Graz University of Technology

PhD Thesis

Sum-Product Networks for Complex Modelling Scenarios

Dipl.-Ing. Martin Trapp

Submitted for the degree of Doktor der technischen Wissenschaften
at the Graz University of Technology.

Supervisors:
Prof. Dr. Franz Pernkopf
Graz University of Technology, Austria

Asst. Prof. Dr. Robert Reharz
Eindhoven University of Technology, Netherlands

Examiners:
Prof. Dr. Franz Pernkopf
Graz University of Technology, Austria

Prof. Dr. Kristian Kersting
Darmstadt University of Technology, Germany

Graz, 12th July 2020

Statutory Declaration

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.

date	(signature)

This thesis is dedicated to Elias.

Contents

Abstract

Sum-Product Networks (SPNs) are flexible general-purpose probabilistic models that have received increasing attention due to their attractive inference properties. Even though there exists a large body of work on parameter and structure learning in SPNs, many of the existing approaches focus on rather simple modelling scenarios. For example, in the case of discriminative parameter learning, the labelled training examples are assumed to be abundant, and we generally consider SPNs to be defined only over a finite set of random variables. Moreover, most approaches to construct SPNs in a data-agnostic way rely on heuristic and ad-hoc strategies rather than proposing a principled solution.

In this thesis, we examine SPNs for complex modelling scenarios. We are particularly interested in: i) principled semi-supervised parameter learning in SPNs, which guarantees that the learner cannot deteriorate in performance when adding additional unlabelled data, ii) principled structure learning in SPNs that is mathematically sound, protects us from overfitting and enables learning under missing data, and iii) extending the framework of SPNs to model possibly infinitely many random variables, and thus, establishing SPNs as a stochastic process model.

As a first main contribution, we introduce an extension of the contrastive pessimistic likelihood for safe semi-supervised parameter learning in SPNs. Our approach is the first semi-supervised learning technique for SPNs, and often obtains a performance that is similar to an SPN trained on a fully labelled datasets. We first derive an objective for generative learning and later extend the approach to discriminative parameter learning. Lastly, we show empirical evidence that safe semi-supervised SPNs perform favourably compared to existing semi-supervised techniques on various classification tasks.

The second main contribution of this thesis is the introduction of principled structure learning in SPNs. While there exists a large body of work on structure learning, none of the approaches asks either of the two essential questions: “What is a good structure?” or “What is a principle to derive a good structure?”. We aim to change this practice and introduce a sound, Bayesian formulation for joint parameter and structure learning in SPNs. Our experiments show that this principled approach competes well with the prior art and that we gain several benefits, such as automatic protection against overfitting, robustness under missing data and a natural extension to nonparametric formulations.

As a third main contribution, we introduce deep structured mixtures of Gaussian processes, which combine tractable inference in SPNs with exact posterior inference in Gaussian processes. Our approach directly extends SPNs to the stochastic process case by equipping SPNs with Gaussian measures, which correspond to Gaussian processes, as leaves. We show that the resulting model allows a natural interpretation as exact Bayesian model averaging over a rich collection of naive-local expert models. In a series of experiments, we show that the proposed technique outperforms existing expert-based approaches and provides low approximation errors when used as an approximation to a Gaussian process.

In addition to the main contributions, we show that gradient-based optimisation in overparameterised SPNs results in intrinsic acceleration effects, which depend directly on the depth of the network. Furthermore, we introduce two formulations for nonparametric SPNs and discuss their advantages and limitations.

Acknowledgements

First of all, I would like to thank my advisors Franz Pernkopf and Robert Peharz. Without the advice and support of both of you, this thesis would probably have never been possible. I'm incredibly grateful to you for believing in me and supporting me in pursuing my research ideas even during difficult times. I also want to thank Robert and Zoubin Ghahramani for giving me the opportunity to visit the machine learning group at the University of Cambridge.

Further, I want to thank everyone in the probabilistic circuits community for making it what it is, an amazing area to work in. Some special thanks go to Antonio, for organising t-prime together with me, and to Guy Van den Broeck and Kristian Kersting for putting collaboration before competition.

I want to thank Tamas Madl for introducing me to sum-product networks and supporting me in developing research ideas on tractable models. Further, I want to thank all my colleagues from OFAI, especially Robert, Brigitte, Fri, Steffi, Anna, Paolo, and Marcin, for their support and the inspiring talks about research, life and beyond. Special thanks go to Dietmar for spending hours at the whiteboard together with me and always believing in me.

I also want to thank all my colleagues from SPSC. Even though I haven't spent much time in Graz, I have always enjoyed the friendly environment and the inspiring discussions. I want to especially thank Wolfgang and Christian for making me feel at home at SPSC and always having an open mind.

Besides my academic colleagues, I'm extremely grateful for all my friends who have supported me all these years. Special thanks go to Helmut, Matthias, Flo and Andreas for helping me through some difficult times.

Last but not least, I want to thank my family and my partner for always being there for me, supporting me and listening whenever I needed to talk to someone.

Nomenclature

Abbreviations

a.e.	almost everywhere
iff	if and only if
w.l.o.g.	without loss of generality
w.r.t.	with respect to

General Notation

$\mathbb{B}$	binary numbers
$\mathcal{D}$	dataset
$\mathcal{D}_{(\mathsf{L})}$	dataset associated with the domain of node L
$\det M$	determinant of M
$\mathbb{1}_{\{x>y\}}$	indicator function
M^{-1}	inverse of M
$\mathrm{L} \overset{K}{\underset{j=1}{\Sigma}} \mathrm{E}$	log-sum-exp operation
$\mathbb{N}$	natural numbers
K	number of classes
D	data dimensionality
N	number of observations
$\|M\|_i$	i^{th} norm of M
$\mathbb{R}$	real numbers
$\mathbb{R}_{>0}$	positive real numbers
$\Rightarrow$	implication
$\left\{ {n \atop k} \right\}$	Stirling partition number
$\boldsymbol{q}$	soft labels
$\boldsymbol{X}$	inputs / covariates
$\boldsymbol{y}$	outputs / labels
I	arbitrary index set

Measure/Set Theory

$\#A$	cardinality
$\mathcal{B}(\cdot)$	Borel σ-algebra
$\lambda(\cdot)$	Lebesgue measure
$\mathcal{G}, \mathcal{F}$	system of sets
Ω	measurable set
$\mu \times \nu$	product measure
$\mu(\cdot), \nu(\cdot)$	measure
$\mathcal{O}$	system of open sets
$\mathcal{P}(\cdot)$	power set
$\sigma(\cdot)$	σ-operator
$\mathcal{A} \otimes \mathcal{B}$	product σ-algebra
$\mathcal{A}, \mathcal{B}$	σ algebra
$A \cap B$	set intersection
$A \cup B$	set union
$A \setminus b$	set difference
$A \sqcup B$	union of disjoint sets
A^c	complement

Probability Theory

$\mathrm{Beta}\,(\alpha,\ \beta)$	Beta distribution
$\mathrm{B}\,(p)$	Bernoulli distribution
$\mathrm{Cat}(\boldsymbol{w})$	Categorical distribution
$\mathrm{CRP}(\alpha)$	Chinese restaurant process
$\mathrm{Dir}(\alpha_1, \ldots, \alpha_K)$	Dirichlet distribution
$\mathrm{DP}(\alpha\, H)$	Dirichlet process
$\mathbb{E}[X]$	expectation of X
$\mathrm{Exp}\,(\lambda)$	Exponential distribution
$\Gamma(\alpha, \beta)$	Gamma distribution
κ	kernel function
μ	mean, mean function
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution
$\mathrm{Poisson}\,(\lambda)$	Poisson distribution
$\mathbb{P}$	probability measure
$(\Omega, \mathcal{A}, \mathbb{P})$	probability space

$\mathrm{val}(X)$	range of a random variable	K	number of children
σ^2	variance	N	generic node
$\mathrm{Var}[X]$	variance of X	$\mathrm{par}(\mathsf{N})$	parents of node N
$\boldsymbol{K}$	kernel matrix	P	product node
$\boldsymbol{m}$	mean vector	$\mathbf{P}$	product nodes
$\boldsymbol{X}, \boldsymbol{F}$	random vector	ψ	scope function
$\boldsymbol{x}, \boldsymbol{f}$	realisation of $\boldsymbol{X}, \boldsymbol{F}$	R	region node
$F(X)$	cumulative density function	$\mathcal{R}$	region graph
$p(X \mid Y)$	conditional probability	$\mathcal{S}$	sum-product network
$p(X, Y)$	joint probability	$\mathcal{T}$	induced tree
X, Y, Z	random / latent variable	S	sum node
x, y, z	realisation of X, Y, Z	$\mathbf{S}$	sum nodes
Sum-Product Networks		θ	leaf parameters
$\mathrm{ch}(\mathsf{N})$	children of node N	$\boldsymbol{w}$	weights
$\mathcal{G}$	computational graph	$E(\mathcal{S})$	edge set of $\mathcal{S}$
L	leaf node	$V(\mathcal{S})$	vertex set of $\mathcal{S}$

1 Introduction

This thesis investigates sum-product networks for complex probabilistic modelling scenarios. Sum-product networks are flexible probabilistic general-purpose models that have received increasing attention due to their attractive inference properties. In this section, we will first motivate the use of probabilistic machine learning and describe why sum-product networks are an attractive tool for complex modelling tasks. Subsequently, we will present open questions, and objectives of this thesis in Section 1.2 and finally introduce the organisation and the contributions of this thesis in Section 1.3.

1.1 Probabilistic Machine Learning

Modern technology increasingly leverages techniques from artificial intelligence, e.g. to automate pattern-recognition tasks or support decision-making. Machine learning is one of the fundamental sub-areas of artificial intelligence research and has gained increasing attention outside of the academic field due to its success stories in pattern recognition, e.g. detection of road signs for autonomous driving or automatic translation of text from one language to another, and probabilistic modelling, e.g. forecasting of the COVID-19 spread under various hypothetical scenarios. The goal of machine learning is to answer the fundamental question: *How can a machine learn from experience?* Common strategies to approach this question consist in defining a set of assumptions about the task, forming the basis of a so-called model, and developing methods to improve the performance of the model for the task based on observed data (experience).

In the context of this thesis, we are particularly interested in probabilistic models, i.e. models that allow us to express and manipulate uncertainties through tools of probability theory. Let us assume, for example, that our task is to perform some form of prediction, such as the prediction of election outcomes. Of course, we expect any reasonable model for such a task to be uncertain about its predictions for unseen data. It should be noted that uncertainties can indeed occur in many different facets, from uncertainties introduced by measurement errors to uncertainties about the model itself, e.g. how flexible should the model be and what should the structure look like? Therefore, probabilistic machine learning, i.e. machine learning for probabilistic models, is inherently concerned with the representation and manipulation of all forms of uncertainties [1].

The probabilistic approach is particularly attractive as it not only allows us to consider all types of uncertainties but is also generally ap-

plicable and conceptually simple. In probabilistic machine learning, we obtain complex probabilistic models by composition of simpler models, with common probability distributions forming the basis of any probabilistic model. Moreover, interesting tasks can often be formulated as a form of inference about missing or latent variables, making probabilistic models a general-purpose approach. Although the probabilistic approach is conceptually attractive, we often encounter computational difficulties as many interesting inference tasks involve the computation of complex and high-dimensional sums or integrals. As a result, exact calculations are often infeasible or limited to simple models.

1.1.1 Modern Probabilistic Machine Learning

Recent techniques for probabilistic machine learning include Generative Adversarial Networks (GANs) [2], Variational Autoencoders (VAE) [3], Normalizing flows (NFlows) [4], [5] and Neural Autoregressive Distribution Estimation (NADE) [6]. However, most of these advances focus only on the expressivity of the model, often at the expense of efficient probabilistic inference. Probabilistic circuits[1], such as Sum-Product Networks (SPNs) [8] and Probabilistic Sentential Decision Diagrams (PSDDs) [9], on the other hand, promise to remedy this dilemma by guaranteeing exact and efficient computation of many inferential tasks, while being able to model complex relationships in the data. Table 1.1 shows a comparison of the inference capabilities of modern deep probabilistic architectures for a number of common inference scenarios.

In particular, although VAEs and GANs are excellent in representing complex high-dimensional data dependencies and have shown impressive results in image generation tasks, they fall short of expectations even in the simplest inference tasks, e.g. calculating the probability of an event. Therefore, it is necessary to rely on approximations for almost all inference scenarios, often without any guarantees on the approximation quality. Normalising flows, on the other hand, have been explicitly designed to allow efficient evaluation of their density function, i.e. we can calculate the probability of an event efficiently and exactly. However, more complex tasks, such as marginalisation, are generally intractable. Note that the capacities of flow-based models depend on the bijection used in the model.[2] NADEs are an interesting method, as they enable several inference scenarios to be solved exactly. For example, calculating conditionals and marginals can be done efficiently in certain settings, i.e. in case the conditioned variables appear first and the marginalised variables at the end of the order used in the model. However, the calculation of these inference tasks in general NADEs is intractable. SPNs, on the other hand, take advantage of the exact integration in compositional tractable distributions and allow a variety of exact inference scenarios, while at the same time being able to capture

[1] The term probabilistic circuits is an umbrella term for a certain class of probabilistic models and was introduced by Vergari, Choi, Peharz *et al.* [7] to emphasise the relationship between these models.

[2] Affine flows with a Gaussian base distribution using a diagonal covariance structure allow for tractable integration and calculation of moments. However, it is currently not well understood which subclasses of normalising flows allow more advanced inference scenarios.

complex dependencies by using hierarchically structured mixture distributions. The result is an attractive model that has proven to be competitive with neural networks and additionally guarantees exact inference [10]. Although most inference scenarios can be calculated exactly, the calculation of the Maximum-A-Posteriori (MAP) estimate can only be approximated. PSDDs, which are a more restrictive model class than SPNs, enforce even stronger structural assumptions and can additionally guarantee the exact computation of any maximum-a-posteriori estimate. However, to obtain competing results, it is often necessary to use large ensembles of PSDDs [11].

Inference Task	GAN	VAE	NFlow	NADE	SPN	PSDD
Sampling	Y	Y	Y	Y	Y	Y
Density	N	N/Y	Y	Y	Y	Y
Marginals	N	N	?	N/Y	Y	Y
Conditionals	N	N	?	N/Y	Y	Y
Moments	N	N	?	N	Y	Y
Max-a-posteriori	N	N	?	N	N/Y	Y

Table 1.1: Comparison of probabilistic inference capabilities for recent deep architectures, adapted from [12]. Y indicates that the inference task can be solved exactly, N/Y indicates the task can be solved approximately, and N indicates that the inference task is intractable.

1.2 Research Questions

In recent years, there has been a rise of novel techniques for parameter and structure learning in SPNs, as well as various flexible extensions of SPNs for complex modelling domains. State-of-the-art SPN parameter learning covers a wide range of well-developed techniques, with various approaches for generative learning.[3] Many approaches maximise the log-likelihood using either gradient-based optimisation [10], [13], [14] or expectation-maximisation (and related schemes) [8], [13], [15], [16]. In addition, there are a variety of Bayesian approaches, each of which utilises approximate posterior inference, e.g. [17]–[19]. Although many techniques have been introduced over the years, little work has been done to analyse the intrinsic behaviour of parameter learning in SPNs. While there is a pearl of conventional wisdom that parameter learning in deep SPNs is faster than in shallow models, the effects of overparametrisation (increased depth) in SPNs is only little understood. This calls for a *theoretic analysis of the acceleration effects in overparameterised SPNs.*

Safe Semi-Supervised Learning

In addition to generative parameter learning, there have been some advances in the realm of discriminative learning[4]. Most notably, Gens and

[3] In the context of this thesis, generative learning refers to learning the parametrisation of a probability distribution used for density estimation or probabilistic reasoning.

[4] Discriminative learning is concerned with learning the parameters of a model that can discriminate well between different pre-specified classes, e.g. distinguishing

Domingos [20] introduced the first parameter learning technique for discriminative learning by optimising the conditional log-likelihood using backpropagation. Recently, Peharz, Vergari, Stelzner *et al.* [10] proposed to learn a hybrid objective by combining the cross-entropy term and the log-likelihood to trade-off between generative and discriminative learning. However, all of these approaches assume that labelled data is abundant, hindering their application in domains in which obtaining class labels is expensive and sometimes infeasible for large amounts of data, e.g. text domain [21], image domain [22], [23] and the domain of biological data (genomics, proteomics, gene expression) [24]–[26]. In contrast to methods developed for SPNs, there exists a large body of work on semi-supervised[5] techniques for traditional machine learning models, e.g. [27], [28], and deep learning, e.g. [23], [29], [30]. The most natural approach to achieve semi-supervised learning for SPNs is self-training, i.e. imputation using the MAP estimate and re-training the model parameters using the imputed labels. However, such an approach can reinforce poor predictions, resulting in a potential degeneration of the model with increasing amounts of unlabelled data [30]. More sophisticated approaches, which could be adopted for SPNs, often exploit low-density regions, e.g. [31], [32], or the geometry of the data, e.g. [33], [34]. However, these approaches can yield sub-optimal accuracy if the induced assumptions are not met, possibly leading to a decrease in accuracy when adding unlabelled data [35]. Therefore, calling for parameter learning techniques which *guarantee that increasing amounts of unlabelled data can increase but not degenerate the performance of the semi-supervised learner.*

Bayesian Structure Learning

A key challenge in learning and applying SPNs is to define a suitable structure. To overcome this issue, a large number of mostly heuristic algorithms for structure learning in SPNs have been introduced. For example, the most prominent structure learning scheme, LearnSPN [36], derives an SPN structure by recursively clustering the data instances (yielding sum nodes) and partitioning data dimensions (yielding product nodes). Each of these steps can be understood as some local structure improvement, and as an attempt to optimise a local criterion. While LearnSPN is an intuitive scheme and elegantly maps the structural SPN semantics onto an algorithmic procedure, the fact that the global objective of structure learning is not declared is unsatisfying. This shortcoming is shared by its many variants such as online LearnSPN [37], ID-SPN [38], LearnSPN-b [39], and mixed SPNs [40]. Note that other approaches also lack a sound learning principle, such as [41], [42], which derive SPN structures from k-means and SVD clustering, respectively. Further, Peharz, Geiger and Pernkopf [43] grow SPNs bottom up using a heuristic based on the information bottleneck, Dennis and Ventura [44] use a heuristic structure exploration, and Kalra, Rashwan, Hsu *et al.* [45] use a variant of hard EM to decide when to enlarge or shrink an SPN structure. All of the mentioned approaches fall short in asking either of the

between x-ray images of healthy and sick patients.

[5] Semi-supervised learning aims to utilise additional unlabelled data to learn the parameters of a model more effectively.

following fundamental questions: *What is a good SPN structure?* or *What is a good principle to derive an SPN structure?*

Deep Structured Mixtures of Gaussian Processes
Orthogonal to work on parameter and structure learning, there has been a rise of flexible extensions of SPNs for complex modelling domains, e.g. SPNs over VAE experts [46], SPNs as automated statistician [17], and SPNs extended to imprecise probabilities [47]. However, all of these extensions are usually restricted to finite data domains, with the exception of [48], and cannot directly be applied in case of time-series data or in situations in which data is observed incrementally. Furthermore, the standard definition of SPNs is based on the assumption that the set of random variables is finite, limiting their applicability as stochastic process models to settings with a finite index set, i.e. random vectors [49]. Probabilistic regression models, such as Gaussian processes [50], on the other hand, are well understood and present a promising avenue for flexible extensions of SPNs. This raises the question *how can we extend SPNs to stochastic process models and integrate Gaussian processes?*

1.3 Contributions & Organisation

This thesis is organised into a chapter introducing the necessary background materials (Chapter 2) followed by a chapter reviewing SPNs (Chapter 3). The subsequent chapters reflect the main contributions of this thesis, i.e. semi-supervised learning in SPNs (Chapter 4), Bayesian structure and parameter learning (Chapter 5) and finally an extension of SPNs as stochastic process model by integrating Gaussian processes (Chapter 6). We conclude the thesis in Chapter 7 by discussing open challenges and potential future directions. The individual contributions of this thesis can be summarised as follows:

Implicit Acceleration Effects in Parameter Learning
In Section 3.4, we show that parameter optimisation in overparameterised SPNs has similar dynamics as observed in linear neural networks [51]. Gradient-based optimisation with small, fixed learning rate and near-zero initialisation of the SPN's weights results in an implicitly adaptive and time-varying learning rate with additional momentum term. Thus, leading to inherent acceleration effects, which depend on the depth of the network. Further, we show that an overparameterised SPN can represent any naturally deep tree-structured SPN.

Generative & Discriminative Safe Semi-Supervised Learning
In Chapter 4, we introduce the first semi-supervised learning technique for SPNs by extending the work on contrastive pessimistic likelihood estimation [35] to SPNs. We show that our approach can be applied to generative and discriminative parameter learning, both guaranteeing that in expectation adding unlabelled data can increase, but not degrade, the performance of the learner on the training set. Furthermore, our approach exploits the tractability of SPNs and has computational

cost linear in the number of data points and model parameters, while inducing little assumptions on the data distribution.

Bayesian Learning of Sum-Product Networks
In Chapter 5, we introduce the first principled approach to structure (and joint parameter) learning by posing the problem as Bayesian inference in a latent variable model. A critical insight for our approach is to decompose structure learning into two steps, namely constructing a computational graph and separately learning the SPN's scope-function – determining the "effective" structure of the SPN as discussed in Section 3.2.1. This decomposition has some interesting implications, as it: i) enables the derivation of principled structure learning, ii) expresses the connection between various types of probabilistic circuits through the scope-function, and iii) unifies the view on structure learning in SPNs. Our experiments show that principled learning competes well with the prior art and that we gain several benefits, such as implicit protection against overfitting, robustness under missing data and a natural extension to nonparametric formulations.

Nonparametric Formulations of Sum-Product Networks
In Section 5.5, we introduce two different nonparametric extensions of SPNs. The first approach augments tree-structured SPNs with so-called group nodes, nodes that represent a uniform prior over all possible partitions of the scope of the parent node. The latter utilises the Bayesian framework presented in Chapter 5 to perform efficient approximate posterior inference using distributed slice sampling [52] in an infinite mixture of Bayesian SPNs.

Sum-Product Networks as Stochastic Process Model
In Chapter 6, we introduce deep structured mixtures of Gaussian processes, which combine SPNs with Gaussian processes as sub-modules, i.e. leaf distributions. For this, we first introduce a measure-theoretic perspective on SPNs in Section 3.1.2, which allows the extension of SPNs to infinitely many random variables by utilising Gaussian processes as leaves. Furthermore, we show that our model, which is a sound stochastic process model, enables efficient and exact posterior inference and has attractive computation costs for hyperparameter optimisation. We discuss that deep structure mixtures of Gaussian processes can be understood to perform exact Bayesian model averaging over a large set of naive-local-experts models and show how to exploit the structure to speed-up computations and model non-stationary data.

2 Background

A central concept in modern mathematics is the notion of a measure, which is foundational for many fields of mathematics such as probability and integration theory. Therefore, we will review the main concepts of measure theory and subsequently introduce probability theory using probability measures and lastly discuss fundamental concepts of graph theory, which are relevant in the context of sum-product networks. Note that the primer on measure theory and probability theory provided in this thesis is admittedly a somewhat steep introduction into the field. We refer the interested reader to the excellent book by Schilling [53] for a thorough and more detailed introduction into measure theory and refer to the book by Kolmogoroff [54] for details on probability theory.

