf{Y}} \lambda_{Y=\mathbf{y}^*[Y]}$ in step 13. The deleted sub-SPN computes $\prod_{Y\in\mathbf{Y}} (\lambda_{Y=\mathbf{y}^*[Y]})^{k_Y}$, where k_Y are integers with $k_Y \geq 1$, and at least one $k_Y > 1$, since otherwise P would be decomposable. Thus, replacing this sub-SPN by the decomposable product in step 13 saves at least one multiplication.

Similarly, when $\mathbf{sc}(\mathsf{P}) \neq \mathbf{Y}$, steps 21–23 prune nodes from the sub-SPN rooted at P, corresponding to the computation of $\prod_{Y \in \mathbf{Y}} (\lambda_{Y=\mathbf{y}^*[Y]})^{k_Y}$ with $k_Y \geq 1$, and at least one $k_Y > 1$. This requires at least one multiplication more than directly connecting $\lambda_{Y=\mathbf{y}^*[Y]}$ to P in step 24. Clearly, steps 17–20 are never performed in SPTs, since no $\mathsf{N} \in \mathbf{desc}(\mathsf{P}) \setminus \{\mathsf{P}\}$ can have a parent outside the descendants of P.

Thus, for every non-decomposable product, Algorithm 2 saves at least one multiplication. □

Proposition 4.5. *Let $\mathcal{S}$ be an SPN and $\mathcal{S}^g$ be its gated SPN. Then*

1. *$\mathcal{S}^g$ is an SPN over $\mathbf{X} \cup \mathbf{Z}$.*
2. *$\mathcal{S}(\mathbf{X}) = \mathcal{S}^g(\mathbf{X}) = \sum_{\mathbf{z} \in \mathbf{val}(\mathbf{Z})} \mathcal{S}^g(\mathbf{X}, \mathbf{z})$.*
3. *For any $X \in \mathbf{X}$ and $k \in \{1, \ldots, |\mathbf{D}^X|\}$, let $\mathbf{X}^D = \mathbf{sc}(\mathsf{D}^{X,k})$ and $\mathbf{X}^R = \mathbf{X} \setminus \mathbf{X}^D$. It holds that*

$$\mathcal{S}^g(\mathbf{X}, Z_X = k, \mathbf{Z} \setminus \{Z_X\}) = \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathcal{S}^g(\mathbf{X}^R, Z_X = k, \mathbf{Z} \setminus \{Z_X\}), \tag{A.16}$$

i.e.

$$\mathbf{X}^D \perp\!\!\!\perp \mathbf{X}^R \cup \mathbf{Z} \setminus \{Z_X\} \,|\, Z_X \tag{A.17}$$

In particular

$$\mathcal{S}^g(\mathbf{X}, Z_X = k) = \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathcal{S}^g(\mathbf{X}^R, Z_X = k), \tag{A.18}$$

Proof of Proposition 4.5.

ad 1) In the gated SPN, each distribution node D is replaced by a product node with scope $\mathbf{sc}(\mathsf{D}) \cup \{Z_X \mid X \in \mathbf{sc}(\mathsf{D})\}$. That is, in the scope of each ancestor of D, an RV X is replaced by a compound RV $\{X, Z_X\}$. This leaves completeness and decomposability intact. The products introduced in (4.65) are also clearly decomposable and the IVs are input distributions over the DSs. Therefore $\mathcal{S}^g$ fulfills Definition 4.15, i.e. it is an generalized SPN, and the scope of the root is $\mathbf{X} \cup \mathbf{Z}$.

ad 2) Since $\mathcal{S}^g$ is a generalized SPN over $\mathbf{X} \cup \mathbf{Z}$, marginalization over $\mathbf{Z}$ is performed by setting all $\lambda_{Z_X=k} = 1$. In this case, the introduced product nodes simply copy the corresponding input distribution, i.e. all remaining nodes in $\mathcal{S}^g$ perform the same computations as in $\mathcal{S}$, so their outputs agree.

ad 3) Consider all internal nodes $\mathbf{N}$ in $\mathcal{S}^g$ which have X in their scope. Since they have X in their scope, they also have Z_X in their scope. Consider a fixed k and assume $Z_X = k$. We split $\mathbf{N}$ into two sets $\mathbf{N}'$ and $\mathbf{N}''$, where $\forall \mathsf{N} \in \mathbf{N}' \colon \mathsf{D}^{X,k} \in \mathbf{desc}(\mathsf{N})$ and $\forall \mathsf{N} \in \mathbf{N}'' \colon \mathsf{D}^{X,k} \notin \mathbf{desc}(\mathsf{N})$.

We first show by induction that all $\mathsf{N} \in \mathbf{N}''$ output 0. Let $\mathsf{N}_1, \ldots, \mathsf{N}_K$ be a topological ordered list of nodes in $\mathbf{N}''$, i.e. $k > l \Rightarrow \mathsf{N}_k \notin \mathbf{desc}(\mathsf{N}_l)$. We assume w.l.o.g. that $\mathsf{N}_1, \ldots, \mathsf{N}_{|\mathbf{D}^X|-1}$, are the gates $\mathsf{P}^{X,l}$, $l \neq k$. These nodes output 0, since $\lambda_{Z_X=l} = 0$ for $Z_X = k$. This is the induction basis. For the induction step, assume that $\mathsf{N}_1, \ldots, \mathsf{N}_M$ output 0, for $|\mathbf{D}^X| - 1 \leq M < K$. We show that then also N_{M+1} outputs 0. If N_{M+1} is a sum S, all its children must also be in $\mathbf{N}''$: they must be in $\mathbf{N}$, i.e. have X in their scope, since S has X in its scope. They also can not be $\mathbf{N}'$, since otherwise S would be in $\mathbf{N}'$. Thus they are in $\mathbf{N}''$, and furthermore they must be in $\{\mathsf{N}_1, \ldots, \mathsf{N}_M\}$, due to the topological ordering. Since they output 0, also S outputs 0. Similarly, when N_{M+1} is a product P, it must have exactly one child with X in its scope, which must be in $\{\mathsf{N}_1, \ldots, \mathsf{N}_M\}$. Since this child outputs 0 also P outputs 0. Therefore all $\mathbf{N}''$ output 0.