2.1 Primer on Measure Theory

Measure theory naturally deals with the question of how one can assign measurement values, e.g. size or length, to a set (of objects). Some of the most natural measures are the *counting measure*[6], the *Lebesgue measure*[7], and the *probability measure*[8]. This section will review the most relevant concepts and introduce the necessary notation.

Before we define measurable spaces and measures, let us briefly review some notation on set theory and countability. In addition to the common notation of a set union $A \cup B$, intersection $A \cap B$ and complement A^c, we write $A \sqcup B$ for a union of pairwise disjoint sets. Further, we use $A \setminus b$ for the exclusion of b from the set A. We say that a set is *countable*, if the cardinality, denoted as $\#A$, of a set A is $\#A \leq \#\mathbb{N}$. A set with $\#A > \#\mathbb{N}$ is said to be *uncountable*. Note that sets with $\#A = \#\mathbb{N}$ are sometimes called countably infinite, while we will not make this distinction.

Say we are given the countable set $\Omega = \{\bigcirc, \triangle, \square, \square\}$ and we aim to define a measure μ over a subset of Ω, e.g. we might want to measure the similarity of objects in Ω. We would expect that any faithful measure $\mu(\cdot)$ for our purposes returns non-negative values and is countable

[6] The counting measure counts the number of elements in a set. If the set in question is finite, the value of a counting measure is the cardinality of the set and positive infinity otherwise.

[7] The Lebesgue measure is a generalising measure of length or area for D-dimensional metric spaces.

[8] As the name suggests, a probability measure assigns probabilities to sets of events.

additive, i.e.

$$\mu(A) \geq 0 \quad \forall A \in \underbrace{\mathcal{P}((\bigcirc, \triangle, \square, \square))}_{\text{power set of } \Omega}, \quad \mu(\emptyset) = 0\,, \tag{2.1}$$

$$\mu(\{\bigcirc, \triangle\}) + \mu(\{\square, \square\}) = \mu(\{\bigcirc, \triangle, \square, \square\})\,. \tag{2.2}$$

power set

Note that the power set, denoted as $\mathcal{P}(\Omega)$, is the set of all subsets (including the empty set $\emptyset$ and the set Ω itself), i.e. $\mathcal{P}(\Omega) = \{A \mid A \subseteq \Omega\}$. In the following we will formally define the notion of a (positive) measure μ, fulfilling the mentioned requirements, as a function mapping from a so-called σ-algebra $\mathcal{A}$ to values in the interval $[0, \infty]$, i.e. $\mu : \mathcal{A} \to [0, \infty]$.

σ-algebra

Definition 2.1 (σ-algebra)**.** *Given a set Ω, a σ-algebra $\mathcal{A}$ is a family of subsets with the following properties:*

$$\Omega \in \mathcal{A}\,, \tag{2.3}$$

$$A \in \mathcal{A} \Rightarrow A^c \in \mathcal{A}\,, \tag{2.4}$$

$$(A_n)_{n\in\mathbb{N}} \subset \mathcal{A} \Rightarrow \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{A}\,. \tag{2.5}$$

We can see from Equations (2.3)–(2.4) that $\emptyset \in \mathcal{A}$, because $\Omega^c = \emptyset$. Further, we see that if A and B are elements of $\mathcal{A}$, then $A \cup B \in \mathcal{A}$, c.f. Equation (2.5). Finally, any σ-algebra is closed under countable intersections, i.e.

$$(A_n)_{n\in\mathbb{N}} \subset \mathcal{A} \Rightarrow \bigcap_{n\in\mathbb{N}} A_n \in \mathcal{A}\,. \tag{2.6}$$

Theorem 2.1. *For any given Ω, the intersection $\cap_i \mathcal{A}_i$ of arbitrarily many σ-algebras on Ω is again a σ-algebra on Ω.*

Proof. Equation (2.3) Since $\Omega \in \mathcal{A}_i$ for all i, we have $\Omega \in \cap_i \mathcal{A}_i$.

Equation (2.4) If $A \in \cap_i \mathcal{A}_i$, then $A^c \in \mathcal{A}_i$ by Equation (2.4) for all i and, therefore, $A^c \in \cap_i \mathcal{A}_i$.

Equation (2.5) If $(A_n)_{n\in\mathbb{N}} \subset \cap_i \mathcal{A}_i$, then for all i we have $A_n \in \mathcal{A}_i$ and, therefore, $\cup_{n\in\mathbb{N}} A_n \in \cap_i \mathcal{A}_i$.

□

We will now review a few relevant examples of σ-algebras.

Example 2.1 (maximal σ-algebra)**.** *For any Ω, the power set $\mathcal{P}(\Omega)$ is the maximal (largest) σ-algebra on Ω.*

Example 2.2 (minimal σ-algebra)**.** *For any non-empty Ω, the set $\{\emptyset, \Omega\}$ is the minimal (smallest) σ-algebra on Ω.*

Example 2.3 (Borel σ-algebra)**.** *The σ-algebra generated by Borel sets on $\Omega = \mathbb{R}$, i.e. open sets in Ω, is called a Borel σ-algebra, see Definition 2.5.*

measurable space

Given a set Ω and a σ-algebra $\mathcal{A}$ on Ω, the tuple $(\Omega, \mathcal{A})$ is said to be a *measurable space*.

Definition 2.2 (measure)**.** *Given a measurable space $(\Omega, \mathcal{A})$, a (positive) measure μ on Ω is a function $\mu : \mathcal{A} \to [0, \infty]$, which satisfies:* *measure*

$$\mu(\emptyset) = 0\,, \tag{2.7}$$

$$\mu\underbrace{\left(\bigsqcup_{n\in\mathbb{N}} A_n\right)}_{\text{disjoint union}} = \sum_{n=\mathbb{N}} \mu(A_n)\,. \tag{2.8}$$

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

Note that a measure with $\mu(\Omega) < \infty$ is called a *finite measure* and a positive measure $\mathbb{P}$ with $\mathbb{P}(\Omega) = 1$ is said to be a *probability measure*. finite measure probability measure

Definition 2.3 (probability space)**.** *Given a sample space Ω, a σ-algebra $\mathcal{A}$ on Ω and a probability measure $\mathbb{P}$, a probability space is the triple $(\Omega, \mathcal{A}, \mathbb{P})$ and $\mathbb{P}(A) \geq 0$, for any $A \in \mathcal{A}$, is called the probability of the event A.* *probability space*

Let $\mathbb{P}_1, \ldots, \mathbb{P}_L$ be probability measures on $(\Omega, \mathcal{A})$, then the convex combination

$$\mathbb{P}(A) = \sum_{i=1}^{L} \beta_i \mathbb{P}_i(A)\,, \quad \beta \geq 0\,, \quad \sum_{i=1}^{L} \beta_i = 1\,, \tag{2.9}$$

is a probability measure on $(\Omega, \mathcal{A})$. Note that from Equation (2.8), it follows that the if events A and B are disjoint, then

$$\mathbb{P}(A \sqcup B) = \mathbb{P}(A) + \mathbb{P}(B)\,. \tag{2.10}$$

A measure μ on $(\Omega, \mathcal{A})$ is said to be σ-finite if there exist countably many $A_1, A_2, \cdots \in \mathcal{A}$ with $\mu(A_n) < \infty$ for all $n \in \mathbb{N}$ such that either i) $\cap_{n\in\mathbb{N}} A_n = \Omega$ or ii) $\cup_{n\in\mathbb{N}} A_n = \Omega$. Note that the Lebesgue measure and any probability measure are σ-finite.

Given a set Ω, a convenient way to define a σ-algebra on Ω seems to be to simply use the power set $\mathcal{P}(\Omega)$. However, such an approach will often be problematic as $\mathcal{A} = \mathcal{P}(\Omega)$ can often be too large to define an interesting measure on $\mathcal{A}$.

Example 2.4. *In case of a probability measure for possibly unfair coin flips we have $\Omega = \{H, T\}$ but we can only assign a measure (probability) to the following events: $\{\emptyset, \Omega\}$ as we do not know the probability of heads (H) or tails (T). Thus, defining a reasonable μ over $\mathcal{P}(\Omega)$ is not possible, and we have to refrain to a smaller σ-algebra, i.e. $\mathcal{A} = \{\emptyset, \Omega\}$*

Example 2.5. *In case we aim to define a uniform probability measure for $\Omega = [0, 1]$, the resulting power set will contain* non-measurable *sets (non-measurable in the sense of Lebesgue measurability), i.e. it will contain Vitali sets [55]. Therefore, we can only define the trivial measure of $\mu(A) = 0$ for all $A \in \mathcal{P}(\Omega)$ in such case or have to exclude those sets that are non-measurable from our σ-algebra [56, pp. 401–402].*

We refer to Schilling [53, pp. 429–436] and Oxtoby [57, pp. 22-3] for a detailed discussion on the construction of non Borel measurable spaces using the *axiom of choice*. To construct one of the most important

measures, i.e. the Lebesgue measure, we need to define an appropriate σ-algebra on the system of open sets $\mathcal{O}$ such that μ has desired properties and is not a trivial measure. For this purpose, let us first introduce the notion of the σ operator.

σ-operator

Definition 2.4 (generated σ-algebra)**.** *For every system of sets $\mathcal{G} \subset \mathcal{P}(\Omega)$ there exists a minimal σ-algebra, which contains $\mathcal{G}$. Therefore, we denote the σ-operator as*

$$\sigma(\mathcal{G}) \coloneqq \bigcap_{\substack{\mathcal{G}\subseteq\mathcal{F} \\ \mathcal{F}\ \sigma\text{-alg.}}} \mathcal{F} \tag{2.11}$$

and say that $\mathcal{A} \coloneqq \sigma(\mathcal{G})$ is generated by $\mathcal{G}$. Note that $\mathcal{A}$ is indeed a valid σ-algebra as the intersection of arbitrary many σ-algebras is a σ-algebra. Further, $\mathcal{A}$ is in fact the minimal σ-algebra containing $\mathcal{G}$ [53].

We can now use the σ-operator to generate a σ-algebra on the system of open sets, which is called Borel σ-algebra.

Borel σ-algebra

Definition 2.5 (Borel σ-algebra)**.** *The σ-algebra generated by a system of open sets $\mathcal{O} \subset \mathcal{P}(\mathbb{R}^D)$ is called Borel σ-algebra and denoted by $\mathcal{B}(\mathbb{R}^D)$, i.e.*

$$\mathcal{B}(\mathbb{R}^D) \coloneqq \sigma(\mathcal{O})\,. \tag{2.12}$$

The elements of $\mathcal{B}(\mathbb{R}^D)$ are called Borel sets and $(\mathbb{R}^D, \mathcal{B}(\mathbb{R}^D))$ is a Borel space.

In case of $\Omega = \mathbb{R}^D$, we can find several generators for Borel σ-algebras, such as using open rectangles

$$\mathcal{B}(\mathbb{R}^D) = \{(a_1, b_1) \times \cdots \times (a_D, b_D) \,|\, a_i < b_i\,, a_i, b_i \in \mathbb{R}\}\,, \tag{2.13}$$

or half-open rectangles

$$\mathcal{B}(\mathbb{R}^D) = \{[a_1, b_1) \times \cdots \times [a_D, b_D) \,|\, a_i \leq b_i\,, a_i, b_i \in \mathbb{R}\}\,. \tag{2.14}$$

After having introduced the notion of a Borel σ-algebra, we can now define one of the most important measures, the Lebesgue measure.

Lebesgue measure

Definition 2.6 (Lebesgue measure)**.** *The Lebesgue measure λ is a measure on the Borel sets $\mathcal{B}(\mathbb{R}^D)$ and has the following properties for $B \in \mathcal{B}(\mathbb{R}^D)$*

- *$\lambda(x + B) = \lambda(B)\,, x \in \mathbb{R}^D$ (translation invariance)*
- *$\lambda(T^{-1}(B)) = \lambda(B)$, where T is a congruence transformation[9] (congruence invariance)*
- *$\lambda(M^{-1}(B)) = |\det(M)|^{-1}\lambda(B)$ for any invertible $M \in \mathbb{R}^{D\times D}$*

Note that even though the Lebesgue measure is not finite, it is a σ-finite measure. In the course of this thesis we will further encounter *product measures*, *measurable functions* and *push-forward measures*. We will, therefore, briefly review these concepts.

[9] Congruence transformations are transformations, which do not change the geometry of an object. These transformations include: translation, rotation, reflection, and transflection.

Definition 2.7 (measurable function)**.** *Given two measurable spaces,* $(\Omega_1, \mathcal{A})$ *and* $(\Omega_2, \mathcal{B})$*, the map* $f\colon \Omega_1 \to \Omega_2$ *is called* $\mathcal{A}$*-measurable or* $\mathcal{A}/\mathcal{B}$*-measurable if for every* $B \in \mathcal{B}$ *the pre-image is in* $\mathcal{A}$*, i.e.*

measurable function

$$f^{-1}(B) \in \mathcal{A}\,, \qquad \forall B \in \mathcal{B}\,. \tag{2.15}$$

We denote a measurable function using $f\colon (\Omega_1, \mathcal{A}) \to (\Omega_2, \mathcal{B})$ to emphasise that measurability of a function is with respect to $\mathcal{A}$ and $\mathcal{B}$.

Theorem 2.2. *Given two measurable spaces,* $(\Omega_1, \mathcal{A})$ *and* $(\Omega_2, \mathcal{B})$ *and* $f\colon (\Omega_1, \mathcal{A}) \to (\Omega_2, \mathcal{B})$*, then for every measure* μ *on* $(\Omega_1, \mathcal{A})$*,*

$$\mu'(B) = \mu(f^{-1}(B)) \coloneqq \{\omega_1 \in \Omega_1 \,|\, f(\omega_1) \in B\} \in \mathcal{A}\,, \quad \forall B \in \mathcal{B} \tag{2.16}$$

defines a so-called push-forward measure on $(\Omega_2, \mathcal{B})$ *and is denoted by* $\mu \circ f^{-1}(\cdot)$*.*

push-forward measure

We refer to Schilling [53, p. 55] for a proof that a push-forward measure is a proper measure.

Definition 2.8 (product σ-algebra)**.** *Given two measure spaces, denoted as* $(\Omega_1, \mathcal{A}, \mu)$ *and* $(\Omega_2, \mathcal{B}, \nu)$*, the* σ*-algebra on the Cartesian product* $\Omega_1 \times \Omega_2$ *generated by the subsets of* $\mathcal{A}$ *and* $\mathcal{B}$*, i.e.*

product σ-algebra

$$\underbrace{\mathcal{A} \otimes \mathcal{B}}_{\text{product } \sigma\text{-algebra}} \coloneqq \sigma(\mathcal{A} \times \mathcal{B})\,, \tag{2.17}$$

is called a product σ*-algebra. The product measure space is denoted as* $(\Omega_1 \times \Omega_2, \mathcal{A} \otimes \mathcal{B}, \mu \times \nu)$*.*

The measure $\tilde{\mu} = (\mu \times \nu)$ on a product measure space $(\Omega_1 \times \Omega_2, \mathcal{A} \otimes \mathcal{B}, \tilde{\mu})$ is said to be a *product measure* and has the property

product measure

$$\tilde{\mu}(A \times B) = \mu(A)\,\nu(B)\,, \tag{2.18}$$

for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$. Note that product measures are only uniquely defined for σ-finite measures, e.g. probability or Lebesgue measures. As we will see later on, a product measure can, in some cases, be understood as a joint distribution of independent random variables.

Theorem 2.3 (Fubini-Tonelli)**.** *Given two* σ*-finite measure spaces denoted as* $(\Omega_1, \mathcal{A}, \mu)$ *and* $(\Omega_2, \mathcal{B}, \nu)$ *and let* $f\colon \Omega_1 \times \Omega_2 \mapsto \mathbb{R}$ *be a positive* $\mathcal{A} \otimes \mathcal{B}$*-measurable function w.r.t. the product measure space* $(\Omega_1 \times \Omega_2, \mathcal{A} \otimes \mathcal{B}, \mu \times \nu)$*. Then*

Fubini-Tonelli theorem

$$\int_{\Omega_1 \times \Omega_2} f \,\mathrm{d}(\mu \times \nu) = \int_{\Omega_1} \left[\int_{\Omega_2} f(\omega_1, \omega_2) \mu(\,\mathrm{d}\omega_1) \right] \nu(\,\mathrm{d}\omega_2) \tag{2.19}$$

$$= \int_{\Omega_2} \left[\int_{\Omega_1} f(\omega_1, \omega_2) \nu(\omega_2) \right] \mu(\,\mathrm{d}\omega_1)\,. \tag{2.20}$$

Further, if at least one of the three integrals is finite, all three are finite.

The Fubini-Tonelli theorem states that we can integrate a measurable function w.r.t. a product measure in arbitrary order.

2.2 Probability Theory

After having revised the concepts of measure theory we will now briefly review relevant aspects of probability theory based on the foundational work of Kolmogoroff [54]. We refer to any of the following works for a more rigorous and detailed introduction [54], [58], [59]. One of the central concepts of probability theory are *independence*, *conditional probabilities* and *random variables*. In the course of this section, we will review theses concepts and discuss the common rules of probability.

First recall that a measure space $(\Omega, \mathcal{A}, \mathbb{P})$ is said to be a probability space if the measure $\mathbb{P}$ is a probability measure, i.e. $\mathbb{P}(\Omega) = 1$ and $\mathbb{P}(A) \geq 0$ for any $A \in \mathcal{A}$.

conditional probability

Definition 2.9 (conditional probability)**.** *Given two events A and B in a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ with $\mathbb{P}(B) > 0$, the conditional probability of A given B is defined as the quotient of the joint probability of A and B, and the probability of B, i.e.*

$$\mathbb{P}(A \mid B) \coloneqq \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)}, \tag{2.21}$$

where $\mathbb{P}(A, B) = \mathbb{P}(A \cap B)$.

Note that the notion of conditional probabilities can be extended to the case of $\mathbb{P}(B) = 0$. We refer to Kolmogoroff [54] for details.

From Definition 2.9 directly follows that

$$\mathbb{P}(A, B) = \mathbb{P}(B)\,\mathbb{P}(A \mid B), \tag{2.22}$$

product rule

which by induction results in the so-called chain or *product rule*,

$$\mathbb{P}(A_1, A_2, \ldots, A_n) = \mathbb{P}(A_1)\,\mathbb{P}(A_2 \mid A_1) \ldots \mathbb{P}(A_n \mid A_1, \ldots, A_{n-1}). \tag{2.23}$$

Note that Equation (2.22) is symmetric and can be equally written as

$$\mathbb{P}(A, B) = \mathbb{P}(A)\,\mathbb{P}(B \mid A), \tag{2.24}$$

Bayes' rule

allowing us to arrive at the famous *Bayes' rule* using some simple algebra,

$$\mathbb{P}(A \mid B)\,\mathbb{P}(B) = \mathbb{P}(A, B) = \mathbb{P}(B \mid A)\,\mathbb{P}(A), \tag{2.25}$$

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\,\mathbb{P}(A)}{\mathbb{P}(B)}. \tag{2.26}$$

Let $\bigsqcup_{i=1}^{n} A_i = \Omega$ and X be arbitrary, then we can write X in the following form

$$X = \bigsqcup_{i=1}^{n} X \cap A_i \tag{2.27}$$

sum rule

and consequently through application of Equation (2.10) and Equation (2.23) we obtain the so-called *sum rule*

$$\mathbb{P}(X) = \sum_{i=1}^{n} \mathbb{P}(A_i)\,\mathbb{P}(X \mid A_i). \tag{2.28}$$

By applying the sum-rule, we can rewrite the Bayes' rule in terms of a so-called prior probability of A_i, i.e. $\mathbb{P}(A_i)$, and the likelihood $\mathbb{P}(X \mid A_i)$ of X provided the hypothesis A_i. The resulting conditional probability

$$\mathbb{P}(A_i \mid X) = \frac{\mathbb{P}(A_i)\,\mathbb{P}(X \mid A_i)}{\sum_{j=1}^{n} \mathbb{P}(X, A_j)}\,, \tag{2.29}$$

is said to be the posterior probability of A_i. Posterior probabilities play an important role in probabilistic machine learning and, depending on the parametric form of the prior and the likelihood, might have an analytic form.

posterior probability

Note that all of the above derivations are also true for conditional probabilities, i.e. the conditional probability measure $\mathbb{P}(A \mid B)$ for a fixed B is again a probability measure.

Definition 2.10 (independence)**.** *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, two events $A \in \mathcal{A}$ and $B \in \mathcal{A}$ are said to be independent if*

independence

$$\mathbb{P}(A, B) = \mathbb{P}(A)\,\mathbb{P}(B)\,, \tag{2.30}$$

which we denote as $A \perp\!\!\!\perp B$.

This notion of independence can be generalised to some family of events $\{A_i\}_{i \in I}$, for which we say that the events $A_i \in \mathcal{A}$ are independent if

$$\mathbb{P}\left(\bigcap_i A_i\right) = \prod_i \mathbb{P}(A_i)\,. \tag{2.31}$$

Example 2.6. *A simple example of independence is to consider any $A \in \mathcal{A}$ and the whole sample space Ω. Clearly, for any A we have $\mathbb{P}(A, \Omega) = \mathbb{P}(A) = \mathbb{P}(A)\,\underbrace{\mathbb{P}(\Omega)}_{=1}$, thus any couple A, Ω is independent.*

Last but not least, we will review the notion of *conditional independence*, which is central in mixture models and sum-product network.

Definition 2.11 (conditional independence)**.** *Given events A, B, C in a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, the events A and B are said to be conditionally independent given C if*

conditional independence

$$\mathbb{P}(A, B \mid C) = \mathbb{P}(A \mid C)\,\mathbb{P}(B \mid C)\,, \tag{2.32}$$

which we denote as $(A \perp\!\!\!\perp B) \mid C$.

2.2.1 Random Variables

A *random variable* can be understood as a measurable function from a probability space to a measurable space. More formally, we define a random variable as follows.

Definition 2.12 (random variable)**.** *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space then the measurable function $X \colon (\Omega, \mathcal{A}) \to (E, \mathcal{E})$ is said to be a random variable (RV). The push-forward measure*

random variable

$$P_X(B) \coloneqq \mathbb{P}(X^{-1}(B)) = \mathbb{P}(X \in B)\,, \tag{2.33}$$

distribution

with $B \in \mathcal{E}$ is its distribution.

Note that $P_X(B)$ fulfils all of our required properties of a probability measure. In many cases X is assumed to be real-valued, thus, the definition above often simplifies to the case in which $E = \mathbb{R}$ and $\mathcal{E} = \mathcal{B}(\mathbb{R})$. We will, therefore, refer to a real-valued RV simply as RV.

Lemma 2.1. *For any RV X, P_X is a probability measure on $\mathcal{B}(\mathbb{R})$. Moreover, for any probability measure μ on $\mathcal{B}(\mathbb{R})$, there exists a probability space and a RV on it s.t. $P_X = \mu$.*

We refer to Kolmogoroff [54] for a proof of Lemma 2.1. Depending on the set of possible numerical values a RV can take, we call a RV discrete or continuous. For this purpose, we denote $\mathrm{val}(X) \coloneqq \{x \in \mathbb{R} \,|\, x = X(\omega), \omega \in \Omega\}$ to be the range of X, i.e. the image of X under Ω. Note that RVs can also be a mixed type, i.e. consisting of a continuous and a discrete part, or singular [59].

Definition 2.13 (discrete random variable)**.** *If $\mathrm{val}(\Omega)$ is countable, an RV X is said to be discrete and we use $p_X(x) \coloneqq P_X(\{x\}) = P(X = x)$ for all $x \in \mathrm{val}(X)$ to denote its probability mass function (PMF).*

probability mass function

cumulative distribution function

Definition 2.14 (cumulative distribution function)**.** *The cumulative distribution function (CDF) of an RV X on $(\Omega, \mathcal{A}, \mathbb{P})$ is defined as*

$$F_X(x) = P_X(X \leq x) \coloneqq P_X(\{\omega \,|\, X(\omega) \leq x\}), \quad \forall x \in \mathrm{val}(X). \tag{2.34}$$

In case of a discrete RV, the CDF is piece-wise constant and can be written as a finite sum, i.e.

$$F_X(x) = \sum_{x' \in \mathrm{val}(X)} p_X(x') \mathbb{1}_{\{x' \leq x\}}, \tag{2.35}$$

indicator function

where $\mathbb{1}_{\{x' \leq x\}}$ is an *indicator function*, which is one if $x' \leq x$ and zero otherwise.

Definition 2.15 (continuous random variable)**.** *Suppose $\mathbb{P}$ is absolute continuous[10] w.r.t. μ on $(\Omega, \mathcal{A})$. Then, if $\mathrm{val}(X)$ is uncountable, and if by the Radon-Nikodym theorem[11] there exists a function $p \colon \mathbb{R} \to [0, \infty)$ s.t.,*

$$P_X(B) = \int_B p_X(x) \,\mathrm{d}x, \quad B \in \mathcal{B}(\mathbb{R}), \tag{2.36}$$

probability density function

then we say that X is a continuous RV and $p_X(x)$ is its probability density function (PDF) or Radon–Nikodym derivative. Note that we can also define the PDF of a continuous RV over its CDF, i.e.

$$F_X(x) = \int_{-\infty}^{x} p_X(x') \,\mathrm{d}x', \tag{2.37}$$

assuming that F_X is continuous everywhere.

[10] We call μ absolute continuous w.r.t. a second measure ν, both defined on the same measurable space $(\Omega, \mathcal{A})$, if $\mu(A) = 0 \Rightarrow \nu(A) = 0$ for $A \in \mathcal{A}$. [53, p. 230]

[11] Given two measures μ and ν on $(\Omega, \mathcal{A})$ and let μ be σ-finite, then if μ is absolute continuous w.r.t. ν we can write ν as $\nu(A) = \int_A f(x) \mu(\mathrm{d}x)$ with some a.e. unique measurable function f with $f(x) \geq 0$. [53, p. 230]

Throughout the rest of this thesis, we will drop the subscript $_X$ when using the PDF or PMF or CDF of a RV X if the dependence is clear from the context.

Introducing random variables as a measurable function has certain benefits, e.g. moments can be defined in terms of functionals from the space of RVs to $\mathbb{R}_+ \cup \{\infty\}$ and can, therefore, be analysed using tools from functional analysis.

expected value

Definition 2.16 (expected value)**.** *The expected value or expectation of a random variable X on $(\Omega, \mathcal{A}, \mathbb{P})$ is its integral over Ω, i.e.*

$$\mathbb{E}[X] \coloneqq \int_{\omega \in \Omega} X(\omega) \, \mathrm{d}\mathbb{P}(\omega) = \int_{x \in \mathbb{R}} x \, p(x) \, \mathrm{d}x \,. \tag{2.38}$$

If X is discrete, then the integral simplifies to a finite sum, i.e.

$$\mathbb{E}[X] \coloneqq \sum_{x \in \mathrm{val}(X)} x \, p(x) \,. \tag{2.39}$$

variance

Definition 2.17 (variance)**.** *The variance of an RV X on $(\Omega, \mathcal{A}, \mathbb{P})$ is the second moment of the centered RV $(X - \mathbb{E}[X])$, i.e.*

$$\mathit{Var}[X] \coloneqq \mathbb{E}[(X - \mathbb{E}[X])^2] = \int_{\omega \in \Omega} (X(\omega) - \mathbb{E}[X])^2 \, \mathrm{d}\mathbb{P}(\omega) \,, \tag{2.40}$$

which again simplifies to a finite sum in case X is discrete.

Note that moments exist only for discrete or integrable RVs, Cauchy distributed RVs are a well-known example for a non-integrable case. We will now extend the previously introduced rules of probability as well as the concepts of conditional probabilities, independence, and conditional independence to RVs. In particular, we can obtain the conditional distribution $p_{X|Y}$ for continuous RVs X and Y by use of Lebesgue's dominated convergence theorem[12], i.e.

$$p_{X|Y}(x \,|\, y) = \frac{p_{X,Y}(x, y)}{p_Y(y)} \,, \tag{2.41}$$

with $p_Y(y) > 0$. A similar formulation can be found for discrete RVs.

Consequently, we can denote the Bayes' rule of probability distributions as

$$p_{X|Y}(x \,|\, y) = \frac{p_X(x) \, p_{Y|X}(y \,|\, x)}{p_Y(y)} \,, \tag{2.42}$$

where $p_Y(y) > 0$. Note that we can obtain the product-rule of probability from Equation (2.41) and Equation (2.23).

independent random variables

Definition 2.18 (independent random variables)**.** *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and let $X_1, \ldots, X_D$ be RVs on $\mathcal{A}$. Then the RVs*

[12] Lebesgue's dominated convergence theorem provides sufficient conditions for the convergence of the expected value of a RV and can be used to show that the conditional probability distribution of continuous RVs converges to Equation (2.41)

$X_1, \ldots, X_D$ are said to be independent if,

$$\mathbb{P}(X_i \in B_i \,|\, i \leq D) = \prod_{i=1}^{D} \mathbb{P}(X_i \in B_i)\,, \tag{2.43}$$

with $B_i \in \mathcal{B}(\mathbb{R})$.

Corollary 1. *Following from Definition 2.18, RVs $X_1, \ldots, X_D$ are independent iff*

$$F_{X_1,\ldots,X_D}(x_1, \ldots, x_D) = \prod_{i=1}^{D} F_{X_i}(x_i)\,. \tag{2.44}$$

Note that there is a close relationship between independence of RVs and product measures. In fact, if $\boldsymbol{X} = X_1, \ldots, X_D$ and if all RVs $X_1, \ldots, X_D$ are independent, then the distribution of $\boldsymbol{X}$, i.e. its push-forward measure is a product measure

$$\mathbb{P} \circ \boldsymbol{X}^{-1} = \mathbb{P} \circ {X_1}^{-1} \times \ldots \times \mathbb{P} \circ {X_D}^{-1}\,, \tag{2.45}$$

where the product measure is over the push-forward measures of the independent RVs. This has some interesting implications, e.g. if $X_1, \ldots X_D$ are independent then the computation of moments such as the expected value simplifies due to application of the Fubini-Tonelli theorem. In particular, if $X_1, \ldots, X_D$ are independent we can obtain for some non-negative measurable functions f_i the following equality,

$$\begin{aligned}
\mathbb{E}\left[\prod_{i=1}^{D} f_i(X_i)\right] &= \int_\Omega \prod_{i=1}^{D} f_i(X_i)\,\mathrm{d}\mathbb{P} && (2.46)\\
&= \int_{(\times_i E_i)} \prod_{i=1}^{D} f_i(x_i) p(x_1, \ldots, x_D)\,\mathrm{d}x_1 \ldots\, \mathrm{d}x_D && (2.47)\\
&= \int_{(\times_i E_i)} \prod_{i=1}^{D} f_i(x_i) \underbrace{\bigtimes_{i=1}^{D} \lambda(x_i)}_{\text{product measure}} \mathrm{d}x_1 \ldots\, \mathrm{d}x_D && (2.48)\\
&= \prod_{i=1}^{D} \int_{E_i} f_i(x_i)\lambda(x_i)\,\mathrm{d}x_i && (2.49)\\
&= \prod_{i=1}^{D} \int_{E_i} f_i(x_i) p(x_i)\,\mathrm{d}x_i = \prod_{i=1}^{D} \mathbb{E}[X_i]\,, && (2.50)
\end{aligned}$$

and

$$\mathbb{E}\left[\sum_{i=1}^{D} f_i(X_i)\right] = \sum_{i=1}^{D} \mathbb{E}[f_i(X_i)]\,, \tag{2.51}$$

which can be shown similarly. The same result is true for higher order moments. We say that two RVs X and Y are conditional independent

of RV Z if

conditionally independent random variables

$$F_{X,Y \mid Z=z}(x, y) = F_{X \mid Z=z}(x)\, F_{Y \mid Z=z}(y)\,, \tag{2.52}$$

where $F_{X \mid Z}$ is the conditional CDF, which is given as

$$F_{X \mid Z=z}(x) = \frac{1}{p_Z(z)} \sum_{x' \leq x} p(x', z)\,, \tag{2.53}$$

in the discrete case and as

$$F_{X \mid Z=z}(x) = \frac{1}{p_Z(z)} \int_{-\infty}^{x'} p(x', z)\, \mathrm{d}x\,, \tag{2.54}$$

in the continuous case.

Last but not least, we will extend the notion of RVs to random vectors, i.e. multivariate distributions.

random vector

Definition 2.19 (random vector)**.** *Given a measurable space $(\Omega, \mathcal{A})$, we call the function $\boldsymbol{X} \colon (\Omega, \mathcal{A}) \to (\mathbb{R}^D, \mathcal{B}(\mathbb{R}^D))$ a random vector or a vector-valued random variable if it is measurable w.r.t. $\mathcal{A}$, c.f. Equation* (2.16)*. Note that $\boldsymbol{X}$ is only a random vector, if all its components $X \in \boldsymbol{X}$ are measurable. Thus, a random vector is a D-tuple of RVs.*

Similar to the one-dimensional case, i.e. in case of a single RV, the distribution of a D dimensional random vector $\boldsymbol{X}$ is defined by its push-forward measure $P_{\boldsymbol{X}}(B) \coloneqq \mathbb{P}(\boldsymbol{X}^{-1}(B))$ for $B \in \mathcal{B}(\mathbb{R}^D)$. We call this distribution to be a *joint probability distribution*. As established in Definition 2.18, if all RVs of $\boldsymbol{X}$ are independent then the joint probability distribution of $\boldsymbol{X}$ is a product measure.

joint probability distribution

The distribution of a random vector may be discrete, in the sense that $\mathrm{val}(\boldsymbol{X}) = \bigtimes_{i=1}^{D} \mathrm{val}(X_i)$ is countable. If the distribution $\boldsymbol{X}$ is absolutely continuous with respect to a D-dimensional Lebesgue measure, and consequently all $X \in \boldsymbol{X}$ have a density, the random vector is said to be continuous. As for single RVs, if a random vector is neither continuous nor discrete we say it is of mixed type. We denote an instantiation of a random vector as $\boldsymbol{x} \in \mathrm{val}(\boldsymbol{X})$.

In many scenarios we are interested in only a subset $\tilde{\boldsymbol{X}} \subset \boldsymbol{X}$ of RVs in $\boldsymbol{X}$. Examples for such cases include situations in which we only have an instantiation of $\tilde{\boldsymbol{X}}$, i.e. $\tilde{\boldsymbol{x}}$, and want to *marginalise out* (integrate over) the remaining RVs, which we did not observe, i.e. $\boldsymbol{Z} = \boldsymbol{X} \setminus \tilde{\boldsymbol{X}}$. In such a cases we are interested in the *marginal distribution* of $\tilde{\boldsymbol{X}}$.

marginal distribution

Definition 2.20 (marginal distribution)**.** *Given a random vector $\boldsymbol{X}$ on $(\Omega, \mathcal{A}, \mathbb{P})$, the marginal distribution of $\tilde{\boldsymbol{X}} \subset \boldsymbol{X}$ where $\boldsymbol{Z} = \boldsymbol{X} \setminus \tilde{\boldsymbol{X}}$ is defined for continuous RVs $\boldsymbol{Z}$ as,*

$$p_{\tilde{\boldsymbol{X}}}(\tilde{\boldsymbol{x}}) \coloneqq \int_{z_1} \cdots \int_{z_K} p_{\boldsymbol{X}}(\tilde{\boldsymbol{x}}, z_1, \ldots, z_K)\, \mathrm{d}z_1 \ldots \mathrm{d}z_K\,, \tag{2.55}$$

and simplifies to finite sums in case all RVs in $\boldsymbol{Z}$ are discrete.

2.3 Primer on Graph Theory

In this section we will briefly review the main concepts of graph theory, relevant for the course of this thesis. We refer to Diestel [60] for further details on graph theory.

graph

Definition 2.21 (graph)**.** *A graph is a tuple $\mathcal{G} = (V, E)$ of sets, with V being a set of vertices (or nodes) and the elements of E, with $E \subseteq V \times V$, being the edges in $\mathcal{G}$. We denote the vertex set of $\mathcal{G}$ as $V(\mathcal{G})$ and the edge set as $E(\mathcal{G})$.*

Given a graph $\mathcal{G}$, we call two vertices v_i, v_j *adjacent* if the edge $(v_i, v_j) \in E(\mathcal{G})$. If all vertices in $\mathcal{G}$ are pairwise adjacent, then $\mathcal{G}$ is said to be *complete*. Consequently, we call two v_i, v_j *separate* if they are not adjacent.

complete graph

sub-graph

Definition 2.22 (sub-graph)**.** *If $V' \subseteq V$ and $E' \subseteq E, E' \subseteq V' \times V'$ for $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$, then we call $\mathcal{G}'$ a sub-graph of $\mathcal{G}$ and $\mathcal{G}$ is called the super-graph of $\mathcal{G}'$, denoted as $\mathcal{G}' \subseteq \mathcal{G}$.*

If the sub-graph $\mathcal{G}' \subseteq \mathcal{G}$ contains all edges $(v_i, v_j) \in E(\mathcal{G})$ connecting pairs of vertices in $v_i, v_j \in V(\mathcal{G}')$, then $\mathcal{G}'$ is said to be an induced sub-graph or simply *induced graph*.

induced graph

path

Definition 2.23 (path)**.** *A path or walk $\mathcal{W} = (V, E)$ is a non-empty graph with $V = \{v_0, \ldots, v_k\}$ and $E = \{(v_0, v_1), \ldots, (v_{k-1}, v_k)\}$ and the cardinality of E is called the length of $\mathcal{W}$. For ease of notation, we denote a path as $\mathcal{W} = v_0, \ldots, v_k$.*

cycle

Definition 2.24 (cycle)**.** *Given a path $\mathcal{W} = v_0, \ldots, v_{k-1}$ with $k \geq 3$, then the graph $\mathcal{W} \cup \{(v_{k-1}, v_0)\}$ is called a cycle.*

connected graph

We call a graph *connected* if the graph is non-empty and if any two of its vertices are linked by a path.

tree

Definition 2.25 (tree)**.** *If $\mathcal{G}$ is an (undirected) acyclic connected graph, then $\mathcal{G}$ is said to be a tree.*

So far, we have only considered *undirected* graphs for which edges do not have a direction. Note that we can represent a graph $\mathcal{G}$ using an *adjacency matrix*, that is a matrix $A \in \mathbb{B}^{|V| \times |V|}$ with $A_{i,j} = 1$ if (v_i, v_j) is an edge in $E(\mathcal{G})$ and $A_{i,j} = 0$ otherwise. We say $\mathcal{G}$ is undirected iff $A_{i,j} = 1 \Rightarrow A_{j,i} = 1$, i.e. A is symmetric, and otherwise say that $\mathcal{G}$ is a *directed* graph. Further, we say that a path $\mathcal{W}$ is a directed cycle if $\mathcal{W}$ is a cycle and all edges in $\mathcal{W}$ have the same direction.

adjacency matrix

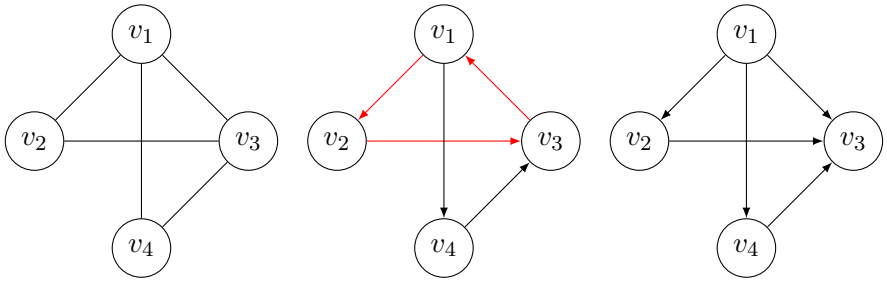
directed graph

Definition 2.26 (directed acyclic graph)**.** *A graph $\mathcal{G}$ is said to be a directed acyclic graph (DAG), if $\mathcal{G}$ is directed and does not contain any directed cycles.*

For a directed graph, we denote the parents of a node v_i, i.e. the set of nodes that feed into v_i, as $\mathrm{par}(v_i) \coloneqq \{v_j \mid A_{j,i} = 1\}$ and the children of v_i as $\mathrm{ch}(v_i) \coloneqq \{v_j \mid A_{i,j} = 1\}$. Further, we say a node is a *root* if it has no parents and a node is said to be a *leaf* if it has no children. If there exists a directed path from v_i to v_j, then v_i is said to be an *ancestor* of v_j and v_j is its *descendant*. Figure 2.1 illustrates an undirected, directed and a directed acyclic graph side-by-side. Note that the arrow direction shows the edge direction in the directed graphs.

root node

leaf node



(a) undirected graph (b) directed graph (c) directed acyclic graph

Figure 2.1: Illustration of an undirected, directed and directed acyclic graph. Directed cycles in 2.1(b) are shown in red.

3 Sum-Product Networks

This chapter provides an introduction to sum-product networks and discusses some recent developments in the field. In particular, we will review generalized sum-product networks, discuss various representations and introduce the foundations of generative and discriminative parameter learning in sum-product networks. Last but not least, we examine implicit acceleration effects of parameter learning in tree-shaped sum-product networks and conclude the chapter with a brief review of existing structure learning techniques.

3.1 Generalized Sum-Product Networks

Sum-product networks (SPN) [8] are a class of probabilistic models that allow exact and efficient computation of many inference task, e.g. marginalisation, conditioning and computation of moments. Besides enjoying preferable inference properties, SPNs can efficiently represent complex high-dimensional probability distributions. Thus, SPNs enjoy increasing attention in the machine learning community and have been successfully applied to various modelling tasks, e.g. [61]–[66].

SPNs arose from the foundational work on efficient inference in Bayesian networks [67]. In particular, Darwiche [67] showed that the network polynomial of a Bayesian network over a discrete random vector can be efficiently represented using an arithmetic circuit.

We will now briefly review the notion of a *network polynomial* and an *arithmetic circuit*.

network polynomial

Definition 3.1 (network polynomial)**.** *Let $\phi(\boldsymbol{x}) \geq 0$ be an unnormalised distribution over a discrete random vector $\boldsymbol{X}$. Further, let $\boldsymbol{x} \in \mathrm{val}(\boldsymbol{X})$ denotes an instantiation of $\boldsymbol{X}$ and $\boldsymbol{x}[X]$ be the projection[13] of $\boldsymbol{x}$ onto $X \in \boldsymbol{X}$ and let $\lambda_{X=\boldsymbol{x}[X]}$ denote an indicator that is one if $\boldsymbol{x}[X] = x$ and zero otherwise. Then the function f_ϕ is called a network polynomial of ϕ if it is defined as*

$$f_\phi(\boldsymbol{\lambda}) = \sum_{\boldsymbol{x} \in \mathrm{val}(\boldsymbol{X})} \phi(\boldsymbol{x}) \prod_{X \in \boldsymbol{X}} \lambda_{X=\boldsymbol{x}[X]} \,. \tag{3.1}$$

The definition of a network polynomial can be generalized to arbitrary unnormalised distributions, i.e. unnormalised distributions over finitely many arbitrary random vectors [8], [68]. We can compute the probability of any instantiation of $\boldsymbol{X}$ by setting all indicators consistent with the

[13] The projection of $\boldsymbol{x}$ onto $X_i \subset \{X_1, \ldots, X_D\}$ with $1 \leq i \leq D$ is defined as the value of X_i given $\boldsymbol{x}$, i.e. $\boldsymbol{x}[X_i] = \{x_j : j = i\}$.

instantiation to one and all others to zero. Further, the *normalisation constant* or potential function can be computed by setting all indicators to one. Note that the number of terms in the network polynomial grows exponential with the number of RVs, making a direct use difficult in practice [69], [70]. However, we can compactly represent a network polynomial using a so-called *arithmetic circuit*.

arithmetic circuit

Definition 3.2 (arithmetic circuit)**.** *An arithmetic circuit (AC) [67], [70] is a rooted directed acyclic graph, whose internal nodes are equipped with arithmetic operations, e.g. addition or multiplication, and whose leaf nodes are numeric inputs.*

Representing a network polynomial using an AC avoids representing redundancies and can be exponentially more compact [69]. We refer to [67], [68], [70] for an in-depth discussion on network polynomials and arithmetic circuits.

We will now introduce SPNs in terms of a generalized SPN, as we will utilise the fact that generalised SPNs can have arbitrary input distributions throughout this thesis.

sum-product network

Definition 3.3 (sum-product network)**.** *A sum-product network (SPN), denoted as $\mathcal{S}$, over RVs $\boldsymbol{X}$ is a connected rooted directed acyclic graph with arbitrary (tractable) distributions as leaves ($\mathsf{L} \in V(\mathcal{S})$) and whose internal nodes are sum nodes ($\mathsf{S} \in V(\mathcal{S})$) and product nodes ($\mathsf{P} \in V(\mathcal{S})$). Sum nodes compute a weighted sum of their children, i.e. $\mathsf{S}(\boldsymbol{x}) = \sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} \mathsf{N}(\boldsymbol{x})$, where each edge emanating from a sum node has non-negative weight, i.e. $w_{\mathsf{S},\mathsf{N}} \geq 0$, and we say that S is normalised if $\sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{N})} w_{\mathsf{S},\mathsf{N}} = 1$ [71]. We denote the set of weights for a sum node S as $\boldsymbol{w}_{\mathsf{S}}$ and use $\boldsymbol{w}$ to denote all weights in an SPN. The value of a product node is calculated by taking the product of its children, i.e. $\mathsf{P}(\boldsymbol{x}) = \prod_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} \mathsf{N}(\boldsymbol{x})$. Finally, the value of a leaf node is calculated by evaluating its respective probability distribution on a subset of $\boldsymbol{X}_{\mathsf{L}} \subseteq \boldsymbol{X}$, i.e. its scope, denoted as $\mathsf{L}(\boldsymbol{x}) = p(\boldsymbol{x}_{\mathsf{L}} \mid \theta_{\mathsf{L}})$. We use θ_{L} to denote the parameters of the distribution at leaf L and refer to the set of all leaf node parameters using $\boldsymbol{\theta}$.*

scope

Throughout this thesis, we denote the scope of a node in an SPN over $\boldsymbol{X}$ using $\psi(\mathsf{N}) \subseteq \boldsymbol{X}$. Note that this notation is different to prior work, which commonly uses scope(N) to denote the scope. Further, we use $\mathbf{N}$ to denote all nodes, $\mathbf{S}$ to denote all sum nodes, $\mathbf{P}$ to denote all product nodes, and $\mathbf{L}$ to denote all leaf nodes in an SPN. Figure 3.1 illustrates an SPN over a three-dimensional random vector.