We now show by induction that all $\mathsf{N} \in \mathbf{N}'$ have the form

$$\mathsf{D}^{X,k}(\mathbf{X}^D)\,\mathsf{N}(\mathbf{X}^{R'}, Z_X = k, \mathbf{Z}' \setminus \{Z_X\}), \tag{A.19}$$

where $\mathbf{X}^{R'} = \mathbf{X}^R \cap \mathbf{sc}(\mathsf{N})$ and $\mathbf{Z}' = \mathbf{Z} \cap \mathbf{sc}(\mathsf{N})$. Let $\mathsf{N}_1, \ldots, \mathsf{N}_K$ be a topological ordered list of nodes in $\mathbf{N}'$, i.e. $k > l \Rightarrow \mathsf{N}_k \notin \mathbf{desc}(\mathsf{N}_l)$. N_1 must be the gate $\mathsf{P}^{X,k}$ which has the form $\mathsf{D}^{X,k} \prod_{Y \in \mathbf{sc}(\mathsf{D}^{X,k})} \lambda_{Z_Y = [\mathsf{D}^{X,k}]^Y}$, which has the form (A.19) with $\mathbf{X}^{R'} = \emptyset$. This is the induction basis. For the induction step, assume that all $\mathsf{N}_1, \ldots, \mathsf{N}_M$ have the form (A.19), for $M \geq 1$. We show that then also N_{M+1} has the form (A.19). When N_{M+1} is a sum node S, all its children must be contained in $\mathbf{N}$ due to completeness and since X is in $\mathbf{sc}(\mathsf{S})$. Partition $\mathbf{ch}(\mathsf{S})$ into $\mathbf{C}' = \mathbf{ch}(\mathsf{S}) \cap \mathbf{N}'$ and $\mathbf{C}'' = \mathbf{ch}(\mathsf{S}) \cap \mathbf{N}''$. $\mathbf{C}'$ must contain at least one node, since S is contained in $\mathbf{N}'$. Furthermore, $\mathbf{C}'$ must be a subset of $\{\mathsf{N}_1, \ldots, \mathsf{N}_M\}$, due to the topological ordering. S computes

$$\mathsf{S}(\mathbf{X}, Z_X = k, \mathbf{Z}') = \sum_{\mathsf{C} \in \mathbf{C}'} w_{\mathsf{S},\mathsf{C}}\, \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathsf{C}(\mathbf{X}^{R'}, Z_X = k, \mathbf{Z}' \setminus \{Z_X\}) + \sum_{\mathsf{C} \in \mathbf{C}''} w_{\mathsf{S},\mathsf{C}}\, 0 \tag{A.20}$$

$$= \mathsf{D}^{X,k}(\mathbf{X}^D) \sum_{\mathsf{C} \in \mathbf{C}'} w_{\mathsf{S},\mathsf{C}}\, \mathsf{C}(\mathbf{X}^{R'}, Z_X = k, \mathbf{Z}'\ \{Z_X\}), \tag{A.21}$$

which has the form (A.19). When N_{M+1} is a product node P it must have exactly one child C in $\mathbf{N}'$, due to decomposability and since P is in $\mathbf{N}'$. This C also must be in $\{\mathsf{N}_1, \ldots, \mathsf{N}_M\}$, due to the topological ordering. The product node computes

$$\mathsf{P}(\mathbf{X}, Z_X = k, \mathbf{Z}') = \mathsf{D}^{X,k}(\mathbf{X}^D)\, \mathsf{C}(\cdot, Z_X = k, \mathbf{Z}' \cap \mathbf{sc}(\mathsf{C}) \setminus \{Z_X\}) \prod_{\mathsf{C}' \in \mathbf{ch}(\mathsf{P}) \setminus \{\mathsf{C}\}} \mathsf{C}'(\cdot), \tag{A.22}$$

which has the form (A.19). Therefore all $\mathbf{N}'$ have this form. Since the root is contained in $\mathbf{N}'$, the claimed property (4.66) follows. Eq. (4.68) follows by marginalizing $\mathbf{Z} \setminus \{Z_X\}$.

□

A.2 Proofs for Chapter 5

Proposition 5.1. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$ and $\mathbf{Z} := \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$. Then $\mathcal{S}'$ is a complete and decomposable SPN over $\mathbf{X} \cup \mathbf{Z}$ with $\mathcal{S}'(\mathbf{X}) \equiv \mathcal{S}(\mathbf{X})$.*

Proof of Proposition 5.1. We restate Algorithm 3 in Algorithm 8. If $\mathcal{S}'$ is a complete and decomposable SPN over $\mathbf{X} \cup \mathbf{Z}$, then $\mathcal{S}'(\mathbf{X}) \equiv \mathcal{S}(\mathbf{X})$ is immediate: In this case, computing $\mathcal{S}'(\mathbf{x})$ for any $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ is done by marginalizing $\mathbf{Z}$, i.e. setting all $\lambda_{Z_\mathsf{S}=k} = 1$. Introducing the links clearly does not change the function computed by the SPN. Also connecting $\lambda_{Z_\mathsf{S}=k}$ to P^k_S does not change the function, when $\lambda_{Z_\mathsf{S}=k}$ is set constantly to 1. The twin sum $\bar{\mathsf{S}}$ also constantly outputs 1, and therefore does not change the computed function when connected to the links of its conditioning sums $\mathbf{S}^c$. Therefore, the outputs of $\mathcal{S}$ and $\mathcal{S}'$ agree for each $\mathbf{x} \in \mathbf{val}(\mathbf{X})$ when $\mathbf{Z}$ is marginalized in $\mathcal{S}'$, i.e. $\mathcal{S}'(\mathbf{X}) \equiv \mathcal{S}(\mathbf{X})$.