An SPN is said to be *normalised* if all sum nodes are normalised. Note that w.l.o.g. we will assume every SPN to be normalised [71]. The scope of internal nodes, i.e. sums and products, is defined by the union of the scopes of their children. Consequently, the scope and the value of the SPN is determined by the root node of the SPN. Note that any sub-graph of an SPN is itself an SPN. In some cases it is sufficient to consider only tree-structured SPNs, which we denote as a sum-product trees (SPTs). Even though SPTs are frequently used, e.g. [36], [39], [72], they are a less compact representation as they scale exponentially in the tree depth.

normalised

sum-product tree

Definition 3.4 (sum-product tree)**.** *A sum-product tree (SPT) is an SPN whose nodes have at most one parent.*

To ensure that an SPN is a sound probability distribution and guarantees efficient probabilistic inference, we require that an SPN is *complete*[14] and *consistent* or *decomposable* [8], [71]. Completeness and consistency or decomposability are essential to render many inference scenarios tractable in SPNs.

complete

Definition 3.5 (complete)**.** *An SPN is said to be complete iff all children under each sum node have the same scope.*

consistent

Definition 3.6 (consistent)**.** *An SPN is said to be consistent iff each two distinct children under each product node do not have different indicators for the same RV in their ancestors.*

Alternative to consistency, SPNs are often assumed to be *decomposable* [67]. This notion originates from the work on arithmetic circuits and has later been adopted to SPNs by Peharz, Tschiatschek, Pernkopf *et al.* [71].

decomposable

Definition 3.7 (decomposable)**.** *An SPN is said to be decomposable iff all children under each product node have mutually disjoint scopes.*

Decomposability is a stronger condition than consistency and implies consistency but not vice-versa. Further, Peharz, Tschiatschek, Pernkopf *et al.* [71] showed that complete and consistent SPNs can only be moderately more compact than complete and decomposable SPNs. Intuitively, we can understand a decomposable product node to represent an independence assumption, i.e. the probability distributions of the children of the product are independent from each other. Sum nodes in SPNs, on the other hand, replace the enforced independence assumptions with conditional independence, c.f. Definition 2.11. Because decomposable SPNs are often easier to construct and have an intuitive interpretation, recent approaches often deal with complete and decomposable SPNs instead of complete and consistent SPNs. Therefore, in the context of this thesis, we always assume that SPNs are complete and decomposable and merely refer to them as SPNs. Note that any complete and decomposable SPN computes the network polynomial of some (unnormalised) distribution [68] and can be understood as a special instantiation of an AC. Further note, that an SPN can also be *selective* or *deterministic*, we refer to Peharz [68] for details.

In SPNs, arbitrary marginalisation tasks reduce to marginalisation at the leaves, i.e. they simplify to tractable marginalisation over a (small) subset of $\boldsymbol{X}$ [71]. In fact, we can compute arbitrary marginals exactly (provided that the leaf nodes allow exact computations) in *linear time* in size of the SPN. For this let us consider the following marginalisation task

$$\int_{x_1 \in X_1} \cdots \int_{x_D \in X_D} \mathcal{S}(x_1, \ldots, x_D) \, \mathrm{d}x_1 \cdots \mathrm{d}x_D \tag{3.2}$$

in which integrals simplify to finite sums in case of discrete RVs. Because of completeness, we can swap the integrals (or finite sums in case of

[14] In the context of arithmetic circuits, completeness is often called smoothness.

discrete RVs) and the summation at sum nodes, i.e.

$$\int_{\boldsymbol{x}\in\boldsymbol{X}} \mathsf{S}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = \sum_{\mathsf{N}\in\mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} \int_{\boldsymbol{x}\in\boldsymbol{X}} \mathsf{N}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x}\,, \tag{3.3}$$

and at decomposable product nodes, the integrals distribute to the children due independence of their probability distributions, i.e.

$$\int_{\boldsymbol{x}\in\boldsymbol{X}} \mathsf{P}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = \prod_{\mathsf{N}\in\mathrm{ch}(\mathsf{P})} \int_{\hat{\boldsymbol{x}}\in\psi(\mathsf{N})} \mathsf{N}(\hat{\boldsymbol{x}})\ \mathrm{d}\hat{\boldsymbol{x}}\,. \tag{3.4}$$

Therefore, we only have to perform marginalisation at the leaves followed by a bottom-up evaluation of the SPN. Computation of moments and conditionals can be tackled in the same way.

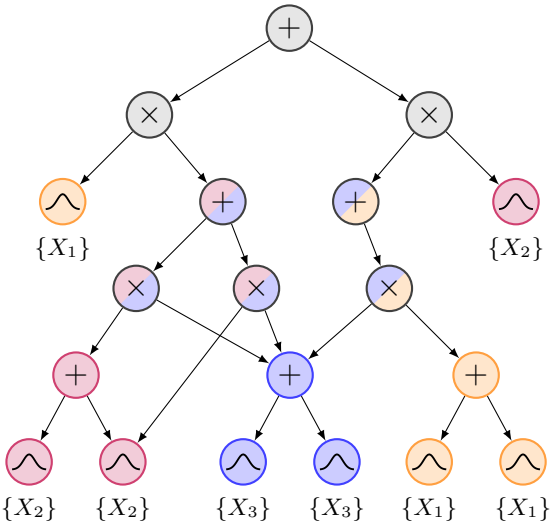


Figure 3.1: Illustration of a complete and decomposable SPN over $\boldsymbol{X} = \{X_1, X_2, X_3\}$ with colors indicating the scope of a node, i.e. gray indicates a full scope.

3.1.1 Induced Trees

As shown in multiple works [19], [73] an SPN can be interpreted as a deep structured mixture model with exponentially many components. For this purpose, Zhao, Adel, Gordon *et al.* [19] introduced the notion of so-called *induced trees*, which are directed sub-trees in an SPN.

Definition 3.8 (Induced Tree [19])**.** *Given an SPN $\mathcal{S}$, an SPT $\mathcal{T} \subseteq \mathcal{S}$ is called an induced tree if:*

- *The root of $\mathcal{S}$ is in $\mathcal{T}$,*
- *For each sum node $\mathsf{S} \in V(\mathcal{T})$ exactly one child of S in $\mathcal{S}$ is in $\mathcal{T}$, and the corresponding edge is in $E(\mathcal{T})$, and*
- *For each product node $\mathsf{P} \in V(\mathcal{T})$ all children of P in $\mathcal{S}$ are in $\mathcal{T}$, and the corresponding edges are in $E(\mathcal{T})$.*

Figure 3.2 illustrates an induced tree of the SPN shown in Figure 3.1. The distribution of any SPN, denoted as $\mathcal{S}(\boldsymbol{x})$, can be written as a mixture whose components correspond to induced trees, i.e.

$$\mathcal{S}(\boldsymbol{x}) = \sum_{\mathcal{T}} \underbrace{\prod_{(\mathsf{S},\mathsf{N})\in E(\mathcal{T})} w_{\mathsf{S},\mathsf{N}}}_{=p(\mathcal{T})} \prod_{\mathsf{L}\in V(\mathcal{T})} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}})\,. \tag{3.5}$$

To see this, let us introduce a latent variable for each observation $\boldsymbol{x}_i$ and each $\mathsf{S} \in \mathbf{S}$ denoted as $Z_{\mathsf{S},i}$. Each latent variable has as many states as S has children and is distributed according to a categorical distribution given by the weights at S, i.e. $z_{\mathsf{S},i} \sim \boldsymbol{w}_{\mathsf{S}}$. Now let us define a function $T(\boldsymbol{z})$, which assigns to each value $\boldsymbol{z}$ of $Z = \{Z_{\mathsf{S},i}\}_{\mathsf{S}\in\mathcal{S}}$ the *induced tree determined by* $\boldsymbol{z}$. Note that the function $T(\boldsymbol{z})$ is surjective, but not injective, and thus, $T(\boldsymbol{z})$ is not invertible. However, it is possible to partially recover $\boldsymbol{z}$. In fact, any $\mathcal{T}$ splits the set of all sum nodes $\mathbf{S}$ into two disjoint sets, with the first being the set of sum nodes $\mathbf{S}_{\mathcal{T}}$ contained in $\mathcal{T}$, and the second being the set of sum nodes $\bar{\mathbf{S}}_{\mathcal{T}}$ that are not in $\mathcal{T}$. For any $\mathcal{T}$, we can clearly identify the state z_{S} for any $\mathsf{S} \in \mathbf{S}_{\mathcal{T}}$, as it corresponds to the unique child of S in $\mathcal{T}$. However, the state of any $\mathsf{S} \in \bar{\mathbf{S}}_{\mathcal{T}}$ is arbitrary.

Let use now define the conditional probability distribution given a latent assignment $\boldsymbol{z}$, i.e. $p(\boldsymbol{x} \,|\, \boldsymbol{z}) = \prod_{\mathsf{L}\in T(\boldsymbol{z})} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}})$. Further, we denote the prior probability of a latent assignment as $p(\boldsymbol{z}) = \prod_{\mathsf{S}\in V(\mathcal{S})} w_{\mathsf{S},z_{\mathsf{S}}}$, where $w_{\mathsf{S},z_{\mathsf{S}}}$ denotes the sum-weight indicated by z_{S}. By marginalising out $\boldsymbol{Z}$ from the joint $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \,|\, \boldsymbol{z})\, p(\boldsymbol{z})$ we obtain

$$\sum_{\boldsymbol{z}} \prod_{\mathsf{S}\in\mathbf{S}} w_{\mathsf{S},z_{\mathsf{S}}} \prod_{\mathsf{L}\in V(T(\boldsymbol{z}))} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}}) \tag{3.6}$$

$$= \sum_{\mathcal{T}} \sum_{\boldsymbol{z}\in T^{-1}(\mathcal{T})} \prod_{\mathsf{S}\in\mathbf{S}} w_{\mathsf{S},z_{\mathsf{S}}} \prod_{\mathsf{L}\in V(T(\boldsymbol{z}))} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}}) \tag{3.7}$$

$$= \sum_{\mathcal{T}} \prod_{(\mathsf{S},\mathsf{N})\in E(\mathcal{T})} w_{\mathsf{S},\mathsf{N}} \prod_{\mathsf{L}\in V(\mathcal{T})} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}}) \underbrace{\left(\sum_{\bar{\boldsymbol{z}}} \prod_{\mathsf{S}\in\bar{\mathbf{S}}_{\mathcal{T}}} w_{\mathsf{S},\bar{z}_{\mathsf{S}}} \right)}_{=1} \tag{3.8}$$

$$= \sum_{\mathcal{T}} \prod_{(\mathsf{S},\mathsf{N})\in E(\mathcal{T})} w_{\mathsf{S},\mathsf{N}} \prod_{\mathsf{L}\in V(\mathcal{T})} p(\boldsymbol{x}_{\mathsf{L}} \,|\, \theta_{\mathsf{L}}) = \mathcal{S}(\boldsymbol{x})\,. \tag{3.9}$$

We have split the sum over all $\boldsymbol{z}$ into a double sum in Equation (3.7). The first one sums over all induced trees $\mathcal{T}$ and the latter sums over all $\boldsymbol{z} \in T^{-1}(\mathcal{T})$, i.e. the set of all $\boldsymbol{z}$ for which $T(\boldsymbol{z}) = \mathcal{T}$. As mentioned, the set $\boldsymbol{z} \in T^{-1}(\mathcal{T})$ is the union of all unique z-assignments that can be exactly identified, i.e. the assignment for all $\mathsf{S} \in \mathbf{S}_{\mathcal{T}}$, and the set of all possible assignments at each $\mathsf{S} \in \bar{\mathbf{S}}_{\mathcal{T}}$. Note that, under the assumption that the SPN is normalised, the summation over the second set of assignments evaluates to one, c.f. Equation (3.8). If the network is not normalised, we can re-normalise the SPN without changing its distribution as shown in Peharz, Gens, Pernkopf *et al.* [16]. Therefore, we can obtain the mixture formulation shown in Equation (3.5) for any SPN.

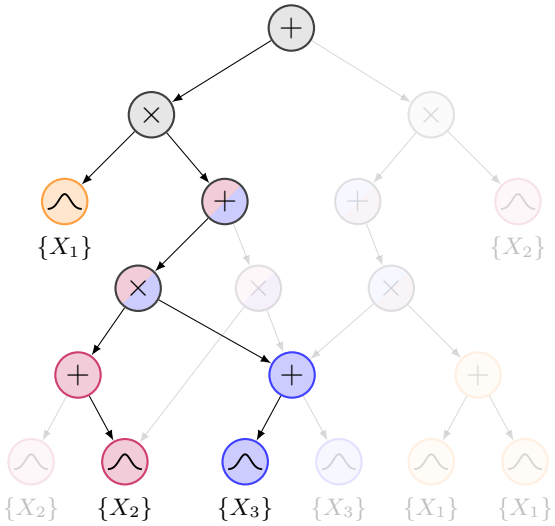


Figure 3.2: Illustration of an induced tree of an SPN over $\boldsymbol{X} = \{X_1, X_2, X_3\}$ with colors indicating the scope of a node.

3.1.2 Probability Measure of Sum-Product-Networks

In addition to the introduced concepts, we can take a step back and consider the probability measure induced by an SPN. For this, consider an SPN with fixed structure and parameters over $\boldsymbol{X}$ and let us define the probability measure of $\mathcal{S}$ bottom up.

Leaf node Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and let N be a leaf node. Further, let $(E, \mathcal{E})$ be a measurable space and $\boldsymbol{X}_\mathsf{N} \coloneqq \psi(\mathsf{N}) \subseteq \boldsymbol{X}$ be the scope of N, i.e. the measurable function $\boldsymbol{X}_\mathsf{N} \colon \Omega \mapsto E$. Then the induced push-forward measure $P_{\boldsymbol{X}_\mathsf{N}}(B)$ with $B \in \mathcal{E}$ is its distribution. Note that with some misuse of notation, $\boldsymbol{X}_\mathsf{N}$ denotes both: a single RV and a random vector; depending on the distribution allocated at N.

Product node Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and let N be a decomposable product node with scope $\boldsymbol{X}_\mathsf{N} = \bigcup_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} \boldsymbol{X}_{\mathsf{N}'}$. Further, let $(E_{\mathsf{N}'}, \mathcal{E}_{\mathsf{N}'})$ with $\mathsf{N}' \in \mathrm{ch}(\mathsf{N})$ be the measurable spaces of the children of N. Then the measurable space of N is a product-space and the distribution of $\boldsymbol{X}_\mathsf{N} \colon \Omega \mapsto E$ is a product measure, i.e.

$$P_{\boldsymbol{X}_\mathsf{N}} \left(\times_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} P_{\boldsymbol{X}_{\mathsf{N}'}} \right) (B) = \prod_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} P_{\boldsymbol{X}_{\mathsf{N}'}}(B_{\mathsf{N}'}) \,, \tag{3.10}$$

with $B \in \otimes_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} \mathcal{E}_{\mathsf{N}'}$ and $B_{\mathsf{N}'} \in \mathcal{E}_{\mathsf{N}'}$ iff the distributions of its children are independent. Because decomposability ensures that the probability distributions of the children under a product node are mutually independent, the probability measure under a product node is a product measure, c.f. Definition 2.18.

Sum node Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and let N be a complete and normalised sum node with scope $\boldsymbol{X}_\mathsf{N} = \bigcup_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} \boldsymbol{X}_{\mathsf{N}'}$ and weights $\boldsymbol{w}_\mathsf{N} = \{w_{\mathsf{N},\mathsf{N}'}\}_{\mathsf{N}' \in \mathrm{ch}\,\mathsf{N}}$. Then the probability measure of N is a convex combination of the probability measures of its children and so is the

distribution of $\boldsymbol{X}_{\mathsf{N}} \colon \Omega \mapsto E$, i.e.

$$P_{\boldsymbol{X}_{\mathsf{N}}}(B) = \sum_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} w_{\mathsf{N},\mathsf{N}'} P_{\boldsymbol{X}_{\mathsf{N}'}}(B)\,, \tag{3.11}$$

with $B \in \mathcal{E}$ and $\mathcal{E}$ being a σ-algebra on E.

By recursively applying the arguments presented above, we see that the probability measure of an SPN is a recursive representation of a convex combination of product measures. Note that SPNs and other probabilistic circuits can also be described in terms of an algebraic commutative semiring, i.e. a semiring in which multiplication is commutative [74], [75]. We refer to Friesen and Domingos [75] for further details.

3.2 Representations of Sum-Product Networks

3.2.1 Computational Graphs & Scope Functions

An SPN can be defined in terms of a *computational graph* ($\mathcal{G}$) and a so-called *scope-function* (ψ). Note that we are deliberately overload the symbol ψ to emphasise the connection to the scope of a node. In particular, we can say that an SPN over RVs $\boldsymbol{X}$ is defined as a 4-tuple $\mathcal{S} = (\mathcal{G}, \psi, \boldsymbol{w}, \boldsymbol{\theta})$, where $\boldsymbol{w} = \{\boldsymbol{w}_{\mathsf{S}}\}_{\mathsf{S} \in \mathbf{S}}$ with $w \geq 0\,, \forall w \in \boldsymbol{w}_{\mathsf{S}}$ denotes the set of all weights in the SPN and $\boldsymbol{\theta} = \{\theta_{\mathsf{L}}\}_{\mathsf{L} \in \mathbf{L}}$ are the parameters at the leaves.

computational graph

Definition 3.9 (computational graph)**.** *A computational graph is a rooted directed acyclic graph $\mathcal{G} = (V, E)$ whose internal nodes are sum and product nodes and whose leaves are equipped with arbitrary probability distributions.*

It is evident from Definition 3.9 that a computational graph is simply an arithmetic circuit with arbitrary probability distributions as leaves. Note that such a graph does not have to fulfill any of the previously mentioned conditions, i.e. completeness, consistency and decomposability. Thus, computational graphs can be easily constructed and can potentially be reused for different SPNs. Figure 3.3 illustrates a simple computational graph.

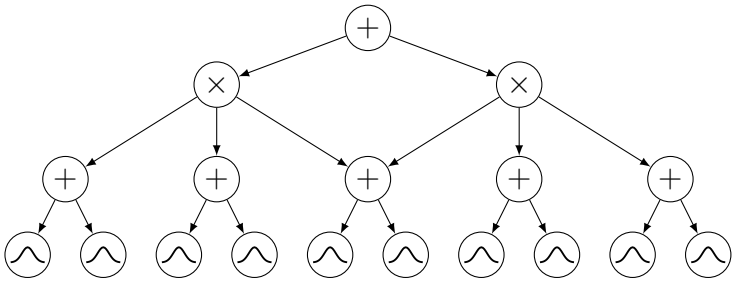


Figure 3.3: Illustration of a computational graph containing sum, product and leaf nodes.

To obtain a complete and decomposable SPN, we define a so-called *scope-function*, which maps the set of nodes in a computational graph to the set of possible sub-scopes of a random vector $\boldsymbol{X}$.

scope-function

Definition 3.10 (scope-function)**.** *Given a computational graph $\mathcal{G}$ and a set of RVs $\boldsymbol{X}$, a scope-function is a function mapping from the set of nodes $\mathbf{N} = V(\mathcal{G})$ in the graph to the power-set $\mathcal{P}(\boldsymbol{X})$, i.e. $\psi\colon \mathbf{N} \mapsto \mathcal{P}(\boldsymbol{X})$. It has the following properties:*

1. *If N is the root node, then $\psi(\mathsf{N}) = \boldsymbol{X}$.*
2. *If N is a sum or product node, then $\psi(\mathsf{N}) = \bigcup_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} \psi(\mathsf{N}')$.*
3. *For each $\mathsf{P} \in \mathbf{P}$ we have $\forall \mathsf{N}, \mathsf{N}' \in \mathrm{ch}(\mathsf{P})\colon \psi(\mathsf{N}) \cap \psi(\mathsf{N}') = \emptyset$ (decomposability).*
4. *For each $\mathsf{S} \in \mathbf{S}$ we have $\forall \mathsf{N}, \mathsf{N}' \in \mathrm{ch}(\mathsf{S})\colon \psi(\mathsf{N}) = \psi(\mathsf{N}')$ (completeness).*

Lemma 3.1. *For any computational graph $\mathcal{G}$ and any set of RVs $\boldsymbol{X}$, there exists a scope-function.*

Proof. We can see that this is true by considering a scope-function ψ, which indexes a subgraph in $\mathcal{G}$ that is a (hierarchical) mixture. For this purpose, we construct a scope-function that ensures that each product node has at most one child with $\psi(\mathsf{N}) \neq \emptyset$. We can find such as scope-function by traversing $\mathcal{G}$ top-down. If the node N is a sum node, we traverse to all children of N. If the node N is a product node, we can have the following scenarios: i) all children of N have only one parent, ii) all children of N have multiple parents or iii) some children have multiple parents. In scenario i), we can freely chose any child to traverse to. In scenario ii-iii), we chose the child to traverse to such that our choice is consistent with the choice of all parents of the child and traverse down to the child. Finally, we set $\psi(\mathsf{N}) = \emptyset$ for each node we did not traverse down to and $\psi(\mathsf{N}) = \boldsymbol{X}$ for each node that we have visited. This construction clearly ensures that ψ fulfils all the required properties of a proper scope-function and indexes a subgraph in $\mathcal{G}$, which is a proper SPN that can be reduced to a (hierarchical) mixture model. □

Given a computational graph $\mathcal{G}$, a D-dimensional random vector $\boldsymbol{X}$ and a scope-function from $V(\mathcal{G})$ to $\mathcal{P}(\boldsymbol{X})$, each node N in $\mathcal{G}$ represents a distribution over the random variables $\psi(\mathsf{N})$. In particular, each leaf L computes a pre-specified distribution over its scope $\psi(\mathsf{L})$. If $\psi(\mathsf{N}) = \emptyset$, we set $\mathsf{N} \equiv 1$. In general, we assume that each leaf L is parametrised by θ_{L}, and that θ_{L} represents a distribution *for any possible choice of* $\psi(\mathsf{L})$. Sum and product nodes respectively compute a weighted mixture distribution and a product distribution as previously described. Note that the definition using a computational graph and a scope-function is quite different from prior art: previously, leaves were defined to be distributions over a fixed predefined scope, while now each leaf defines a distribution over all possible scopes at all times. In practice, the scope-function will either be defined once, simplifying to the notion in Definition 3.3, or can be learned, which we will discuss in detail in Chapter 5. Figure 3.4 illustrates an SPN obtained by applying a scope-function on the computational graph illustrated in Figure 3.3.