It remains to show that $\mathcal{S}'$ is complete, decomposable and totally normalized, and that its root has $\mathbf{X} \cup \mathbf{Z}$ in its scope. After the first part of Algorithm 3 the SPN is clearly complete and decomposable, but does not have any Z_S in its scope. We show that after each execution of the for-loop body ($10 - 21$), the SPN remains complete, decomposable and contains Z_S in its root's scope. After step 12, the links P^k_S are clearly decomposable, because no $\lambda_{Z_\mathsf{S}=k}$ has

Algorithm 8 Augment SPN $\mathcal{S}$

$\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}),\ \forall k \in \{1, \dots, K_{\mathsf{S}}\}$ let $w_{\mathsf{S},k} = w_{\mathsf{S},\mathsf{C}^k_{\mathsf{S}}}$, $\bar{w}_{\mathsf{S},k} = \bar{w}_{\mathsf{S},\mathsf{C}^k_{\mathsf{S}}}$

Introduce links:

1: **for** $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
2: **for** $k = 1 \dots K_{\mathsf{S}}$ **do**
3: Introduce a new product node $\mathsf{P}^k_{\mathsf{S}}$ in $\mathcal{S}$
4: Disconnect $\mathsf{C}^k_{\mathsf{S}}$ from S
5: Connect $\mathsf{C}^k_{\mathsf{S}}$ as child of $\mathsf{P}^k_{\mathsf{S}}$
6: Connect $\mathsf{P}^k_{\mathsf{S}}$ as child of S with weight $w_{\mathsf{S},k}$
7: **end for**
8: **end for**

Introduce IVs and twins:

9: **for** $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
10: **for** $k \in \{1, \dots, K_{\mathsf{S}}\}$ **do**
11: Connect new IV $\lambda_{Z_{\mathsf{S}}=k}$ as child of $\mathsf{P}^k_{\mathsf{S}}$
12: **end for**
13: **if** $\mathbf{S}^c(\mathsf{S}) \neq \emptyset$ **then**
14: Introduce a twin sum node $\bar{\mathsf{S}}$ in $\mathcal{S}$
15: $\forall k \in \{1, \dots, K_{\mathsf{S}}\}$: connect $\lambda_{Z_{\mathsf{S}}=k}$ as child of $\bar{\mathsf{S}}$, and let $w_{\bar{\mathsf{S}},\lambda_{Z_{\mathsf{S}}=k}} = \bar{w}_{\mathsf{S},k}$
16: **for** $\mathsf{S}^c \in \mathbf{S}^c(\mathsf{S})$ **do**
17: **for** $k \in \{k \mid \mathsf{S} \notin \mathbf{desc}(\mathsf{P}^k_{\mathsf{S}^c})\}$ **do**
18: Connect $\bar{\mathsf{S}}$ as child of $\mathsf{P}^k_{\mathsf{S}^c}$
19: **end for**
20: **end for**
21: **end if**
22: **end for**

been connected to any other node in the SPN before. Since $\mathsf{P}^k_{\mathsf{S}}$ are reachable from the root, also $\lambda_{Z_{\mathsf{S}}=k}$ are reachable, i.e. Z_{S} is now in the scope of the root. Since S was complete before step 12, i.e. all links had the same scope, it remains complete after step 12. The scopes of all ancestors of S are now augmented by Z, which could corrupt decomposability and completeness of the ancestors. However, there can be no non-decomposable ancestors: if there was a non-decomposable product node it must have Z in the scope of at least two children. This would imply that S is reachable from both children, contradicting that the product node was decomposable beforehand. However, in general there can be incomplete sum nodes as illustrated in Figure 5.2. These are treated in steps $13 - 21$:

The twin sum $\bar{\mathsf{S}}$, if introduced, is clearly complete and has scope $\{Z\}$. Furthermore, incompleteness of any conditioning sum S^c can only be caused by links not having Z_{S} in their scope. The scope of these links is augmented by Z_{S} in step 18. These links remain decomposable and moreover, S^c is rendered complete now. These operations do not affect the scope of any further ancestor of S^c, since their scope already contains Z. Therefore, the SPN remains complete and decomposable, and adds Z_{S} into the root's scope. Finally, it is easy to see that all sum nodes have normalized weights, i.e. $\mathcal{S}'$ is a complete, decomposable and locally normalized SPN over $\mathbf{X} \cup \mathbf{Z}$. □

Proposition 5.2. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, $\mathbf{Y} \subseteq \mathbf{Z}_{\mathbf{S}(\mathcal{S})}$ and $\mathbf{Z} = \mathbf{Z}_{\mathbf{S}(\mathcal{S})} \setminus \mathbf{Y}$. Let $\mathbf{y} \in \mathbf{val}(\mathbf{Y})$ and let $\mathcal{S}' = \mathbf{aug}(\mathcal{S})$. It holds that*

1. *Each node in $\mathcal{S}^{\mathbf{y}}$ has the same scope as its corresponding node in $\mathcal{S}'$.*

Algorithm 9 Configure SPN

1: $\mathcal{S}^{\mathbf{y}} \leftarrow \mathbf{aug}(\mathcal{S})$
2: **for** $Y \in \mathbf{Y}$ **do**
3: **for** $y \in \mathbf{val}(Y), y \neq \mathbf{y}[Y]$ **do**
4: Delete $\lambda_{Y=y}$ and link $\mathsf{P}^{y}_{\mathsf{S}_Y}$
5: **end for**
6: **end for**
7: Delete all nodes in $\mathcal{S}^{\mathbf{y}}$ which are not reachable from the root

2. *$\mathcal{S}^{\mathbf{y}}$ is a complete and decomposable SPN over $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$.*

3. *For any node N in $\mathcal{S}^{\mathbf{y}}$ with $\mathbf{sc}(\mathsf{N}) \cap \mathbf{Y} = \emptyset$, we have that $\mathcal{S}^{\mathbf{y}}_{\mathsf{N}} = \mathcal{S}'_{\mathsf{N}}$.*

4. *For $\mathbf{y}' \in \mathbf{val}(\mathbf{Y})$ it holds that*

$$\mathcal{S}^{\mathbf{y}}(\mathbf{X}, \mathbf{Z}, \mathbf{y}') = \begin{cases} \mathcal{S}'(\mathbf{X}, \mathbf{Z}, \mathbf{y}') & \text{if } \mathbf{y}' = \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \tag{A.23}$$

Proof of Proposition 5.2. We restate Algorithm 4 in Algorithm 9.

ad *1.)* We show that *1.* holds after each iteration of the main for-loop (steps 3–6) of Algorithm 9. In each iteration, IVs $\lambda_{Y=y}$ are disconnected and links $\mathsf{P}^{y}_{\mathsf{S}_Y}$ are deleted for $y \neq \mathbf{y}[Y]$. Clearly, this can only change the scopes of their ancestors. The direct ancestors S_Y and $\bar{\mathsf{S}}_Y$ still have the same scope, since they are left with one child, either $\mathsf{P}^{\mathbf{y}[Y]}_{\mathsf{S}_Y}$ or $\lambda_{Y=\mathbf{y}[Y]}$. Any further ancestor is an ancestor of S_Y or $\bar{\mathsf{S}}_Y$. Since the scopes of S_Y or $\bar{\mathsf{S}}_Y$ remain the same, also the scopes of these ancestors remain the same.

ad *2.)* The graph of $\mathcal{S}^{\mathbf{y}}$ is rooted and acyclic, since the root can not be a link and deleting nodes and edges can not introduce cycles. When an IV $\lambda_{Y=y}$ is deleted, also the link $\mathsf{P}^{y}_{\mathsf{S}_Y}$ is deleted, so no internal nodes are left as leaves. The roots in $\mathcal{S}^{\mathbf{y}}$ and $\mathbf{aug}(\mathcal{S})$ are the same, and by point *1.*, $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$ is the scope of the root. $\mathcal{S}^{\mathbf{y}}$ is also complete and decomposable: After each iteration of the main for-loop (steps 3–6), S_Y and $\bar{\mathsf{S}}_Y$ are trivially complete, since they are left with a single child. $\mathsf{P}^{\mathbf{y}[Y]}_{\mathsf{S}_Y}$ is also clearly decomposable. Completeness or decomposability of any ancestor of S_Y or $\bar{\mathsf{S}}_Y$ is left intact, since S_Y or $\bar{\mathsf{S}}_Y$ do not change their scopes.

ad *3.)* According to *1.*, the scope of N in $\mathcal{S}'$ and $\mathcal{S}^{\mathbf{y}}$ is the same. Since $\mathbf{sc}(\mathsf{N}) \cap \mathbf{Y} = \emptyset$, the disconnected IVs and deleted links are no descendants of N, i.e. no descendants of N are disconnected in the for-loop (steps 3–6) in Algorithm 9. Since N is present in $\mathcal{S}^{\mathbf{y}}$, it must be still reachable from the root after Algorithm 9. Therefore also all descendants of N are reachable, i.e. $\mathcal{S}^{\mathbf{y}}_{\mathsf{N}} = \mathcal{S}'_{\mathsf{N}}$.

ad *4.)* When the input for $\mathcal{S}'$ is fixed to $\mathbf{x}, \mathbf{z}, \mathbf{y}$, all disconnected IVs $\lambda_{Y=y}$ and deleted links $\mathsf{P}^{y}_{\mathsf{S}_Y}$ would evaluate to zero. The outputs of S_Y and $\bar{\mathsf{S}}_Y$ are therefore the same in $\mathcal{S}'$ and $\mathcal{S}^{\mathbf{y}}$. All other ancestors of $\lambda_{Y=y}$ and $\mathsf{P}^{y}_{\mathsf{S}_Y}$ are ancestors of S_Y or $\bar{\mathsf{S}}_Y$, i.e. their outputs also are the same. This includes the root and therefore $\mathcal{S}^{\mathbf{y}}(\mathbf{x}, \mathbf{z}, \mathbf{y}) = \mathcal{S}'(\mathbf{x}, \mathbf{z}, \mathbf{y})$, for any $\mathbf{x}, \mathbf{z}$.