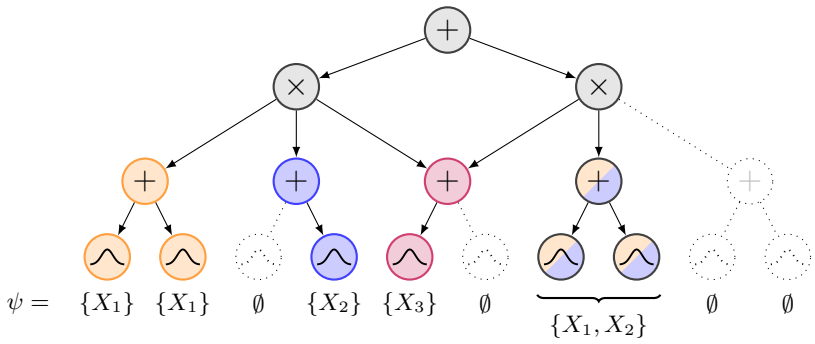


Figure 3.4: Illustration of an SPN defined using a computational graph and a scope-function ψ on a random vector $\boldsymbol{X} = \{X_1, X_2, X_3\}$. Nodes with empty scope are shown in dotted lines.

In contrast to Definition 3.3, structural conditions, e.g. completeness and decomposability, are now expressed only through the scope-function. Note that we can interpret applying a scope-function as masking nodes in the computational graph and selecting a sub-graph that fulfils the mentioned structural conditions. Therefore, we can express various effective structures of SPNs by simply applying different scope-functions on the same computational graph.

3.2.2 Region Graphs

Instead of defining the structure of an SPN directly, either by means of an entangled representation or by using a computational graph and a scope-function, we can define an SPN through a so-called region graph [76]. Region graphs are a natural representation of vectorised SPNs and allow more efficient computation through hardware-accelerated dense linear algebra. Therefore, this representation has gained increasing attention and has been successfully used in various real-world applications, e.g. [10], [13], [46], [66], [77].

region graph

Definition 3.11 (region graph)**.** *Given a set of RVs $\boldsymbol{X}$, a region graph $(\mathcal{R}, \psi)$ is a rooted directed acyclic graph consisting of region nodes (R) and partition nodes (P). The root and the leaves of a region graph are regions. The root is said to be the root region, while the leaves are called atomic regions. $\mathcal{R}$ is bipartite w.r.t. regions and partitions, i.e. children of region nodes are only of type P and vice versa. We use $\mathbf{R}$ and $\mathbf{P}$ to denote the set of all region nodes and all partition nodes, respectively.*

The scope-function of a region graph, i.e. $\psi \colon \mathbf{R} \cup \mathbf{P} \mapsto \mathcal{P}(\boldsymbol{X})$, has the following properties:

1. *If $R \in \mathbf{R}$ is the root, then $\psi(R) = \boldsymbol{X}$.*
2. *If $R \in \mathbf{R}$ is a region with children, then $\psi(R) = \bigcup_{P \in \mathrm{ch}(R)} \psi(P)$ and $\psi(R) = \psi(P)\ \forall P \in \mathrm{ch}(R)$.*
3. *For each $P \in \mathbf{P}$ we have $\psi(P) = \bigcup_{R \in \mathrm{ch}(P)} \psi(R)$ and $\forall R, R' \in \mathrm{ch}(P) \colon \psi(R) \cap \psi(R') = \emptyset$.*

Note that we generalised the previous notion of a region graph [10], [43], [76] by decoupling its computational graph $\mathcal{R}$ from the scope-function ψ. Given a region graph $\mathcal{R}$, we can easily construct an SPN

structure as follows. First, we introduce a single sum node for the root region in $\mathcal{R}$; this sum node will be the output of the SPN. Then, for each atomic region we introduce $I > 0$ SPN leaves and for each other region R, which is neither root nor atomic, we introduce $J > 0$ sum nodes. Both I and J are hyperparameters and may vary depending on the region or depth of the network, if necessary.

For each partition P, we introduce all possible cross-products (Cartesian product) of nodes from P's child regions. More precisely, let $\text{ch}(P) = \{R_1, \ldots, R_k\}$ and let $\mathbf{N}_i$ be the sets of nodes assigned to regions $R_i \in \text{ch}(P)$. Then, we construct the set of all cross-products, i.e. $\mathsf{P} = \mathsf{N}_1 \times \cdots \times \mathsf{N}_k$ where $\mathsf{N}_i \in \mathbf{N}_i$ for all $i = 1 \ldots k$. After having constructed all cross-products, we connect each cross-product to each sum node in each parent region of P.

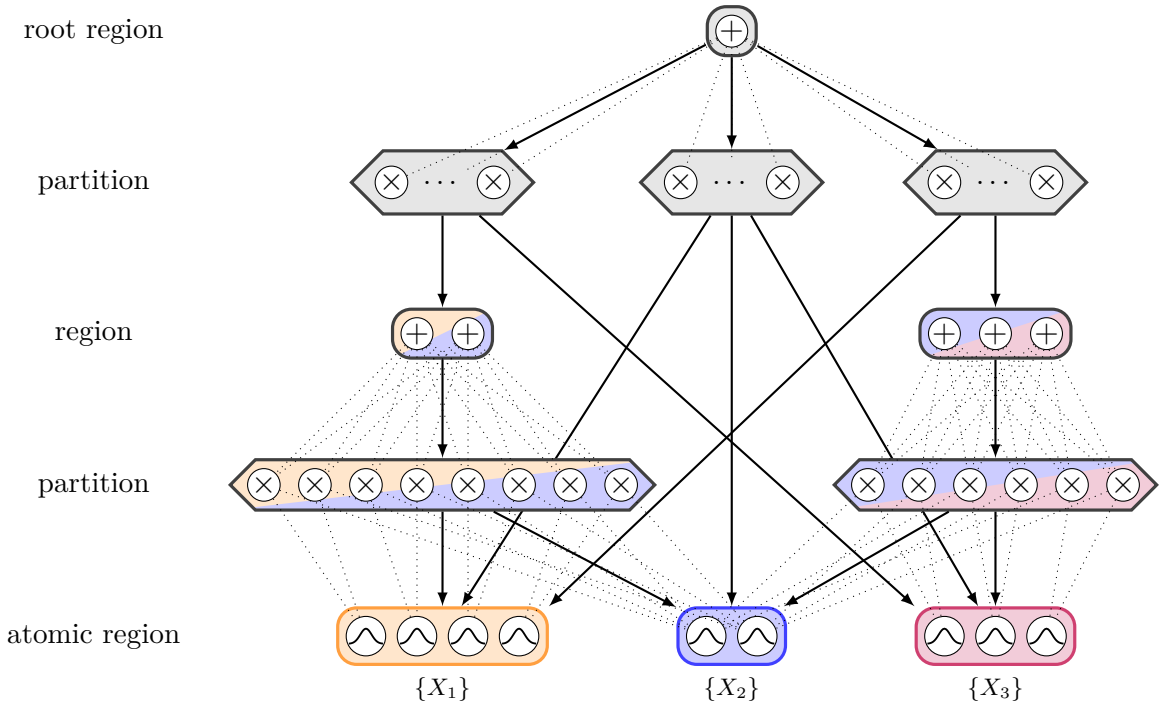


Figure 3.5: Illustration of a region graph over RVs $\boldsymbol{X} = \{X_1, X_2, X_3\}$. Regions contain a collection of sum nodes, while partitions contain all possible cross-products (Cartesian product). Atomic regions contain a collection of leaf nodes with different parameters.

Figure 3.5 illustrates a region graph, and an SPN structure constructed based on the region graph, over $\boldsymbol{X} = \{X_1, X_2, X_3\}$ with varying numbers of sum or leaf nodes per region. In this example, each atomic region holds a different number of leaf nodes, each representing a univariate distribution over the respective scope of the atomic region. Consequently, the number of cross-products in the parent partitions of the atomic regions varies from $4 \times 2 = 8$ products in case of the partition with $\psi = \{X_1, X_2\}$, $2 \times 3 = 6$ products in case of the partition with $\psi = \{X_2, X_3\}$ and $4 \times 2 \times 3 = 24$ products in case of the partition shown in the upper middle of the figure. An algorithm to convert a region graph into an SPN is illustrated in Section A.1.

3.3 Generative & Discriminative Learning

After having discussed the basics and various representations of SPNs, we will now provide an introduction into generative and discriminative parameter learning in SPNs. We want to emphasise that this is not an exhaustive discussion but instead aims to introduce the core concepts, which we will extend to the semi-supervised learning scenario in Chapter 4.

3.3.1 Generative Learning

In this section, we will review a few common approaches for generative learning of SPNs, i.e. learning SPNs for density estimation. Note that there exists a plethora of work on generative parameter learning, including various *maximum likelihood* approaches using either gradient-based optimisation [10], [77], [78] or expectation-maximisation (and related schemes) [8], [15], [16]. In addition, several Bayesian approaches to parameter learning, e.g. [18], [19], [79], have been developed over the years. This section will review the main concepts for maximum likelihood approaches and we refer to the respective publications on advanced techniques for the interested reader.

First, let $\mathcal{S} = (\mathcal{G}, \psi, \boldsymbol{w}, \boldsymbol{\theta})$ be the SPN of interest with fixed computational graph and scope-function. Moreover, let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ denote a training set containing N observations. Our goal is now to find a plausible model, specified by the weights $\boldsymbol{w} \in \mathcal{W}$ and leaf node parameters $\boldsymbol{\theta} \in \Theta$, for the data we have observed. A common approach is to employ Bayes' rule for this purpose and to consider the conditional probability distribution given as

$$p(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \mathcal{D}) = \frac{p(\mathcal{D} \,|\, \boldsymbol{w}, \boldsymbol{\theta}) p(\boldsymbol{w}, \boldsymbol{\theta})}{\underbrace{\int_{\boldsymbol{w}'} \int_{\boldsymbol{\theta}'} p(\mathcal{D}, \boldsymbol{w}', \boldsymbol{\theta}') \,\mathrm{d}\boldsymbol{w}' \,\mathrm{d}\boldsymbol{\theta}}_{=p(\mathcal{D})}} \propto \underbrace{p(\mathcal{D} \,|\, \boldsymbol{w}, \boldsymbol{\theta})}_{\text{likelihood}} \underbrace{p(\boldsymbol{w}, \boldsymbol{\theta})}_{\text{prior}}, \quad (3.12)$$

which is said to be the posterior distribution. Intuitively, we can interpret the posterior distribution as an updated prior distribution in which we have transformed our prior knowledge about $\boldsymbol{w}$ and $\boldsymbol{\theta}$ into posterior knowledge through the data $\mathcal{D}$. One possible approach to use Equation (3.12) for our purpose is to consider a single point estimate, e.g. the maximum of Equation (3.12). This approach is referred to as the maximum a-posteriori (MAP) estimate, i.e.

posterior distribution

maximum a-posteriori

$$\{\boldsymbol{w}_{\mathrm{MAP}}, \boldsymbol{\theta}_{\mathrm{MAP}}\} = \operatorname*{arg\,max}_{\boldsymbol{w} \in \mathcal{W}, \boldsymbol{\theta} \in \Theta} p(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \mathcal{D}) \,. \quad (3.13)$$

Under the usual i.i.d. assumption, the likelihood term of an SPN is written as

$$p(\mathcal{D} \,|\, \boldsymbol{w}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\boldsymbol{x}_i \,|\, \boldsymbol{w}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{S}_{\boldsymbol{w}, \boldsymbol{\theta}}(\boldsymbol{x}_i) = \mathcal{S}_{\boldsymbol{w}, \boldsymbol{\theta}}(\mathcal{D}) \,, \quad (3.14)$$

where we use the notation $\mathcal{S}_{\boldsymbol{w}, \boldsymbol{\theta}}$ to make the dependence of the value of the SPN on the parameters explicit. Note that under certain conditions

(Bernshteĭn-von-Mises theorem[15]) with increasing number of observations, the MAP estimate will converge towards the Maximum Likelihood Estimate (MLE), i.e.

maximum likelihood

$$\{\boldsymbol{w}_{\text{MLE}}, \boldsymbol{\theta}_{\text{MLE}}\} = \arg\max_{\boldsymbol{w}\in\mathcal{W},\boldsymbol{\theta}\in\Theta} \mathcal{S}_{\boldsymbol{w},\boldsymbol{\theta}}(\mathcal{D})\,. \tag{3.15}$$

Note that we often assume that the required assumptions[16] are met and that we have "enough" data such that the MAP estimate converges towards the MLE.

The two most prominent approaches to maximise Equation (3.15) are based on: i) gradient-based optimisation and ii) expectation maximisation. In case of gradient-based optimisation, we exploit the fact that we can easily compute the partial derivatives of any SPN w.r.t. the parameters of interest [67], [68]. This approach is particularly interesting as we can employ a large body of work on gradient-based black-box optimisers [81], while the gradients can be computed using automatic differentiation [82].

To avoid numerical instabilities, it is common to maximise the *log-likelihood* instead of directly maximising the likelihood term in Equation (3.15). Maximising the log-likelihood, i.e.

log-likelihood

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}) = \sum_{i=1}^{N} \log \mathcal{S}_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}_i)\,. \tag{3.16}$$

is equivalent to maximising the likelihood as the natural logarithm is a monotonically increasing transformation of the likelihood. For any sum node S in $\mathcal{S}$, the derivative of the log-likelihood given an observation $\boldsymbol{x}$ with respect to its weights is given as

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x})}{\partial w_{\mathsf{S},\mathsf{N}}} = \frac{1}{\mathcal{S}(\boldsymbol{x})} \frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial \mathsf{S}} \mathsf{N}(\boldsymbol{x})\,. \tag{3.17}$$

The gradient of data log-likelihood w.r.t. the weight $w_{\mathsf{S},\mathsf{N}}$ is therefore given as

$$\nabla_{w_{\mathsf{S},\mathsf{N}}} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x})}{\partial w_{\mathsf{S},\mathsf{N}}}\right]\,. \tag{3.18}$$

The gradients for the leaf node parameters can be obtained in a similar fashion. Note that only sum and leaf nodes are parametrised in an SPN. Given the respective gradients we can iteratively update the parameters of our model using vanilla gradient ascent, i.e. the weight will be updated at iteration $(t+1)$ using

$$w_{\mathsf{S},\mathsf{N}}^{(t+1)} \leftarrow\!\shortmid\, w_{\mathsf{S},\mathsf{N}}^{(t)} + \eta\, \nabla_{w_{\mathsf{S},\mathsf{N}}^{(t)}}\,, \tag{3.19}$$

[15] The Bernshteĭn-von-Mises theorem essentially states that the posterior distribution is asymptotically normal distributed around the maximum likelihood estimate with variance equal to the inverse of the Fisher information. See Vaart [80] for details.

[16] The Bernshteĭn-von-Mises theorem requires the following assumptions: the parameters of the true model have to lie in the interior of the parameter space, the true parameters have non-zero prior probability, the likelihood is smooth, and the number of parameters of the true model is finite. We refer to Vaart [80] for a detailed discussion.

or by utilising a more advanced optimiser. Note that the resulting weights will generally not be normalised, making it necessary to project them back onto the k-simplex, where $k = |\operatorname{ch}(\mathsf{S})|$ is the number of children of S. This projection can be done efficiently using the algorithm in Duchi, Shalev-Shwartz, Singer *et al.* [83].

Expectation Maximisation As an alternative to gradient-based parameter learning, latent variable models can also be learned using expectation maximisation (EM) [8], [16]. The EM algorithm iteratively maximises the data log-likelihood by alternating between an expectation (E) step, which computes the expected sufficient statistics according to the expected data log-likelihood, and a maximisation (M) step, which maximises the expected data log-likelihood w.r.t. the parameters. In contrast to a gradient-based approach, which requires us to project the updated weights back onto the simplex, EM results in closed-form updates at each step and automatically enforces the required constraints. This makes the application of EM for SPNs an attractive choice. Peharz, Gens, Pernkopf *et al.* [16] showed that the EM algorithm can be derived for SPNs by considering an augmentation of an SPN with so-called twin nodes. We refer to Peharz, Gens, Pernkopf *et al.* [16] for a comprehensive introduction of the latent variable model interpretation of SPNs and a derivation of the EM algorithm for SPNs.

expectation maximisation

In the E-step we compute the expected sufficient statistics, which are given for sum node S as

$$r_{\mathsf{S},\mathsf{N}}^{(t)}(\boldsymbol{x}_i) = \frac{1}{\mathcal{S}_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i)}{\partial \mathsf{S}} w_{\mathsf{S},\mathsf{N}}^{(t)} \mathsf{N}(\boldsymbol{x}_i)\,, \tag{3.20}$$

and the respective M-step in which we maximise the expected data log-likelihood w.r.t. $w_{\mathsf{S},\mathsf{N}}$ is given as

$$w_{\mathsf{S},\mathsf{N}}^{(t+1)} \leftarrow\!\shortmid \frac{\sum_{i=1}^{N} r_{\mathsf{S},\mathsf{N}}^{(t)}(\boldsymbol{x}_i)}{\sum_{\mathsf{N}' \in \operatorname{ch}(\mathsf{S})} \sum_{i=1}^{N} r_{\mathsf{S},\mathsf{N}'}^{(t)}(\boldsymbol{x}_i)}\,. \tag{3.21}$$

In a similar fashion, we can update the parameters of each leaf node, if its distribution is in the exponential family[17], using the expected sufficient statistics. Let the natural parameter of L be denoted as θ_{L}, then the E and the M-step are given as

$$g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i) = \frac{1}{\mathcal{S}(\boldsymbol{x}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i)}{\partial \mathsf{L}} p(\boldsymbol{x}_{\mathsf{L},i} \,|\, \theta_{\mathsf{L}}^{(t)})\,, \tag{3.22}$$

$$\theta_{\mathsf{L}}^{(t+1)} \leftarrow\!\shortmid \frac{\sum_{i=1}^{N} g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i) t(\boldsymbol{x}_i)}{\sum_{i=1}^{n} g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i)}\,, \tag{3.23}$$

where $t(\cdot)$ denotes the sufficient statistics.

[17] The exponential family is a class of probability distributions, which can be represented by means of a specific parametrisation, i.e. the density function is written as $p(x \,|\, \theta) = h(x) \exp(\theta\, t(x) - A(\theta))$ where $A(\theta)$ is the log-partition function, $t(x)$ the sufficient statistic of the distribution and θ the natural parameter, and admits a sufficient statistic [84], [85].

3.3.2 Discriminative Learning

After having revised generative learning in SPNs, we will briefly introduce discriminative learning. The goal in discriminative learning is to train an SPN such that it is able to discriminate between predefined classes based on a set of features. A natural approach to measure the performance of a discriminative model is the use of the *conditional log-likelihood*. Gens and Domingos [20] introduced discriminative training for SPNs, which utilises gradient-based optimisation to maximise the conditional log-likelihood. More recently, Rashwan, Poupart and Chen [86] proposed an extended Baum-Welch algorithm to maximise the conditional log-likelihood, while Peharz, Vergari, Stelzner *et al.* [10] proposed to train SPNs by maximising the cross-entropy. In this thesis we will focus on gradient-based optimisation of the conditional log-likelihood and refer to the respective works for further reading.

For a given classification problem with $1, \dots, K$ classes, let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ be a set of tuples, each consisting of a feature or covariate $\boldsymbol{x}_i$ and a label or target $\boldsymbol{y}_i \in \mathbb{B}^K$ with $\|\boldsymbol{y}_i\|_0 \equiv 1$, i.e. one-hot encoding.

conditional log-likelihood

Then, the *conditional log-likelihood* of an SPN is given as

$$\mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}) = \sum_{i=1}^{N} \log \underbrace{\left(\sum_{k=1}^{K} y_{i,k}\, w_k\, \mathcal{S}_k(\boldsymbol{x}_i) \right)}_{=\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)} - \log \underbrace{\left(\sum_{k=1}^{K} w_k\, \mathcal{S}_k(\boldsymbol{x}_i) \right)}_{=\mathcal{S}(\boldsymbol{x}_i)} . \tag{3.24}$$

Note that the last term marginalises over all classes.

To optimise Equation (3.24), Gens and Domingos [20] proposed to either perform gradient ascent based on the partial derivatives of an SPN ("soft" inference) or to use gradients computed for "hard" inference, i.e. replacing sum nodes with max-operations. In the first case, the partial derivatives of the weights and the leaf node parameters are given as

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y})}{\partial w_{\mathsf{S},\mathsf{N}}} = \frac{1}{\mathcal{S}(\boldsymbol{x}, \boldsymbol{y})} \frac{\partial \mathcal{S}(\boldsymbol{x}, \boldsymbol{y})}{\partial w_{\mathsf{S},\mathsf{N}}} - \frac{1}{\mathcal{S}(\boldsymbol{x})} \frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial w_{\mathsf{S},\mathsf{N}}} , \tag{3.25}$$

and

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y})}{\partial \theta_{\mathsf{L}}} = \frac{1}{\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta_{\mathsf{L}}} - \frac{1}{\mathcal{S}(\boldsymbol{x}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i)}{\partial \theta_{\mathsf{L}}} . \tag{3.26}$$

The derivatives of $\mathcal{S}$ w.r.t. the weights or leaf node parameters are calculated as follows

$$\frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial w_{\mathsf{S},\mathsf{N}}} = \frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial \mathsf{S}} \mathsf{N}(\boldsymbol{x}) , \tag{3.27}$$

$$\frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial \theta_{\mathsf{L}}} = \frac{\partial \mathcal{S}(\boldsymbol{x})}{\partial \mathsf{L}} \frac{\partial p(\boldsymbol{x}_{\mathsf{L}} \mid \theta_{\mathsf{L}})}{\partial \theta_{\mathsf{L}}} . \tag{3.28}$$

To calculate the respective partial derivatives of $\mathcal{S}$ w.r.t. a node in the SPN, we apply the chain-rule. By setting the gradient at the root node to one, the gradients of the subsequent nodes can be computed in a single top-down pass. Table 3.1 lists the respective update-rules

for nodes within an SPN. Note that we will focus on "soft" inference in this thesis and refer to Gens and Domingos [20] for details on "hard" inference and the respective update rules.

Table 3.1: Gradient propagation rules for node N *in an SPN.*

Parent Node	Propagation Rule
Sum S	$\frac{\partial \mathcal{S}}{\partial \mathsf{N}} \mapsto \frac{\partial \mathcal{S}}{\partial \mathsf{N}} + w_{\mathsf{S},\mathsf{N}} \frac{\partial \mathcal{S}}{\partial \mathsf{S}}$
Product P	$\frac{\partial \mathcal{S}}{\partial \mathsf{N}} \mapsto \frac{\partial \mathcal{S}}{\mathsf{N}} + \frac{\partial \mathcal{S}}{\partial \mathsf{P}} \prod_{\mathsf{N}' \in \mathrm{ch}(\mathsf{P}) \setminus \{\mathsf{N}\}} \mathsf{N}'$

3.4 Implicit Acceleration Effects

So far, we have discussed various ways to perform parameter learning in SPNs. Even though many approaches utilise gradient-based optimisation, it is not clear if and to which extent the depth of an SPNs has an effect on the optimisation. However, analysing the dynamics of the optimisation for linear neural networks – see Baldi and Hornik [87] for a survey on linear neural networks – has gained increasing attention [88]–[90]. In particular, Arora, Cohen and Hazan [90] showed that increasing the depth in linear neural networks results in implicit acceleration effects and can speed up the optimisation. In this section, we will discuss the implicit acceleration effects of overparameterisation in tree-shaped SPNs, i.e. SPTs.