When $\mathbf{y}' \neq \mathbf{y}$, then there must be a $Y \in \mathbf{Y}$ such that the IV $\lambda_{Y=\mathbf{y}'[Y]}$ can not be reached from the root. Thus, with Lemma 4.1 it follows that if $\mathbf{y}' \neq \mathbf{y}$, then $\mathcal{S}^{\mathbf{y}}(\mathbf{x}, \mathbf{z}, \mathbf{y}') = 0$.

□

Lemma 5.1. *Let $\mathcal{S}$ be an SPN over $\mathbf{X}$, let S be a sum node in $\mathcal{S}$ and $\mathbf{Z} = \mathbf{Z}_{\mathbf{ancs}(\mathsf{S}) \setminus \{\mathsf{S}\}}$. For any $\mathbf{z} \in \mathbf{val}(\mathbf{Z})$, the configured SPN $\mathcal{S}^{\mathbf{z}}$ contains either S or its twin $\bar{\mathsf{S}}$, but not both.*

Proof of Lemma 5.1. $\mathcal{S}^{\mathbf{z}}$ must contain either S or $\bar{\mathsf{S}}$, since Z_{S} is in the scope of the root by Proposition 5.2 and any IV related to Z_{S} can be reached only via S or $\bar{\mathsf{S}}$. To show that *not both* are in $\mathcal{S}^{\mathbf{z}}$, let $\mathbf{\Pi}_k$ denote the set of paths of length k from the root to any node N with $Z_{\mathsf{S}} \in \mathbf{sc}(\mathsf{N})$. A path in $\mathbf{\Pi}_k$ is denoted as $(\mathsf{N}_1, \ldots, \mathsf{N}_k)$, cf. Section 2.2. For $k > 1$, all paths in $\mathbf{\Pi}_k$ can be constructed by extending all paths in $\mathbf{\Pi}_{k-1}$, i.e. each path in $\mathbf{\Pi}_{k-1}$ is extended by all children of the path's last node. Let K be the smallest number such that there is a path in $\mathbf{\Pi}_k$ containing S or $\bar{\mathsf{S}}$.

We show by induction, that $|\mathbf{\Pi}_k| = 1$, $k = 1, \ldots, K$. Note that $\mathbf{\Pi}_1$ contains a single path (N), where N is the root, therefore the induction basis holds.

For the induction step, we show that given $|\mathbf{\Pi}_{k-1}| = 1$, then also $|\mathbf{\Pi}_k| = 1$. Let $(\mathsf{N}_1, \ldots, \mathsf{N}_{k-1})$ be the single path in $\mathbf{\Pi}_{k-1}$. If N_{k-1} is a product node, then it has a single child C with $Z_{\mathsf{S}} \in \mathbf{sc}(\mathsf{C})$, due to decomposability. If N_{k-1} is a sum node, then it must be in $\mathbf{anc_S}(\mathsf{S}) \setminus \{\mathsf{S}\}$, and therefore has a single child in the configured SPN. Therefore, there is a single way to extend the path and therefore $|\mathbf{\Pi}_k| = 1, k = 1, \ldots, K$. This single path does either lead to S or $\bar{\mathsf{S}}$. Since $\mathsf{S} \notin \mathbf{desc}(\bar{\mathsf{S}})$ and $\bar{\mathsf{S}} \notin \mathbf{desc}(\mathsf{S})$, $\mathcal{S}^{\mathbf{z}}$ contains a single path to one of them, but not to both. □

A.3 Proofs for Chapter 6

Lemma 6.1. *For each complete, decomposable and selective SPN $\mathcal{S}$ and each sample $\mathbf{x}$, $\mathcal{T}_{\mathbf{x}}$ is a tree.*

Proof of Lemma 6.1. For each node N in $\mathcal{T}_{\mathbf{x}}$ there exists a calculation path $\Pi_{\mathbf{x}}(\mathsf{N})$ in $\mathcal{S}$ by definition. We have to show that this path is unique. Suppose there were two distinct calculation paths $\Pi_{\mathbf{x}}(\mathsf{N}) = (\mathsf{N}_1, \ldots, \mathsf{N}_J)$ and $\Pi'_{\mathbf{x}}(\mathsf{N}) = (\mathsf{N}'_1, \ldots, \mathsf{N}'_{J'})$. Since all calculation paths start at the root, there must be a smallest j such that $\mathsf{N}_j = \mathsf{N}'_j$, $\mathsf{N}_{j+1} \neq \mathsf{N}'_{j+1}$ and $\mathsf{N} \in \mathbf{desc}(\mathsf{N}_{j+1}) \wedge \mathsf{N} \in \mathbf{desc}(\mathsf{N}'_{j+1})$. Such N_j does not exist: if N_j is a product node, $\mathsf{N} \in \mathbf{desc}(\mathsf{N}_{j+1}) \wedge \mathsf{N} \in \mathbf{desc}(\mathsf{N}'_{j+1})$ contradicts decomposability. If N_j is a sum node, it would contradict selectivity, since $\mathsf{N}_{j+1}(\mathbf{x}) > 0$ and $\mathsf{N}'_{j+1}(\mathbf{x}) > 0$. Therefore each calculation path is unique, i.e. $\mathcal{T}_{\mathbf{x}}$ is a tree. □

References

[1] "OEIS Foundation Inc. (2011), The On-Line Encyclopedia of Integer Sequences, http://oeis.org/A000537."

[2] "ETSI: Digital cellular telecommunications system (phase 2+); Enhanced full rate (EFR) speech transcoding, ETSI EN 300 726 V8.0.1," Nov. 2000.

[3] J. Aczél, *Lectures on Functional Equations and their Applications*. Academic Press, 1966.