3.4.1 Preliminaries

As shown by Arora, Cohen and Hazan [90], gradient-based optimisation of linear regression models benefits from moving from a convex objective function to a non-convex objective. In particular, let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with $\boldsymbol{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$ denote a training set. Our goal is to find model parameters $\boldsymbol{w} \in \mathbb{R}^D$ for a linear regression function $\hat{y} = \boldsymbol{x}^\top \boldsymbol{w}$ by minimizing the ℓ_p loss function, i.e.

$$\ell_p(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{x}_i, y_i) \sim \mathcal{D}} \left[\frac{1}{p} (\boldsymbol{x}_i^\top \boldsymbol{w} - y_i)^p \right] . \tag{3.29}$$

Suppose that we now artificially overparameterise this linear model using $\boldsymbol{w} = \boldsymbol{w}_1 \; w_2$, for which $\boldsymbol{w}_1 \in \mathbb{R}^D$ and $w_2 \in \mathbb{R}$. Note that this overparameterisation results in a non-convex loss function, i.e.

$$\ell_p(\boldsymbol{w}_1, w_2) = \mathbb{E}_{(\boldsymbol{x}_i, y_i) \sim \mathcal{D}} \left[\frac{1}{p} (\boldsymbol{x}^\top \boldsymbol{w}_1 w_2 - y_i)^p \right] , \tag{3.30}$$

while our original loss function was convex. Let us now consider the respective gradients, i.e.

$$\nabla_{\boldsymbol{w}_1} = \mathbb{E}_{(\boldsymbol{x}_i, y_i) \sim \mathcal{D}} \left[(\boldsymbol{x}_i^\top \boldsymbol{w}_1 w_2 - y_i)^{p-1} w_2 \, \boldsymbol{x}_i \right] = \nabla_{\boldsymbol{w}} \, w_2 \, , \tag{3.31}$$

$$\nabla_{w_2} = \mathbb{E}_{(\boldsymbol{x}_i, y_i) \sim \mathcal{D}} \left[(\boldsymbol{x}_i^\top \boldsymbol{w}_1 w_2 - y_i)^{p-1} \boldsymbol{w}_1^\top \boldsymbol{x}_i \right] . \tag{3.32}$$

To optimise the loss function we will update the parameters at time $t+1$ as follows,

$$\boldsymbol{w}_1^{(t+1)} \leftarrow\!\shortmid \boldsymbol{w}_1^{(t)} - \eta \nabla_{\boldsymbol{w}_1^{(t)}} , \tag{3.33}$$

$$\boldsymbol{w}_2^{(t+1)} \leftarrow\!\shortmid \boldsymbol{w}_2^{(t)} - \eta \nabla_{\boldsymbol{w}_2^{(t)}} , \tag{3.34}$$

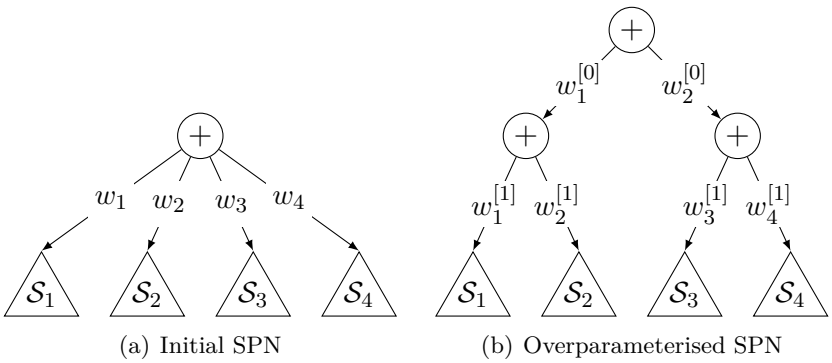
where $\eta \geq 0$ is a fixed (usually small) learning rate. To understand the dynamics of the underlying parameter $\boldsymbol{w} = \boldsymbol{w}_1 \, w_2$, Arora, Cohen and Hazan [90] showed that

$$\begin{aligned}
\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}_1^{(t+1)} \, w_2^{(t+1)} \\
&= \left(\boldsymbol{w}_1^{(t)} - \eta \nabla_{\boldsymbol{w}_1^{(t)}}\right) \left(w_2^{(t)} - \eta \nabla_{w_2^{(t)}}\right) \\
&= \boldsymbol{w}^{(t)} - \eta \, w_2^{(t)} \nabla_{\boldsymbol{w}_1^{(t)}} - \eta \, \boldsymbol{w}_1^{(t)} \nabla_{w_2^{(t)}} + \underbrace{\eta^2 \nabla_{\boldsymbol{w}_1^{(t)}} \nabla_{w_2^{(t)}}}_{\approx 0} \\
&\approx \boldsymbol{w}^{(t)} - \underbrace{\eta (w_2^{(t)})^2}_{=\rho^{(t)}} \nabla_{\boldsymbol{w}^{(t)}} - \underbrace{\eta / w_2^{(t)} \nabla_{w_2^{(t)}}}_{=\gamma^{(t)}} \boldsymbol{w}^{(t)} \\
&= \boldsymbol{w}^{(t)} - \rho^{(t)} \nabla_{\boldsymbol{w}^{(t)}} - \gamma^{(t)} \boldsymbol{w}^{(t)} .
\end{aligned} \tag{3.35}$$

Arora, Cohen and Hazan [90] further show that overparameterising fully connected linear neural networks, i.e. increasing the number of layers, leads to an adaptive learning rate and gradient projection amplification that can be thought of as momentum optimisation [91]. Later, Arora, Cohen, Hu *et al.* [88] showed that an increased depth in linear neural networks also enhances an implicit tendency towards low-rank solutions of the matrix factorisation represented by the network.

3.4.2 Overparameterisation in Sum-Product Networks

Following the approach by Arora, Cohen and Hazan [90] we will now discuss the implicit dynamics of gradient-based optimisation in SPTs. To simplify the discussion, consider the network structure illustrated in Figure 3.6(a), i.e. an SPN with a root node that computes a weighted sum over sub-networks $\mathcal{S}_1, \ldots, \mathcal{S}_K$.



(a) Initial SPN (b) Overparameterised SPN

Figure 3.6: Illustration of an overparameterised SPN. Triangles denote sub-SPNs or distributions.

Let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ denote a training set, then the data log-likelihood of the SPN illustrated in Figure 3.6(a) is given as

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \mathcal{D}) = \sum_{i=1}^{N} \log \Bigg(\underbrace{\sum_{j=1}^{K} w_j \mathcal{S}_j(\boldsymbol{x}_i)}_{=\mathcal{S}(\boldsymbol{x}_i)} \Bigg) - \log \mathcal{S}(*)\,, \tag{3.36}$$

where $\mathcal{S}(*)$ denotes an evaluation of the SPN with all indicator set to one, i.e. the normalisation constant or partition function. Note that the normalisation is only necessary if the network is not (locally) normalised, i.e. $\mathcal{S}(*) \neq 1$. To derive the gradient updates in case of overparameterisation, we will assume that all weights $w_j \in \mathbb{R}_{>0}$ are initialised close to zero, i.e. $w_j \gtrapprox 0$, and thus the SPN is unnormalised.

To maximise Equation (3.36) we assume vanilla gradient ascend using

$$\nabla_{w_j} = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}} \left[\frac{1}{\mathcal{S}(\boldsymbol{x})} \mathcal{S}_j(\boldsymbol{x}) - \frac{1}{\mathcal{S}(*)} \mathcal{S}_j(*) \right], \tag{3.37}$$

$$w_j^{(t+1)} = w_j^{(t)} + \eta \, \nabla_{w_j}\,, \tag{3.38}$$

where $\mathcal{S}_j(*)$ denotes the partition function of the j^{th} sub-network and $\eta \geq 0$ is assumed to be small.

Let us now consider an overparameterised version by introducing additional sum nodes into the model. For this purpose, let each weight w_j be decomposed into multiple independent weights, akin to the linear regression example. Figure 3.6(b) illustrates the overparameterised version of our initial network. This SPN is in its expressiveness equivalent to our initial model and only introduces additional parameters. Note that both models are assumed to be tree-structured.

Let $w_i^{[l]}$ denote the i^{th} weight of layer l. We can define the decomposition of w_j by L sum node layers as

$$w_j = \prod_{l=0}^{L} w_{\phi(j,l)}^{[l]}\,, \tag{3.39}$$

where $\phi(j, l)$ maps the index j onto the respective index at layer l. For example, in the case of our model in Figure 3.6 the mapping $\phi(j, l)$ is defined as

$$\phi(j,l) \coloneqq \begin{cases} \left\lceil \frac{j}{2} \right\rceil, & \text{if } l = 0 \\ j, & \text{otherwise} \end{cases}, \tag{3.40}$$

where $\lceil\, \rceil$ denotes the ceiling operator. The definition of $\phi(j, l)$ generally depends on the network structure of the SPN.

Similar to Equation (3.36), we can denote the log-likelihood function of the overparameterised network as

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \mathcal{D}) = \sum_{i=1}^{N} \log \Bigg(\sum_{j=1}^{K} \underbrace{\prod_{l=0}^{L} w_{\phi(j,l)}^{[l]}}_{=w_j} \mathcal{S}_j(\boldsymbol{x}_i) \Bigg) - \log \mathcal{S}(*)\,. \tag{3.41}$$

Let $\gamma_j \coloneqq \left\lceil \frac{j}{2} \right\rceil$ and let us assume that $w_j = w_{\gamma_j}^{[0]} w_j^{[1]}$ for each weight. Therefore, the gradient of weight $w_j^{[1]}$ is given as

$$\begin{aligned} \nabla_{w_j^{[1]}} &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{w_{\gamma_j}^{[0]}}{\mathcal{S}(\boldsymbol{x}_i)}\mathcal{S}_j(\boldsymbol{x}_i) - \frac{w_{\gamma_j}^{[0]}}{\mathcal{S}(*)}\mathcal{S}_j(*)\right] \\ &= w_{\gamma_j}^{[0]} \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{1}{\mathcal{S}(\boldsymbol{x}_i)}\mathcal{S}_j(\boldsymbol{x}_i) - \frac{1}{\mathcal{S}(*)}\mathcal{S}_j(*)\right]}_{=\nabla_{w_j}} . \end{aligned} \tag{3.42}$$

The gradient for weight $w_{\gamma_j}^{[0]}$, on the other hand, depends on the value of each child under sum node $S_{\phi(j,0)}^{[1]}$ the edge, which is associated with the weight $w_{\gamma_j}^{[0]}$, terminates in, i.e.

$$\begin{aligned} \nabla_{w_{\gamma_j}^{[0]}} &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\sum_{\mathsf{N}\in\mathrm{ch}\left(\mathsf{S}_{\phi(j,0)}^{[1]}\right)} \frac{w_{\mathsf{N}}^{[1]}}{\mathcal{S}(\boldsymbol{x}_i)}\mathcal{S}_{\mathsf{N}}(\boldsymbol{x}_i) - \frac{w_{\mathsf{N}}^{[1]}}{\mathcal{S}(*)}\mathcal{S}_{\mathsf{N}}(*)\right] \\ &= \sum_{\mathsf{N}\in\mathrm{ch}\left(\mathsf{S}_{\phi(j,0)}^{[1]}\right)} w_{\mathsf{N}}^{[1]} \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{1}{\mathcal{S}(\boldsymbol{x}_i)}\mathcal{S}_{\mathsf{N}}(\boldsymbol{x}_i) - \frac{1}{\mathcal{S}(*)}\mathcal{S}_{\mathsf{N}}(*)\right]}_{=\nabla_{w_{\mathsf{N}}}} . \end{aligned} \tag{3.43}$$

In more general terms we can say that the gradient of $w_{\phi(j,l)}^{[l]}$ at layer l is given as

$$\nabla w_{\phi(j,l)}^{[l]} = \sum_{w_{\phi(j,l)}^{[l]} \rightsquigarrow \mathsf{N}} \frac{w_{\mathsf{N}}}{w_{\phi(j,l)}^{[l]}} \nabla w_{\mathsf{N}} \,, \tag{3.44}$$

where we use $w_{\phi(j,l)}^{[l]} \rightsquigarrow \mathsf{N}$ to denote the set of all weights $w_{\mathsf{N}} = \prod_{l=0}^{L} w_{\phi(\mathsf{N},l)}^{[l]}$, which are part of the decomposition of w_{N}. In other words, this set contains the weights of all edges on the unique path from the root to the node N.

Similar to Arora, Cohen and Hazan [90], we can examine the dynamics of w_j by assuming a small learning rate $\eta \geq 0$ and an initialisation of all weights close to zero. For this let $w_j^{(t+1)}$ denote w_j at time $t+1$. We can now perform a derivation similar to the one used in case of a linear regression model, i.e.

$$w_j^{(t+1)} = w_{\gamma_j}^{[0](t+1)} \, w_j^{[1](t+1)} \tag{3.45}$$

$$= \left(w_{\gamma_j}^{[0](t)} + \eta \nabla_{w_{\gamma_j}^{[0](t)}}\right)\left(w_j^{[1](t)} + \eta \nabla_{w_j^{[1](t)}}\right) \tag{3.46}$$

$$= w_j^{(t)} + \eta (w_{\gamma_j}^{[0](t)})^2 \nabla_{w_j^{(t)}} + \eta w_{\gamma_j}^{[0](t)} \nabla_{w_{\gamma_j}^{[0](t)}} w_j^{(t)} \tag{3.47}$$

$$+ \underbrace{\eta^2 \nabla_{w_{\gamma_j}^{[0](t)}} \nabla_{w_j^{[1](t)}}}_{\approx 0} . \tag{3.48}$$

Consequently, we can reformulate Equation (3.48) to results in an gradient update consisting of an adaptive learning rate and an adaptive

momentum term, i.e.

$$w_j^{(t+1)} \approx w_j^{(t)} + \underbrace{\eta(w_{\gamma_j}^{[0](t)})^2}_{=\rho^{(t)}} \nabla_{w_j^{(t)}} + \underbrace{\eta / w_{\gamma_j}^{[0](t)} \nabla_{w_{\gamma_j}^{[0](t)}}}_{=\gamma^{(t)}} w_j^{(t)} \,. \tag{3.49}$$

We can find the solution for an arbitrary decompositions represented in an overparameterised SPN, c.f. Equation (3.39), using the gradients in Equation (3.44). The resulting update rule for the weights of any overparameterised SPNs can be written as

$$w_j^{(t+1)} \leftarrow\!\shortmid w_j^{(t)} + \eta(w_{\phi(j,0)}^{[0]})^2 \nabla_{w_j^{(t)}} + \left[\sum_{l=0}^{L-1} \eta \nabla_{w_{\phi(j,l)}^{[l]}} (w_{\phi(j,l)}^{[l]})^{-1}\right] w_j^{(t)} \,, \tag{3.50}$$

where we dropped terms that are approximately zero, provided that η is small. We can now define the adaptive and time-varying learning rate

$$\rho^{(t)} \coloneqq \eta(w_{\phi(j,0)}^{[0]})^2 \,, \tag{3.51}$$

and let

$$\gamma^{(t)} \coloneqq \sum_{l=0}^{L-1} \eta \nabla_{w_{\phi(j,l)}^{[l]}} (w_{\phi(j,l)}^{[l]})^{-1} \,. \tag{3.52}$$

Note that we pulled out $l = 0$ to obtain w_j, c.f. last term in Equation (3.50), resulting in the factor $\gamma^{(t)}$ to be a summation over $L - 1$ terms. Therefore, the gradient updates of weight w_j are directly influenced by the depth of the network.

If all weights are initialisation near zero, then we can understand the updated weight $w_j^{(t)}$ as a weighted combination of past gradients and, thus, there exists a $\mu^{(t,\tau)} \in \mathbb{R}$ such that the dynamics of $w_j^{(t)}$ correspond to gradient optimisation with an additional momentum term, i.e.

$$w_j^{(t)} \approx w_j^{(t)} + \rho^{(t)} \nabla_{w_j^{(t)}} + \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \nabla_{w_K^{(\tau)}} \,. \tag{3.53}$$

Corollary 2. *Gradient-based optimisation of an overparameterised sum-product network with small (fixed) learning rate and near zero initialisation of the weights is equivalent to gradient-based optimisation with adaptive and time-varying learning rate and momentum term.*

So far we have only considered the case in which we artificially introduce additional sum nodes to the network. However, an important question is if the depth of an SPN also implicitly accelerates parameter learning. To answer this question we will examine the sub-class of SPTs. As discussed in Section 3.1.1, the value of any SPN (and any SPT) can be written as a mixture over induced trees. Using the representation of an SPT by a mixture of K many induced trees, the weight of every component $1 \leq j \leq K$ is given by the weights of the respective induced

tree, i.e.

$$w_j = \prod_{w_{\mathsf{S},\mathsf{N}} \in E(\mathcal{T}_j)} w_{\mathsf{S},\mathsf{N}} \,. \tag{3.54}$$

Let ϕ denote an index function $\phi\colon \mathbb{Z}\times\mathbb{Z} \mapsto \mathbf{S}\times\mathbf{N}$, where $\mathbf{S}$ is the set of sum nodes in $\mathcal{S}$ and $\mathbf{N}$ the set of nodes. Further, let there exist a ϕ such that the decompositions of the component weights can be represented by an overparameterised SPN without changing the decompositions, the order of the weights or the weights itself, i.e.

$$w_j = \prod_{l=0}^{L_{\mathcal{T}_j}} w^{[l]}_{\phi(j,l)} = \prod_{w_{\mathsf{S},\mathsf{N}} \in E(\mathcal{T}_j)} w_{\mathsf{S},\mathsf{N}} \,, \tag{3.55}$$

where $L_{\mathcal{T}_j}$ is the depth of the induced tree. Note that there always exists such an index function if the SPN is a tree, as there exist a unique path to each node in the SPN. Thus, we can say that gradient-based optimisation using a mixture representation of the SPN, i.e. $w_j = \prod_{w_{\mathsf{S},\mathsf{N}} \in \mathcal{T}_j} w_{\mathsf{S},\mathsf{N}}$, is equivalent to the optimisation on its overparameterised representation and entails similar acceleration effects.

Claim 1. *Gradient-based optimisation of any deep tree-structured sum-product network, using its mixture representation, with small (fixed) learning rate and near zero initialisation of the weights is equivalent to gradient-based optimisation with adaptive and time-varying learning rate and momentum terms.*

Note that this claim indicates that the depth of a network can help to accelerate learning the weight of a tree-shaped SPN even if the leaf nodes contain non-linearities. Thus, learning deep tree-shaped SPNs is potentially more efficient, in terms of the number of iterations required, than gradient-based learning of shallow models.

3.4.3 Empirical Results

To empirically assess the effects of overparameterisation in SPNs, we trained SPNs with different number of additional layers on three binary datasets. Note that each deep model[18] has the same expressive power as the shallow equivalent. We used SPNs with 8 components (leaves) and initialised all weights randomly close to zero. To ensure all weights will be positive, we used a bijective transformation of the weights into an unconstrained space and performed the optimisation w.r.t. the unconstrained weights, i.e. the constrained weights $w_{\mathsf{S},\mathsf{N}} \in \mathbb{R}_{>0}$ are obtained from the unconstrained parameters $v_{\mathsf{S},\mathsf{N}} \in \mathbb{R}$ using $w_{\mathsf{S},\mathsf{N}} = \exp(v_{\mathsf{S},\mathsf{N}})$. Further, we used vanilla gradient ascend for 1000 iterations with a learning rate of $\eta = 0.1$. The results on the data extracted from the National Long Term Care Survey (NLTCS), the USDA plants dataset (Plants) [92] and the Audio dataset [20] are shown in Figure 3.7. We see from the empirical experiment that increasing the number of sum node layers, which is equivalent to increasing the depth of the network, can lead to faster parameter optimisation. We want to emphasise that all models, no matter what depth, are equally expressive and have the same

[18] We consider a model to be deep if it has more than one sum layer.

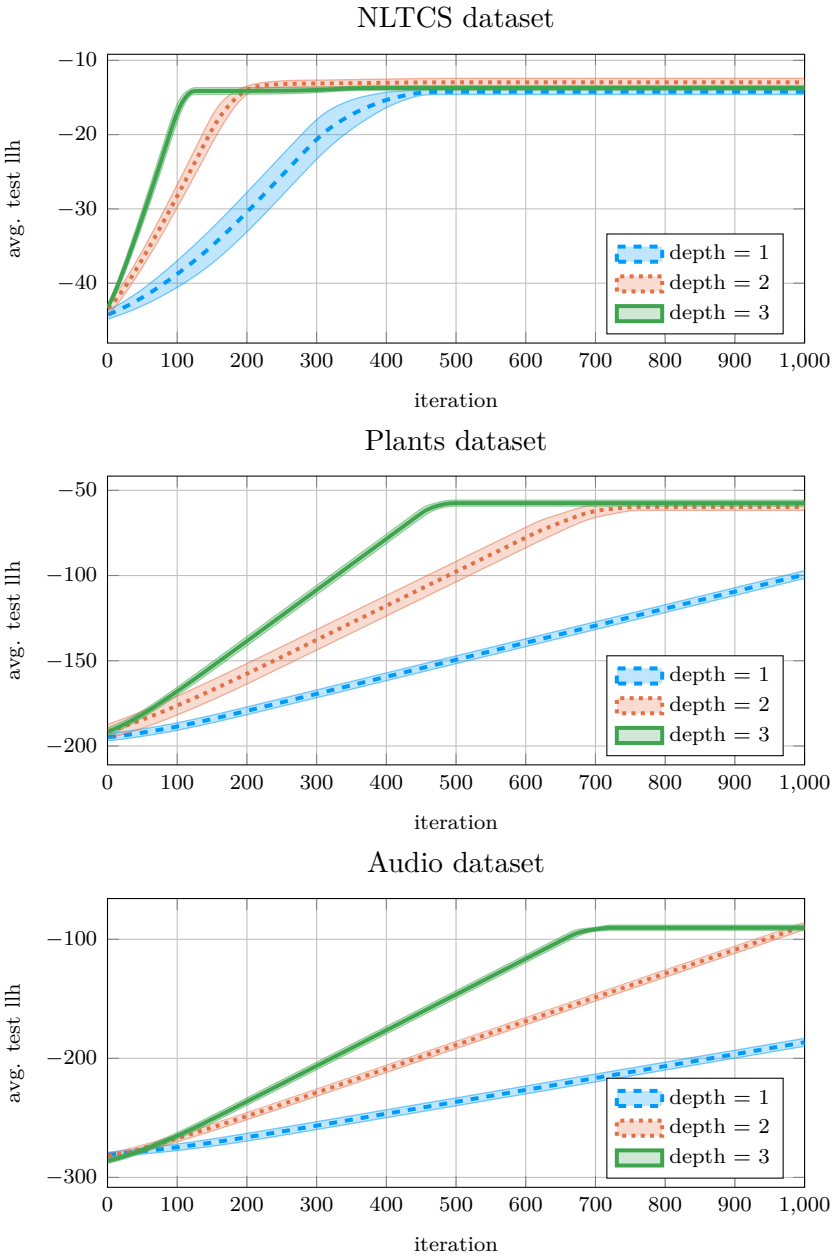


Figure 3.7: Evaluation of overparameterisation in SPNs on three discrete datasets. We can observe a general trend towards acceleration in learning for deeper models.

number of leaves, i.e. the models differ only in the number of sum node parameters.