[4] M. R. Amer and S. Todorovic, "Sum-Product Networks for Modeling Activities with Stochastic Structure," in *Proceedings of CVPR*, 2012, pp. 1314–1321.

[5] S. Arnborg and Sjördin, "On the foundations of Bayesianism," in *Bayesian Inference and Maximum Entropy Methods in Science and Engineering, 20th International Workshop*, A. Mohammad-Djarafi, Ed., 2001, pp. 61–71.

[6] P. Billingsley, *Probability and Measure*. Wiley-Interscience, 1995.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Springer, Oct. 2007.

[8] H. L. Bodlaender, F. van den Eijkhof, and L. C. van der Gaag, "On the complexity of the MPA problem in probabilistic networks," in *Proceedings of the 15th European Conference on Artificial Intelligence*, 2002, pp. 675–679.

[9] V. I. Bogachev, *Measure Theory*. Springer Berlin Heidelberg New York, 2007, vol. I and II.

[10] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller, "Context-Specific Independence in Bayesian Networks," in *UAI*, 1996.

[11] W. Buntine, "Theory Refinement on Bayesian Networks," in *Uncertainty in Artificial Intelligence (UAI)*, 1991, pp. 52–60.

[12] H. Chan and A. Darwiche, "When do numbers really matter?" *JAIR*, vol. 17, pp. 265–287, 2002.

[13] ——, "Sensitivity analysis in Bayesian networks: From single to multiple parameters." in *UAI*, 2004, pp. 67–75.

[14] W. C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai, "Language Modeling with Sum-Product Networks," in *Proceedings of Interspeech*, 2014.

[15] D. M. Chickering, "Learning Bayesian Networks is NP-Complete," in *Learning from Data: Artificial Intelligence and Statistics V*. Springer-Verlag, New York, 1996, pp. 121–130.

[16] ——, "Learning Equivalence Classes of Bayesian-Network Structures," *Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.

[17] D. M. Chickering, D. Geiger, and D. Heckerman, "Learning Bayesian Networks is NP-hard," Technical Report No. MSR-TR-94-17, Microsoft Research, Redmond, Washington, Tech. Rep., 1994.

[18] ——, "Learning Bayesian networks: Search methods and experimental results," in *Proc. of Fifth Conference on Artificial Intelligence and Statistics*, 1995, pp. 112–128.

[19] D. M. Chickering and D. Heckerman, "A Bayesian approach to learning Bayesian networks with local structure," in *UAI*, 1997, pp. 80–89.

[20] C. K. Chow and C. N. Liu, "Approximating Discrete Probability Distributions with Dependence Trees," in *IEEE Transactions on Information Theory*, vol. 14, no. 3, 1968, pp. 462–467.

[21] M. Cooke, J. Barker, S. Cunningham, and X. Shao, "An audio-visual corpus for speech perception and automatic speech recognition," *J. Acoust. Soc. Amer.*, vol. 120, pp. 2421–2424, Nov 2005.

[22] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[23] R. Cowell, *Introduction to inference for Bayesian networks.* in Learning in Graphical Models, (M.I. Jordan, editor), MIT Press, 1999.

[24] R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter, *Probabilistic networks and expert systems.* Springer, 1999.

[25] R. Cox, "Probability, frequency, and reasonable expectation," *American Journal of Physics*, vol. 14, pp. 1–13, 1946.

[26] A. Darwiche, "A logical approach to factoring belief networks," in *Proceedings of KR*, 2002, pp. 409–420.

[27] ——, "A Differential Approach to Inference in Bayesian Networks," *ACM*, vol. 50, no. 3, pp. 280–305, 2003.

[28] C. de Campos, Z. Zeng, and Q. Ji, "Structure learning of Bayesian networks using constraints," in *International Conference on Machine Learning (ICML)*, 2009, pp. 113–120.

[29] B. de Finetti, *Foresight: Its Logical Laws, Its Subjective Sources*, H. E. Kyburg and H. E. K. Smokler, Eds. Robert E. Kreiger Publishing Co, 1937.

[30] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[31] A. Dennis and D. Ventura, "Learning the Architecture of Sum-Product Networks Using Clustering on Variables," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 2042–2050.

[32] D. Dubois and H. Prade, *Possibility Theory: an Approach to Computerized Processing of Uncertainty*, 1988.

[33] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, " Efficient Projections onto the L1-Ball for Learning in High Dimensions," in *ICML*, 2008.

[34] G. Elidan and N. Friedman, "Learning the dimensionality of hidden variables," in *Proc. 17th Conference on Uncertainty in Artificial Intelligence*, 2001.

[35] ——, "Learning Hidden Variable Networks: The Information Bottleneck Approach," *Journal of Machine Learning Research*, vol. 6, pp. 81–127, 2005.

[36] G. Elidan, N. Lotner, N. Friedman, and D. Koller, "Discovering Hidden Variables: A Structure-Based Approach," in *Neural Information Processing Systems*. MIT Press, 2001, pp. 479–485.

[37] N. Friedman, "The Bayesian structural EM algorithm," in *Proc. 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 129–138.

[38] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *SOFTWARE - PRACTICE AND EXPERIENCE*, vol. 30, no. 11, pp. 1203–1233, 2000.

[39] R. Gens and P. Domingos, "Discriminative Learning of Sum-Product Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 3248–3256.

[40] ——, "Learning the Structure of Sum-Product Networks," *Proceedings of ICML*, pp. 873–880, 2013.

[41] F. Glover, "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[42] D. Griffin and J. Lim, "Signal estimation from modified short-time Fourier transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[43] J. Halpern, "A counterexample to theorems of Cox and Fine," *Journal of Artificial Intelligence Research*, vol. 10, pp. 67–85, 1999.