3.4.4 Conclusion

We have seen that overparameterised SPNs exhibit similar dynamics as observed in linear neural networks. In fact, gradient-based optimisation in SPNs corresponds to optimisation with adaptive and time-varying learning rate and momentum term, naturally leading to an implicit acceleration of the optimisation. Further, deep SPTs entail the same acceleration effects as overparameterised SPNs. The acceleration effects in

SPNs are not completely surprising as SPNs are multi-linear functions in their weights, which can be represented by sparsely connected linear-neural networks with potentially non-linear inputs. However, given that SPTs have been successfully applied in non-linear estimation and classification task, e.g. [20], [93]–[95], these results indicate that the depth of the network can help to learn suitable parameters for complex modelling tasks more efficiently.

3.5 Structure Learning

So far we have assumed that the structure, i.e. the computational graph and the scope-function, of an SPN is known. Even though this can be the case, e.g. SPN structures can be obtain through compilation from a Bayesian network [67], it is often desirable to construct a structure that is tailored to the data or task at hand.

Therefore, a large variety of techniques for structure learning in SPNs were proposed over the years. The majority of structure learning approaches for SPNs, such as LearnSPN [36] and its variants, e.g. [38]–[40], [96], or other approaches, such as [41], [43]–[45], [76], heuristically generate a structure by optimising some local criterion. LearnSPN [36], which is one of the most prominent techniques, recursively constructs the SPN structure by: i) adding product nodes if the algorithm is able to identify mutually independent subsets of RVs (using a G-test), or ii) adding sum nodes obtained using hard clustering. The recursion terminates if the number of RVs reduces to a single RV or the number of observations reaches a lower threshold, in both cases resulting in a full factorisation of the scope. Therefore, the procedure potentially constructs highly complex SPTs with univariate leaves. Later, Rooshenas and Lowd [38] extended the LearnSPN algorithm to overcome its various limitations, e.g. full factorisation and potentially very deep networks, which are prune to overfitting, and proposed a complex technique by combining SPN structure learning with learning tractable Markov networks as leaves. The resulting algorithm, called ID-SPN [38], is one of the best performing structure learning approaches on discrete data as of today. Other approach construct random structures [10] or exploit rank-1 submatrices [41] to construct the structure. We refer to Paris, Sanchez-Cauce and Diez [97] for an overview on existing approaches.

In addition to the plethora of heuristic structure learners, there exists a few approaches for principled structure learning in SPNs. Most notably, Vergari, Molina, Peharz *et al.* [17] proposed ABDA, which uses posterior inference over parameters of latent variable models located at the leaves of the SPN. Even though this approach relies on a pre-defined SPN structure, the approximate Bayesian inference applied in ABDA can be understood as some kind of local Bayesian structure learning. Moreover, there has been some work on Bayesian nonparametric formulations of SPNs [98], [99], promising structure learning through the use of flexible nonparametric priors for both structure and parameters. However, both approaches had only limited success. In Chapter 5 we will introduce a fully Bayesian approach, which leverages the notion of a computational graph and a scope-function to efficiently perform ap-

proximate posterior inference over SPN structures and parameters. In further consequence, we will highlight how this framework can also be used to learn nonparametric formulations efficiently.

Besides structure learning in SPNs, there exist various approaches for other probabilistic circuits, such as Probabilistic Sentential Decision Diagrams (PSDDs) [9] and Cutset Networks (CNets) [100]. Most notably, the work by Liang, Bekker and Broeck [11] introduces a greedy approach that optimises a global objective for structure learning in PSDDs, similar to structure learning of selective SPNs [101], which are a restricted sub-type of SPNs. Existing approaches for CNets often use heuristics to define the structure, e.g. Mauro, Vergari, Basile *et al.* [102] constructs random structures and Rahman, Jin and Gogate [103] compile a learned latent variable model, such as an SPNs, into a CNet.

4

Semi-Supervised Learning of Sum-Product Networks

4.1 Motivation

In several domains, unlabelled data is abundant and cheap to acquire, while obtaining class labels is expensive and sometimes infeasible at large scale. In such cases, semi-supervised learning can be used to exploit large amounts of unlabelled data in addition to the few labelled observations. Applications that utilise semi-supervised learning range from natural language processing, e.g. [34], and image processing, e.g. [22], [23], [30], to processing of biological data, e.g. in genomics [25]. We refer to Chapelle, Schölkopf and Zien [27] for a detailed overview.

Motivated by the abundance of real-world applications for semi-supervised learning, a large number of approaches have been proposed over the years, including self-learning [27], Transductive Support-Vector Machines (TSVM) [32], and graph-based methods [104]. Many semi -supervised learning approaches exploit the intrinsic geometry of the data, e.g. TSVMs and its recent extensions [31], exploit low-density regions along the decision boundary. However, such approaches can yield suboptimal performance if the enforced assumptions are not met. We refer to Zhu and Goldberg [28] and Engelen and Hoos [105] for a comprehensive survey on semi-supervised learning.

For probabilistic models, such as SPNs, a natural approach is to incorporate the missing missing labels into the probabilistic model. In such a setting, for a classification problem with K classes, the unknown labels $\boldsymbol{v} = \{v_j\}_{j=1}^{M}$ of the M unlabelled observations $\{\boldsymbol{u}_j\}_{j=1}^{M}$ are then treated as additional parameters of the probabilistic model, i.e.

$$\arg\max_{\theta \in \Theta} \left(\mathcal{L}(\theta \mid \mathcal{D}) + \max_{\boldsymbol{v} \in V^M} \sum_{j=1}^{M} \log p(\boldsymbol{u}_j, v_j \mid \theta) \right) , \tag{4.1}$$

where $V = \{1, \ldots, K\}$, θ compromises the parameters of the model, and $\mathcal{D}$ contains only the labelled data. Clearly, the number of possible labellings grows exponentially with the number of unlabelled examples in this formulation. Approaches, such as self-learning or EM, aim to circumvent this difficulty by instead: i) training the model only on the labelled data, ii) obtaining the "best" labels for the unlabelled data from the trained model and iii) retraining the model using a fully labelled dataset. EM effectively optimises the same objective as self-learning, but uses the posterior probabilities instead of hard labels [106].

The primary concern with these approaches is that they can suffer

from degrading performance with an increasing number of unlabelled data [30], [107]. To overcome this issue, Loog [35] proposed a formulation for *safe*[19] semi-supervised learning of generative linear models, which provides guarantees that the semi-supervised learner cannot deteriorate in performance when trained with additional unlabelled data.

In this chapter, we will first review the work on contrastive pessimistic likelihood estimation [35] for linear models in Section 4.2, and subsequently introduce extensions to safe semi-supervised learning in SPNs, c.f. Section 4.3. We discuss the generative case in Section 4.3.1 and extend the framework to discriminative safe semi-supervised SPNs in Section 4.3.2. Finally, we discuss an algorithm to learn safe semi-supervised SPNs and show empirical evidence that this approach is a promising strategy for semi-supervised learning, c.f. Section 4.4.

4.2 Preliminaries

4.2.1 Contrastive Pessimistic Likelihood Estimation

Most semi-supervised learning approaches require strong assumptions on the data, e.g. low density assumptions, and can lead to decreased performance with an increasing number of unlabelled data if these assumptions are violated [30], [105], [107]. Loog [35] proposed a principled objective for semi-supervised learning, which facilitates performance guarantees, while only relying on the assumptions of the underlying generative model. We will briefly review this approach for linear generative models, as introduced in [35], and extend this approach in the subsequent sections.

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ denote the set of labelled data and let $\mathcal{U} = \{\boldsymbol{u}_j\}_{j=1}^M$ be the set of unlabelled data. Further, let $q_{j,k} \geq 0$ denote soft labels (or hypothetical posterior probabilities) for each unlabelled example and let them be defined on the $K-1$ simplex, i.e. $\boldsymbol{q}_j \in \Delta_{K-1}$. We will denote the model parameters obtained by maximising the log-likelihood of the model on $\mathcal{D}$ using θ_{sup}, i.e. the supervised model parameters, and use θ for the semi-supervised model parameters.

The general idea of Loog [35] is to implicitly constrain the parameter space Θ of the semi-supervised learner through the relative improvement of the semi-supervised learner compared to a supervised one. In more practical terms, let $\mathcal{L}(\mu, \Sigma \,|\, \mathcal{D})$ denote the log-likelihood function of a Linear Discriminant Analysis (LDA) for labelled data, i.e.

$$\mathcal{L}(\mu, \Sigma \,|\, \mathcal{D}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}_{\{y_i = k\}} \log \left(\pi_k \, \mathcal{N}(\boldsymbol{x}_i \,|\, \mu_k, \Sigma) \right) , \tag{4.2}$$

where π_k with $\sum_k \pi_k = 1$ denotes the k^{th} class prior and $\mu = \{\mu_k\}_{k=1}^K$ and Σ denote the model parameters we aim to learn. To incorporate the unlabelled data, Loog [35] suggests a natural extension of the log-likelihood function for the joint dataset (labelled and unlabelled data)

[19] Safe refers to the property that adding unlabelled data can increase, but not degrade, the performance of the semi-supervised learner.

by additionally accounting for the unlabelled data given fixed soft labels, i.e.

$$\mathcal{L}(\mu, \Sigma \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) = \mathcal{L}(\mu, \Sigma \,|\, \mathcal{D}) + \sum_{j=1}^{M} \sum_{k=1}^{K} q_{j,k} \log p(\boldsymbol{u}_j, k \,|\, \mu, \Sigma) \tag{4.3}$$

$$= \mathcal{L}(\mu, \Sigma \,|\, \mathcal{D}) + \sum_{j=1}^{M} \sum_{k=1}^{K} q_{j,k} \log \left(\pi_k \, \mathcal{N}(\boldsymbol{u}_j \,|\, \mu_k, \Sigma) \right) . \tag{4.4}$$

Consequently, for any given $\boldsymbol{q}$ we can measure the relative improvement of a semi-supervised learner θ, where θ constitutes μ and Σ in our example, compared to a purely supervised learner θ_{sup} in terms of their ratio, which can be expressed as the difference of their respective log-likelihood values, i.e.

$$\mathcal{L}(\theta \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) - \mathcal{L}(\theta_{\text{sup}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) . \tag{4.5}$$

However, in semi-supervised learning, $\boldsymbol{q}$ is of course unknown, raising the question of how to choose $\boldsymbol{q}$ in the first place. Loog [35] suggests to consider the most pessimistic setting of $\boldsymbol{q}$, i.e. choosing the soft labels such that the relative improvement is minimised. The resulting objective is called the Contrastive Pessimistic Likelihood (CPL) and is given as

contrastive pessimistic likelihood

$$\text{CPL} = \min_{\boldsymbol{q} \in \Delta_{K-1}^{M}} \mathcal{L}(\theta \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) - \mathcal{L}(\theta_{\text{sup}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) . \tag{4.6}$$

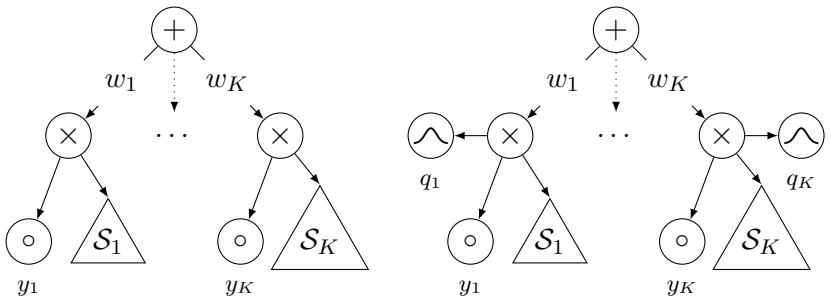
Maximising the CPL objective, i.e.

$$\theta_{\text{CPL}} = \arg\max_{\theta \in \Theta} \min_{\boldsymbol{q} \in \Delta_{K-1}^{M}} \mathcal{L}(\theta \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) - \mathcal{L}(\theta_{\text{sup}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) , \tag{4.7}$$

results in an estimate of the semi-supervised learner such that the performance of the model, provided the true labelling $\boldsymbol{q}^*$ of $\mathcal{U}$, is at least as good as in the supervised case, i.e.

$$\mathcal{L}(\theta_{\text{CPL}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}^*) \geq \mathcal{L}(\theta_{\text{sup}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}^*) . \tag{4.8}$$

In the case of LDA, Loog [35] could show that this inequality is strict if the data is continuous. One desirable property of the CPL objective is its computational simplicity, i.e. the CPL scales linear in the number of classes, allowing us to learn the semi-supervised solution efficiently.



(a) Supervised SPN (b) Semi-supervised SPN

Figure 4.1: Illustration of a supervised and a semi-supervised SPN over K classes. The supervised SPN uses hard class labels, while the semi-supervised SPN uses soft labels (q) and hard labels (y) as inputs. In case hard labels are accessible, we marginalise out the soft labels and vice versa.

4.3 Learning Safe Semi-Supervised Sum-Product Networks

4.3.1 Generative Learning

After having revised the basics on CPL estimation, we will first extend the approach to semi-supervised learning of generative SPNs using EM. Subsequently, we will discuss how to utilise the approach for discriminative SPNs in which we will maximise the relative improvement of the conditional log-likelihood instead.

Similar to Loog [35], we can arrive at a CPL formulation by first considering the log-likelihood of the full dataset. For this, let us consider the SPN illustrated in Figure 4.1 (a), which we will call the supervised SPN. Further, let $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ denote the labelled training set, with $\|\boldsymbol{y}_i\|_0 \equiv 1$ being one-hot encoded class labels. Then the data log-likelihood of the supervised model is given as

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} w_k \, y_{i,k} \, \mathcal{S}_k(\boldsymbol{x}_i) \right) , \tag{4.9}$$

where w_k can be understood as the class prior for the k^{th} class and $\mathcal{S}_k$ is the k^{th} sub-network, illustrated using a triangle in Figure 4.1.

Now, let $\mathcal{U} = \{\boldsymbol{u}_j\}_{j=1}^M$ denote the set of unlabelled observations and let $\boldsymbol{q} = \{\boldsymbol{q}_j \in \Delta_{K-1}\}_{j=1}^M$ be the set of soft labels, as defined before. To incorporate unlabelled data with its associated soft labels, we will extend the model architecture in Figure 4.1 (a) to have additional numerical inputs representing the soft label assignments. By using the extended SPN architecture, illustrated in Figure 4.1 (b), we can obtain the data log-likelihood, jointly for labelled and unlabelled data, through marginalisation of either the hard labels ($\boldsymbol{y}$) or the soft labels ($\boldsymbol{q}$). The

resulting data log-likelihood can be written as

$$\begin{aligned}\mathcal{L}(\boldsymbol{w},\boldsymbol{\theta}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q}) = &\sum_{i=1}^{N}\log\Bigg(\overbrace{\sum_{k=1}^{K} w_k\, y_{i,k}\,\mathcal{S}_k(\boldsymbol{x}_i)}^{=\mathcal{S}(\boldsymbol{x}_i,\boldsymbol{y}_i)}\Bigg)\\ &+\sum_{j=1}^{M}\log\Bigg(\underbrace{\sum_{k=1}^{K} w_k\, q_{j,k}\,\mathcal{S}_k(\boldsymbol{u}_j)}_{=\mathcal{S}(\boldsymbol{u}_j,\boldsymbol{q}_j)}\Bigg)\,.\end{aligned} \tag{4.10}$$

Given the data log-likelihood term, the CPL objective for generative SPNs is defined as

$$\mathrm{CPL} = \min_{\boldsymbol{q}\in\Delta^M_{K-1}} \mathcal{L}(\boldsymbol{w},\boldsymbol{\theta}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q}) - \mathcal{L}(\boldsymbol{w}_{\mathrm{sup}},\boldsymbol{\theta}_{\mathrm{sup}}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q})\,. \tag{4.11}$$

Again, we can obtain the optimal parameters of the semi-supervised learner by maximising the CPL objective, i.e.

$$\boldsymbol{w}_{\mathrm{CPL}},\boldsymbol{\theta}_{\mathrm{CPL}} = \arg\max_{\boldsymbol{w},\boldsymbol{\theta}} \min_{\boldsymbol{q}\in\Delta^M_{K-1}} \mathcal{L}(\boldsymbol{w},\boldsymbol{\theta}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q}) - \mathcal{L}(\boldsymbol{w}_{\mathrm{sup}},\boldsymbol{\theta}_{\mathrm{sup}}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q})\,. \tag{4.12}$$

To learn both, parameters and soft labels, we incrementally update both by alternating between maximisation and minimisation. In particular, to find suitable network parameters we keep the soft labels fixed and maximise

$$\max_{\boldsymbol{w},\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{w},\boldsymbol{\theta}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q}) - \underbrace{\mathcal{L}(\boldsymbol{w}_{\mathrm{sup}},\boldsymbol{\theta}_{\mathrm{sup}}\,|\,\mathcal{D},\mathcal{U},\boldsymbol{q})}_{\text{constant}}\,, \tag{4.13}$$

in which the last term is a constant and can be ignored in the optimisation. Therefore, we can readily obtain the updates of the semi-supervised parameters and the respective expected sufficient statistics, when using EM for parameter learning. In particular, we compute the expected sufficient statistics for the weights using

$$\begin{aligned}r^{(t)}_{\mathsf{S},\mathsf{N}} = &\sum_{i=1}^{N}\frac{1}{\mathcal{S}(\boldsymbol{x}_i,\boldsymbol{y}_i)}\frac{\partial\mathcal{S}(\boldsymbol{x}_i,\boldsymbol{y}_i)}{\partial\mathsf{S}}w^{(t)}_{\mathsf{S},\mathsf{N}}\,\mathsf{N}(\boldsymbol{x}_i)\\ &+\sum_{j=1}^{M}\frac{1}{\mathcal{S}(\boldsymbol{u}_j,\boldsymbol{q}_j)}\frac{\partial\mathcal{S}(\boldsymbol{u}_j,\boldsymbol{q}_j)}{\partial\mathsf{S}}w^{(t)}_{\mathsf{S},\mathsf{N}}\,\mathsf{N}(\boldsymbol{u}_j)\,,\end{aligned} \tag{4.14}$$

and update the networks weights based on the computed sufficient statistics, i.e.

$$w^{(t+1)}_{\mathsf{S},\mathsf{N}} \leftarrow\!\shortmid \frac{r^{(t)}_{\mathsf{S},\mathsf{N}}}{\sum_{\mathsf{N}'\in\mathrm{ch}(\mathsf{S})} r^{(t)}_{\mathsf{S},\mathsf{N}'}}\,. \tag{4.15}$$

The sufficient statistic and the updates of the leaf node parameters (assuming distributions in the exponential family) can be found in a similar

fashion, i.e.

$$g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i, \boldsymbol{y}_i) = \frac{1}{\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \mathsf{L}} \mathsf{L}(\boldsymbol{x}_i \,|\, \theta_{\mathsf{L}}^{(t)}) \,, \tag{4.16}$$

$$g_{\mathsf{L}}^{(t)}(\boldsymbol{u}_j, \boldsymbol{q}_j) = \frac{1}{\mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)} \frac{\partial \mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial \mathsf{L}} \mathsf{L}(\boldsymbol{u}_j \,|\, \theta_{\mathsf{L}}^{(t)}) \,, \tag{4.17}$$

and

$$\theta_{\mathsf{L}}^{(t+1)} \hookleftarrow \frac{\sum_{i=1}^{N} g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i, \boldsymbol{y}_i) t(\boldsymbol{x}_i) + \sum_{j=1}^{m} g_{\mathsf{L}}^{(t)}(\boldsymbol{u}_j, \boldsymbol{q}_j) t(\boldsymbol{u}_j)}{\sum_{i=1}^{N} g_{\mathsf{L}}^{(t)}(\boldsymbol{x}_i, \boldsymbol{y}_i) + \sum_{j=1}^{M} g_{\mathsf{L}}^{(t)}(\boldsymbol{u}_j, \boldsymbol{q}_j)} \,. \tag{4.18}$$

To learn the soft labels, we fix the parameters of both networks and perform a single gradient descent step, as proposed by Loog [35], using the partial derivatives of Equation (4.11) w.r.t. the soft labels, i.e.

$$q_{j,k}^{(t+1)} \hookleftarrow q_{j,k}^{(t)} - \eta \nabla_{q_{j,k}^{(t)}} \,, \tag{4.19}$$

with

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial q_{j,k}} = \frac{w_k \, \mathcal{S}_k(\boldsymbol{u}_j)}{\sum_{k'=1}^{K} w_{k'} \, q_{j,k'} \, \mathcal{S}_{k'}(\boldsymbol{u}_j)} \,, \tag{4.20}$$

and

$$\nabla_{q_{j,k}} = \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial q_{j,k}} - \frac{\partial \mathcal{L}(\boldsymbol{w}_{\text{sup}}, \boldsymbol{\theta}_{\text{sup}} \,|\, \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial q_{j,k}} \,. \tag{4.21}$$

Note that after each gradient update it is necessary to ensure that the soft labels are still on the $K-1$ simplex. Therefore, we project the soft labels back to the simplex using the approach by Duchi, Shalev-Shwartz, Singer *et al.* [83] after each iteration. Alternatively, one can use a bijection from an unconstrained space to the $K-1$ simple and learn the unconstrained parameters using gradients descent [108].