[44] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," *Machine Learning*, vol. 20, pp. 197–243, 1995.

[45] T. Jaakkola, D. Sontag, A. Globerson, and M. Meila, "Learning Bayesian Network Structure using LP Relaxations," in *AI Statistics*, 2010, pp. 358–365.

[46] M. Jaeger, J. Nielsen, and T. Silander, "Learning probabilistic decision graphs ," *International Journal of Approximate Reasoning*, vol. 42, pp. 84–100, 2006.

[47] P. Jax and P. Vary, "On artificial bandwidth extension of telephone speech," *Signal Processing*, vol. 83, pp. 1707–1719, 2003.

[48] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[49] F. Jensen, *An introduction to Bayesian networks*. UCL Press Limited, 1996.

[50] D. Kisa, G. V. den Broeck, A. Choi, and A. Darwiche, "Probabilistic Sentential Decision Diagrams," in *KR*, 2014.

[51] M. Koivisto and K. Sood, "Exact Bayesian Structure Discovery in Bayesian Networks," *Journal of Machine Learning Research*, vol. 5, pp. 549–573, 2004.

[52] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[53] B. Kosko, "Fuzziness vs. Probability," *International Journal of General Systems*, vol. 17, pp. 211–240, 1990.

[54] J. Kwisthout, "Most Probable Explanations In Bayesian Networks: Complexity And Tractability ," *International Journal of Approximate Reasoning*, vol. 52, pp. 1452–1469, 2011.

[55] W. Lam and F. Bacchus, "Learning Bayesian belief networks: an approach based on the MDL principle," *Computational Intelligence*, vol. 10, no. 3, pp. 269–293, 1994.

[56] H. Langseth and T. Nielsen, "Latent Classification Models," *Machine Learning*, vol. 59, pp. 237–265, 2005.

[57] ——, "Classification using Hierarchical Naïve Bayes models," *Machine Learning*, vol. 63, pp. 135–159, 2006.

[58] S. Lauritzen and D. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. Royal Statistical Society B*, vol. 50, pp. 157–224, 1988.

[59] J. Le Roux, "Exploiting Regularities in Natural Acoustical Scenes for Monaural Audio Signal Estimation, Decomposition, Restoration and Modification," Ph.D. dissertation, The University of Tokyo & Université Paris, 2009.

[60] C. Leitner and F. Pernkopf, "Speech Enhancement Using Pre-Image Iterations," in *ICASSP*, 2012, pp. 4665–4668.

[61] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transaction on Communication*, vol. 28, no. 1, pp. 84–95, 1980.

[62] H.-A. Loeliger, "An Introduction to Factor Graphs," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 28–41, 2004.

[63] D. Lowd and A. Rooshenas, "Learning Markov Networks with Arithmetic Circuits," *Proceedings of AISTATS*, pp. 406–414, 2013.

[64] D. Lowd and P. Domingos, "Learning Arithmetic Circuits," in *Twenty Fourth Conference on Uncertainty in Artificial Intelligence*, 2008, pp. 383–392.

[65] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*. Cambridge, MA: MIT Press, 1999, pp. 355–368.

[66] C. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.

[67] J. Paris, *The Uncertain Reasoner's Companion*. Cambridge University Press, 1994.

[68] J. D. Park, "MAP Complexity Results and Approximation Methods," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2002, pp. 338–396.

[69] J. D. Park and A. Darwiche, "Complexity Results and Approximation Strategies for MAP Explanations ," *Journal of Artificial Intelligence Research*, vol. 21, pp. 101–133, 2004.

[70] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[71] R. Peharz, B. Geiger, and F. Pernkopf, "Greedy Part-Wise Learning of Sum-Product Networks," in *ECML/PKDD*, vol. 8189. Springer Berlin, 2013, pp. 612–627.

[72] R. Peharz, R. Gens, and P. Domingos, "Learning Selective Sum-Product Networks," in *ICML-LTPM Workshop*, 2014.

[73] R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf, "Modeling Speech with Sum-Product Networks: Application to Bandwidth Extension," in *Proceedings of ICASSP*, 2014.

[74] R. Peharz, M. Stark, and F. Pernkopf, "A Factorial Sparse Coder Model for Single Channel Source Separation," in *Interspeech*, 2010, pp. 386–389.

[75] R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos, "On Theoretical Properties of Sum-Product Networks," in *Proceedings of AISTATS*, 2015.

[76] R. Peharz and F. Pernkopf, "Exact Maximum Margin Structure Learning of Bayesian Networks," in *ICML*, 2012.

[77] F. Pernkopf, R. Peharz, and S. Tschiatschek, *Introduction to Probabilistic Graphical Models*, ser. Academic Press Library in Signal Processing. Elsevier, 2013, vol. 1, ch. 18.

[78] H. Poon and P. Domingos, "http://alchemy.cs.washington.edu/spn/," 2011, (online).

[79] ——, "Sum-Product Networks: A New Deep Architecture," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2011, pp. 337–346.

[80] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," in *Proceedings of the IEEE*, vol. 77, no. 2, 1989, pp. 257–286.

[81] M. Ratajczak, S. Tschiatschek, and F. Pernkopf, "Sum-Product Networks for Structured Prediction: Context-Specific Deep Conditional Random Fields." in *ICML-LTPM Workshop*, 2014.

[82] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[83] R. W. Robinson, "Counting labeled acyclic digraphs," in *New Directions in the Theory of Graphs*, F. Harary, Ed. Academic Press, NY, 1973, pp. 239–273.

[84] A. Rooshenas and D. Lowd, "Learning Sum-Product Networks with Direct and Indirect Variable Interactions," *ICML – JMLR W&CP*, vol. 32, pp. 710–718, 2014.

[85] J. S. Rosenthal, *A first look at rigorous probability theory*, 2nd ed. World Scientific, 2006.

[86] F. Samaria and A. Harter, "Parameterisation of a Stochastic Model for Human Face Identification," in *Proceedings of the 2nd IEEE Workshop on Applications of Computer Vision*, 1994, pp. 138–142.

[87] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[88] T. Silander and P. Myllymäki, "A simple approach for finding the globally optimal Bayesian network structure," in *UAI*, 2006, pp. 445–452.

[89] A. P. Singh and A. W. Moore, "Finding optimal Bayesian networks by dynamic programming," Carnegie Mellon University, Tech. Rep., 2005.

[90] G.-B. Song and P. Martynovich, "A study of HMM-based bandwidth extension of speech signals," *Signal Processing*, vol. 89, pp. 2036–2044, 2009.

[91] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. MIT Press, 2000.

[92] M. Stark, M. Wohlmayr, and F. Pernkopf, "Source-Filter based Single Channel Speech Separation using Pitch Information," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 19, no. 2, pp. 242–255, 2011.

[93] N. Sturmel and L. Daudet, "Signal reconstruction from STFT magnitude: a state of the art," pp. 375–386, 2011.

[94] J. Suzuki, "A construction of Bayesian networks from databases based on an MDL principle," in *UAI*, 1993, pp. 266–273.

[95] ——, "Learning Bayesian Belief Networks based on the minimum description length principle: An efficient algorithm using the B&B technique," in *Int. Conference on Machine Learning*, 1996, pp. 462–470.

[96] M. Teyssier and D. Koller, "Ordering-Based Search : A Simple and Effective Algorithm for Learning Bayesian Networks," in *Proc. 21st Conference on Uncertainty in Artificial Intelligence*, vol. 51, 2005, pp. 584–590.

[97] S. Tschiatschek, P. Reinprecht, M. Muecke, and F. Pernkopf, "Bayesian Network Classifiers with Reduced Precision Parameters," in *ECML PKDD*, 2012.

[98] K. S. Van Horn, "Constructing a logic of plausible inference: a guide to Cox's theorem," *International Journal of Approximate Reasoning*, pp. 3–24, 2003.

[99] T. Verma and J. Pearl, "An algorithm for deciding if a set of observed independencies has a causal explanation, ," in *UAI*, 1992.

[100] G. Vitali, *Sul problema della misura dei gruppi di punti di una retta.* Tip. Gamberini e Parmeggiani, 1905.

[101] H. Wettig, P. Grünwald, T. Roos, P. Myllymaki, and H. Tirri, "When discriminative learning of Bayesian network parameters is easy," in *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003, pp. 491–496.

[102] N. Zhang, "Hierarchical Latent Class Models for Cluster Analysis," *Journal of Machine Learning Research*, vol. 5, pp. 697–723, 2004.

[103] N. Zhang, T. Nielsen, and F. Jensen, "Latent variable discovery in classification models," *Artificial Intelligence in Medicine*, vol. 30, pp. 283–299, 2004.

List of Publications

[1] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf and Pedro Domingos. On Theoretical Properties of Sum-Product Networks. In *Proceedings of AISTATS*, 2015.

[2] Robert Peharz, Robert Gens, and Pedro Domingos. Learning Selective Sum-Product Networks. In *ICML Workshop on Learning Tractable Probabilistic Models*, 2014.

[3] Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling Speech with Sum-Product Networks: Application to Bandwidth Extension. In *Proceedings of ICASSP*, pages 3699–3703, 2014.

[4] Robert Peharz, Bernhard Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *Proceedings of ECML/PKDD*, 2013.

[5] Franz Pernkopf, Robert Peharz, and Sebastian Tschiatschek. *Introduction to Probabilistic Graphical Models*. Chapter 18 in E-Reference Signal Processing. Elsevier, 2013.

[6] Robert Peharz, Sebastian Tschiatschek, and Franz Pernkopf. The most generative maximum margin Bayesian networks. In *Proceedings of ICML*, pages 235–243, 2013.

[7] Robert Peharz and Franz Pernkopf. Exact maximum margin structure learning of Bayesian networks. In *Proceedings of ICML*, 2012.

[8] Robert Peharz and Franz Pernkopf. On linear and mixmax interaction models for single channel source separation. In *Proceedings of ICASSP*, pages 249–252, 2012.

[9] Robert Peharz and Franz Pernkopf. Sparse nonnegative matrix factorization with ℓ^0-constraints. *Neurocomputing*, 80:38–46, 2012.

[10] Robert Peharz, Michael Wohlmayr, and Franz Pernkopf. Gain-robust multi-pitch tracking using sparse nonnegative matrix factorization. In *Proceedings of ICASSP*, pages 5416–5419, 2011.

[11] Michael Wohlmayr, Robert Peharz, and Franz Pernkopf. Efficient implementation of probabilistic multi-pitch tracking. In *Proceedings of ICASSP*, pages 5412–5415, 2011.

[12] Robert Peharz, Michael Stark, and Franz Pernkopf. A factorial sparse coder model for single channel source separation. In *Proceedings of Interspeech*, 386–389, 2010.

[13] Robert Peharz, Michael Stark, and Franz Pernkopf. Sparse nonnegative matrix factorization using ℓ^0-constraints. In *Proceedings of MLSP*, 83–88, 2010.