4.3.2 Discriminative Learning

As described in Section 3.3.2, it is more natural to learn the parameters of a model for classification tasks by maximising the conditional log-likelihood. Therefore, we extend the approach by Loog [35] and maximise the relative improvement of the conditional log-likelihood instead of the (generative) log-likelihood, when training a discriminative semi-supervised SPN. The conditional log-likelihood function of the SPN shown in Figure 4.1 (b) is given as

$$\begin{aligned} \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}) = &\sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} y_{i,k} \, w_k \, \mathcal{S}_k(\boldsymbol{x}_i) \right) - \log \left(\sum_{k=1}^{K} w_k \, \mathcal{S}_k(\boldsymbol{x}_i) \right) \\ &+ \sum_{j=1}^{M} \log \left(\sum_{k=1}^{K} q_{j,k} \, w_k \, \mathcal{S}_k(\boldsymbol{u}_j) \right) - \log \left(\sum_{k=1}^{K} w_k \, \mathcal{S}_k(\boldsymbol{u}_j) \right) \,, \end{aligned} \tag{4.22}$$

which is equivalent to writing the conditional log-likelihood in the following more amenable form

$$\mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}, \mathcal{U}, \boldsymbol{q}) = \sum_{i=1}^{N} \log \underbrace{\left(\frac{\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\mathcal{S}(\boldsymbol{x}_i)} \right)}_{=\mathcal{S}(\boldsymbol{y}_i \mid \boldsymbol{x}_j)} + \sum_{j=1}^{m} \log \underbrace{\left(\frac{\mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)}{\mathcal{S}(\boldsymbol{u}_j)} \right)}_{=\mathcal{S}(\boldsymbol{q}_j \mid \boldsymbol{u}_j)} . \quad (4.23)$$

Then the safe semi-supervised model parameters for discriminative SPNs are estimated by maximising the CPL objective w.r.t. the difference of the conditional log-likelihood functions of both models, i.e.

$$\boldsymbol{w}_{\mathrm{CPL}}, \boldsymbol{\theta}_{\mathrm{CPL}} = \arg\max_{\boldsymbol{w}, \boldsymbol{\theta}} \min_{\boldsymbol{q} \in \Delta_{K-1}^{M}} \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}, \mathcal{U}, \boldsymbol{q}) - \mathcal{C}(\boldsymbol{w}_{\mathrm{sup}}, \boldsymbol{\theta}_{\mathrm{sup}} \mid \mathcal{D}, \mathcal{U}, \boldsymbol{q}) . \quad (4.24)$$

Similar to Gens and Domingos [20], we can approach the maximisation problem by employing gradient ascent. For this purpose, let the partial derivatives of the discriminative semi-supervised SPN w.r.t. the weight be given as

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial w_{\mathsf{S},\mathsf{N}}} = \frac{1}{\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial w_{\mathsf{S},\mathsf{N}}} - \frac{1}{\mathcal{S}(\boldsymbol{x}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i)}{\partial w_{\mathsf{S},\mathsf{N}}} , \quad (4.25)$$

and

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial w_{\mathsf{S},\mathsf{N}}} = \frac{1}{\mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)} \frac{\partial \mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial w_{\mathsf{S},\mathsf{N}}} - \frac{1}{\mathcal{S}(\boldsymbol{u}_j)} \frac{\partial \mathcal{S}(\boldsymbol{u}_j)}{\partial w_{\mathsf{S},\mathsf{N}}} . \quad (4.26)$$

Likewise, we can derive the partial derivatives w.r.t. the leaf node parameters, which are given as

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta_{\mathsf{L}}} = \frac{1}{\mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta_{\mathsf{L}}} - \frac{1}{\mathcal{S}(\boldsymbol{x}_i)} \frac{\partial \mathcal{S}(\boldsymbol{x}_i)}{\partial \theta_{\mathsf{L}}} , \quad (4.27)$$

and

$$\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial \theta_{\mathsf{L}}} = \frac{1}{\mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)} \frac{\partial \mathcal{S}(\boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial \theta_{\mathsf{L}}} - \frac{1}{\mathcal{S}(\boldsymbol{u}_j)} \frac{\partial \mathcal{S}(\boldsymbol{u}_j)}{\partial \theta_{\mathsf{L}}} . \quad (4.28)$$

After calculating the respective gradients based on the labelled and unlabelled data, i.e.

$$\begin{aligned} \nabla_{w_{\mathsf{S},\mathsf{N}}} = & \mathbb{E}_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim \mathcal{D}} \left[\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial w_{\mathsf{S},\mathsf{N}}} \right] \\ & + \mathbb{E}_{(\boldsymbol{u}_j, \boldsymbol{q}_j) \sim (\mathcal{U}, \boldsymbol{q})} \left[\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial w_{\mathsf{S},\mathsf{N}}} \right] , \end{aligned} \quad (4.29)$$

and

$$\begin{aligned} \nabla_{\theta_{\mathsf{L}}} = & \mathbb{E}_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim \mathcal{D}} \left[\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta_{\mathsf{L}}} \right] \\ & + \mathbb{E}_{(\boldsymbol{u}_j, \boldsymbol{q}_j) \sim (\mathcal{U}, \boldsymbol{q})} \left[\frac{\partial \mathcal{C}(\boldsymbol{w}, \boldsymbol{\theta} \mid \boldsymbol{u}_j, \boldsymbol{q}_j)}{\partial \theta_{\mathsf{L}}} \right] , \end{aligned} \quad (4.30)$$

we can applying gradient ascent to update the network parameters

$(\boldsymbol{\theta}, \boldsymbol{w})$ of the semi-supervised learner. Given a (small) learning rate η, the gradient updates take the form of

$$w_{\mathsf{S},\mathsf{N}}^{(t+1)} \leftarrow\!\shortmid w_{\mathsf{S},\mathsf{N}}^{(t)} + \eta \nabla_{w_{\mathsf{S},\mathsf{N}}^{(t)}} , \tag{4.31}$$

and

$$\theta_{\mathsf{L}}^{(t+1)} \leftarrow\!\shortmid \theta_{\mathsf{L}}^{(t)} + \eta \nabla_{\theta_{\mathsf{L}}^{(t)}} . \tag{4.32}$$

To pessimistically update the soft labels, we perform a single gradient descent step as described in Section 4.3.1 and project the updated soft labels back to the $K-1$ simplex. We will now discuss a concrete realisation of the proposed technique and examine the implications of learning semi-supervised SPNs by maximising the CPL objective.

4.3.3 Learning Maximum Contrastive Pessimistic Sum-Product Networks

To learn safe semi-supervised SPNs, we employ Algorithm 2 in Appendix B, which returns a Maximum Contrastive Pessimistic SPN (MCP-SPN). The key steps of learning MCP-SPNs are i) maximising the respective CPL objective w.r.t. the parameters of the safe semi-supervised SPN, i.e. Equation (4.11) or Equation (4.23), and ii) minimising the objective w.r.t. the soft labels. Those two steps are optimised in an alternating fashion, resulting in an approximate solution to Equation (4.12) and Equation (4.24), respectively.

In case the dataset is discrete, i.e. the leaf nodes are indicator functions, or only the weights of the safe-semi supervised SPN are learned, the objective is a multi-linear function in the parameters we want to optimise, i.e. the weights of the SPN. Therefore, the min-max objective in Equation (4.12) and Equation (4.24) is guaranteed to result in a saddle point solution [109]. In these cases, the semi-supervised SPN will improve upon the supervised SPN (if the SPN contains sum nodes with multiple children and $\mathcal{U} \neq \emptyset$) in expectation, i.e.

$$\mathbb{E}[\mathcal{L}(\boldsymbol{w}_{\mathrm{CPL}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}^*)] > \mathbb{E}[\mathcal{L}(\boldsymbol{w}_{\mathrm{sup}} \,|\, \mathcal{D}, \mathcal{U}, \boldsymbol{q}^*)] , \tag{4.33}$$

where the expectation is over the unlabelled data and $\boldsymbol{q}^*$ denotes the true labelling of the unlabelled data. Note that we can only guarantee a strict improvement over the supervised solution in the expectation, as the probability that $\boldsymbol{w}_{\mathrm{CPL}}$ is different from $\boldsymbol{w}_{\mathrm{sup}}$ is non-zero. This may imply that it is preferable to learn SPN parameters by maximising the CPL objective instead of purely supervised parameter learning.

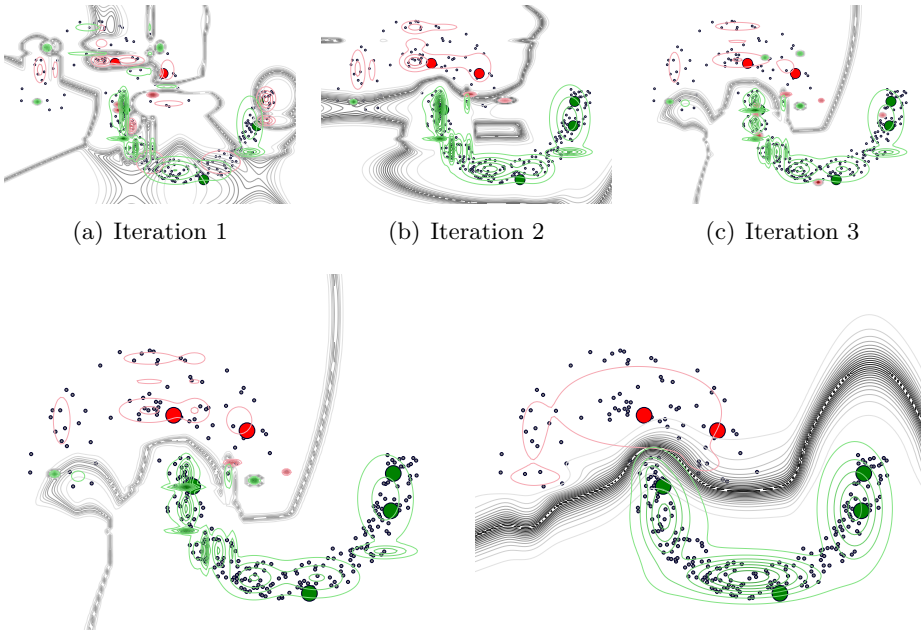
4.4 Experiments

To assess the performance of safe semi-supervised SPNs, we will first examine the results obtained using safe semi-supervision qualitatively and later provide quantitative results on benchmark datasets. For the quantitative experiments, we pre-processed the data by removing zero-

variance features and applying z-score normalisation. To learn SPN structures for each experiment, we use the learnSPN [36] algorithm with k-Means clustering for sum nodes and the Hilbert-Schmidt Independence Criterion (HSIC) with Gamma approximation [110] to construct product nodes. Finally, we added a layer to model the class conditionals, as illustrated in Figure 4.1. Note that learnSPN produces deep SPT structures, which might be prone to overfitting. Therefore, we use a maximum depth to restrict the model complexity selected using the Akaike Information Criterion (AIC) [111] for generative learning and based on the performance on the validation set in case of discriminative learning.

4.4.1 Qualitative Experiments

To assess the implications of the additional flexibility induced by (non-linear) SPNs on the performance of the semi-supervised learner obtained through maximisation of the CPL objective, we first performed a qualitative experiment on the synthetic two moons dataset [112]. Figure 4.2 compares the classification boundary obtained using safe semi-supervised learning of SPN to the classification boundary of the supervised model trained on the fully labelled dataset (Oracle SPN).



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

(d) Final SSL SPN (e) Oracle SPN

Figure 4.2: Qualitative results on the two moons dataset. Class labels are indicated in red and green, respectively, as well as the estimated density regions of the classes. The labelled training examples are shown using large red and greed dots, respectively, and unlabelled examples are shown in grey. The decision boundary of each model is illustrated using a contour plot coloured in grey.

Figure 4.2 indicates that safe semi-supervised SPNs learn a more conservative decision boundary than supervised SPNs, i.e. the transition from one class to another is less steep than in the case of the oracle SPN. Further, we can see that despite the additional flexibility and

degrees of freedom, we obtain a remarkably effective semi-supervised learner in only a few iterations, c.f. upper row of Figure 4.2. Note that the soft labels have been initialised using random draws from a Dirichlet distribution, explaining the random behaviour at the first iteration.

4.4.2 Quantitative Experiments

In addition to the qualitative assessment of safe semi-supervised SPNs, we compared their performance quantitatively against common semi-supervised techniques on a collection of UCI datasets [113]. Details on the selected datasets are shown in Table 4.1, where the last column indicates the number of labelled examples used in the experiments; calculated as proposed by Loog [35]. For each experiment, we split each dataset into training (80%) and test set (20%) and randomly draw $2\,D+K$ many labelled observations stratified from the training set. Further, we used an additional labelled validation set of $2\,D + K$ observations for early stopping, and treated all remaining observations as unlabelled data.

Dataset	Obs. (N)	Dims. (D)	Classes (K)	$2\,D + K$
BUPA	345	6	2	14
Fertility	100	9	2	20
Haberman	306	3	2	8
ILPD	583	10	2	22
Ionosphere	351	34	2	70
Iris	150	4	3	11
Parkinsons	197	23	2	48
WDBC	569	32	2	66
Wine	178	13	3	29

Table 4.1: Datasets used for qualitative experiments: number of observations (N), dimensionality (D), number of classes (K) and number of labelled examples ($2\,D + K$).

Generative Learning

We first examined the performance of safe semi-supervised learning in the generative setting. For this purpose, we constructed SPN structures with Gaussian leaves as described before, and initialised all soft labels using random draws from a Dirichlet distribution with uniform concentration parameter. To prevent degeneration of the leaves, we lower bounded the variance of each Gaussian distribution. The lower bound was chosen according to the first percentile with a value above zero computed for the pairwise distances between all observations. This approach ensures that the lower bound has minimal influence on the model expressiveness for the given dataset.

Table 4.2 lists the average test log-likelihood for supervised SPNs, safe semi-supervised SPNs (MCP-SPN), maximum contrastive pessimistic LDA (MCP-LDA) [35], and the performance of an SPN trained on the fully labelled training set (Oracle SPN). To reduce the influence of the randomly chosen labelled dataset on the comparison, we estimated

the average log-likelihood scores over 100 independent runs. We observe that generative semi-supervised parameter learning improves the test log-likelihood in most cases. In the case of *Parkinsons*, *WDBC*, and *Wine*, the purely supervised learner finds a solution close to the oracle solution, while the semi-supervised approach struggles to improve upon the supervised solution. However, on datasets such as *Fertility*, and *Haberman*, the MCP-SPN excels in performance and reaches near oracle performance. Furthermore, we see that safe semi-supervised SPNs generally outperform MCP-LDA, which is not too surprising given that the MCP-LDA is a linear classification model while MSP-SPNs can learn complex non-linear decision boundaries. Last but not least, we want to emphasise that MCP-SPNs generally obtain standard errors comparable to those of the oracle network, indicating that the approach is sufficiently robust against the choice of the labelled subset.

Dataset	Supervised SPN	MCP-SPN	MCP-LDA	Oracle SPN
BUPA	-438.75 ± 7.00	$\mathbf{-7.31} \pm 0.06$	-9.07 ± 0.03	-8.80 ± 0.18
Fertility	-3.31 ± 0.03	$\mathbf{-3.06} \pm 0.01$	-12.68 ± 0.05	-3.00 ± 0.01
Haberman	-138.63 ± 4.00	$\mathbf{-5.05} \pm 0.07$	-7.83 ± 0.10	-5.14 ± 0.06
ILPD	-5.62 ± 3.00	$\mathbf{-1.15} \pm 0.02$	-37.54 ± 0.10	-1.00 ± 0.01
Ionosphere	-2.84 ± 0.05	$\mathbf{-1.61} \pm 0.01$	-46.12 ± 0.05	-1.52 ± 0.01
Iris	-20.65 ± 0.85	-3.78 ± 0.03	$\mathbf{-2.65} \pm 0.05$	-2.17 ± 0.01
Parkinsons	$\mathbf{-1.32} \pm 0.01$	-1.34 ± 0.00	-2.27 ± 0.05	-1.30 ± 0.00
WDBC	$\mathbf{-1.90} \pm 0.00$	-1.93 ± 0.00	-10.75 ± 0.01	-1.88 ± 0.00
Wine	$\mathbf{-2.47} \pm 0.00$	$\mathbf{-2.47} \pm 0.00$	-15.28 ± 0.02	-2.44 ± 0.00

Table 4.2: Average test log-likelihood and standard errors estimated over 100 independent trials. The best results for each dataset obtained by a supervised or semi-supervised model are shown in ***bold****. The oracle solution (trained on the fully labelled dataset) is shown for comparison.*

MNIST Experiments

In addition to the comparison with MCP-LDA, we performed an experiment comparing purely supervised SPNs and self-learning in SPNs against MCP-SPNs on the well-known MNIST [114] dataset. For reasons of efficiency, we used the recent SPN architecture proposed by Peharz, Lang, Vergari *et al.* [13] and constructed a region graph structure based on the approach of Poon and Domingos [8] with binomial distributions as leaves for each class. Each model was trained using generative learning by leveraging stochastic EM, as described in [13], using the first N examples as labelled data and the remaining as unlabelled data. The test log-likelihood scores for a varying amount of label information is shown in Figure 4.3.

We see that MCP-SPNs consistently outperform supervised learning and self-learning in case only a few observations are labelled. With an increase of labelled data, self-learning tends to obtain competitive results. This behaviour is to be expected, as the decision boundary of the supervised learner will become more accurate with an increasing amount of labelled data. Therefore, resulting in better predictions for the unlabelled examples in self-learning.

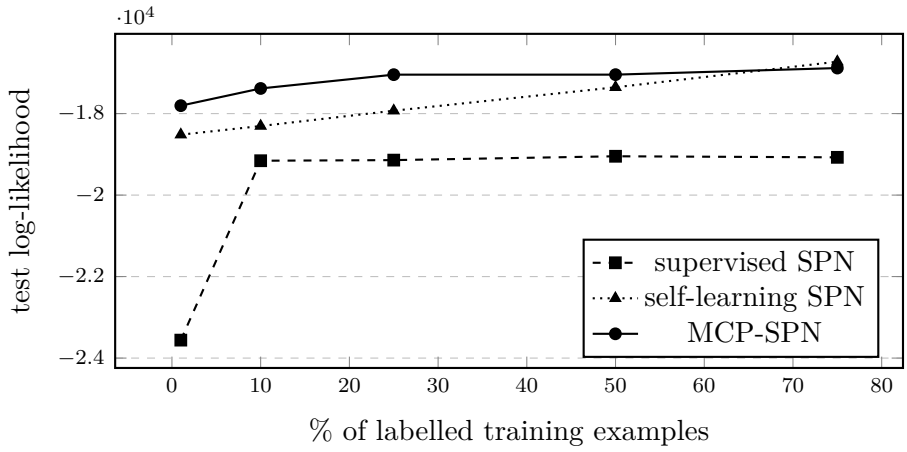


Figure 4.3: Test log-likelihood scores on MNIST for varying number of labelled observations.

Discriminative Learning

As previously discussed, discriminative SPNs are a more natural approach to learn an SPN in case of a classification problem. Therefore, we compared the performance of our discriminative approach against supervised discriminative SPNs, and the following discriminative semi-supervised learning techniques: Transductive SVM (TSVM) [32], Minimum Entropy Regularisation (MER) [115] and the Implicitly Constrained Least Squares (ICLS) [116]. To assess the performance of each method, we computed the F_1 score for binary classification tasks. In cases of multi-class datasets, we used the macro average[20] F_1 score on the test set, i.e.

F_1 score

$$F_1(\boldsymbol{y}_k, \hat{\boldsymbol{y}}_k) = 2\frac{f_p\, f_r}{f_p + f_r} \qquad f_p \coloneqq \frac{\|\boldsymbol{y}_k \odot \hat{\boldsymbol{y}}_k\|_0}{\underbrace{\|\hat{\boldsymbol{y}}_k\|_0}_{=\sum_i \mathbb{1}_{\{\hat{y}_{i,k}\neq 0\}}}} \tag{4.34}$$

$$f_r \coloneqq \frac{\|\boldsymbol{y}_k \odot \hat{\boldsymbol{y}}_k\|_0}{\sum_i y_{i,k}\, \mathbb{1}_{\{y_{i,k}=\hat{y}_{i,k}\}} + (1 - y_{i,k})\, \mathbb{1}_{\{y_{i,k}\neq\hat{y}_{i,k}\}}}\,, \tag{4.35}$$

where $\odot$ denotes an element-wise multiplication, and the one-hot encoded vectors $\boldsymbol{y}_k \in \mathbb{B}^{N_{\text{test}}}$ and $\hat{\boldsymbol{y}}_k \in \mathbb{B}^{N_{\text{test}}}$ denote the true and the predicted labels for the k^{th} class, respectively. To compute multi-class predictions for approaches designed only for binary classification, we used the one-vs-rest approach [117]. Similar to the quantitative evaluation of the generative approach, we estimated the performance of each method over 100 independent trails. Table 4.3 lists the average F_1 scores for each approach and additionally shows the oracle performance for comparison.

We see that safe semi-supervised learning in SPNs achieves competitive results for most datasets, and in some cases MCP-SPNs obtain a test F_1 score comparable to the performance of the oracle solution, e.g. *Wine*. The F_1 scores on *Fertility*, *Haberman* and *ILPD* are generally very low, likely as a result of imbalanced classes. In general, the experiments indicate that safe semi-supervised learning of discriminative SPNs can achieve competitive results, even without imposing strong assumptions

[20] The macro average is computed as the arithmetic mean of the class-wise F_1 scores.

on the data geometry. We want to note that, even though the CPL objective provides guarantees on the training set, those guarantees do not apply to the test set if used in an inductive setting. In all experiments we have treated MCP-SPNs as an inductive classifier, i.e. the learned semi-supervised classifier predicts the labels of the unseen test examples solely based on the learned CPL model parameters. However, MCP-SPN can also be learned in a transductive setting by jointly learning soft labels for the test and the unlabelled data. Therefore, allowing us to express guarantees on the whole dataset in exchange of higher computational costs, i.e. the CPL objective has to be maximised each time we observe new test examples.

Data Set	Supervised SPN	MCP-SPN	TSVM	ICLSC	MER	Oracle SPN
BUPA	0.41 ± 0.01	0.40 ± 0.01	0.36 ± 0.02	$\mathbf{0.47} \pm 0.01$	0.42 ± 0.01	0.48 ± 0.01
Fertility	0.07 ± 0.02	0.03 ± 0.01	0.07 ± 0.02	0.07 ± 0.02	$\mathbf{0.12} \pm 0.02$	0.06 ± 0.02
Haberman	0.24 ± 0.02	0.28 ± 0.02	0.20 ± 0.02	$\mathbf{0.33} \pm 0.01$	0.27 ± 0.02	0.25 ± 0.00
ILPD	0.17 ± 0.02	0.20 ± 0.02	0.23 ± 0.02	0.29 ± 0.01	$\mathbf{0.33} \pm 0.02$	0.25 ± 0.00
Ionosphere	0.79 ± 0.00	$\mathbf{0.82} \pm 0.00$	0.66 ± 0.01	0.61 ± 0.01	0.70 ± 0.01	0.87 ± 0.00
Iris	0.73 ± 0.01	$\mathbf{0.88} \pm 0.01$	0.72 ± 0.01	0.74 ± 0.02	0.81 ± 0.01	0.93 ± 0.00
Parkinsons	0.72 ± 0.01	$\mathbf{0.77} \pm 0.00$	0.74 ± 0.01	0.67 ± 0.02	0.68 ± 0.01	0.82 ± 0.00
PID	0.38 ± 0.01	0.45 ± 0.01	0.46 ± 0.01	0.54 ± 0.01	$\mathbf{0.57} \pm 0.01$	0.64 ± 0.00
WDBC	0.85 ± 0.00	0.90 ± 0.00	0.91 ± 0.00	0.88 ± 0.00	$\mathbf{0.92} \pm 0.00$	0.92 ± 0.00
Wine	0.82 ± 0.01	$\mathbf{0.97} \pm 0.00$	$\mathbf{0.97} \pm 0.00$	0.95 ± 0.01	0.95 ± 0.01	0.97 ± 0.00

*Table 4.3: Macro-average test F_1 scores, estimated over 100 independent trials. The best results are indicated in **bold**.*

5 Bayesian Learning of Sum-Product Networks

5.1 Motivation

As discussed in Chapter 3, learning SPNs can naturally be organised into *structure* and *parameter learning*. We have seen that state-of-the-art SPN parameter learning covers a wide range of well-developed techniques, including approaches for unsupervised, supervised and semi-supervised tasks. Concerning structure learning, however, the situation is remarkably different. Although there exist a plethora of approaches, we have seen in Section 3.5 that most approaches can be described as heuristic. The existing techniques often represent intuitive schemes for structure learning, but fall short on declaring the *global goal of structure learning in SPNs*. This is surprising as it should be of uttermost importance to first ask either of the following fundamental questions: *What is a good SPN structure?* or *What is a good principle to derive an SPN structure?* The literature on probabilistic graphical models, on the other hand, offers a rich set of learning principles, with the main strategy being the optimisation of a *structure score* such as the Minimum-Description-Length (MDL) [118], the Bayesian Information Criterion (BIC) [119] or the Bayes-Dirichlet (BD) score [120], [121]. Moreover, Friedman and Koller [122] introduced an approximate, but asymptotically correct, MCMC sampler for Bayesian structure learning of Bayesian networks.

In this chapter, we will introduce a well-principled Bayesian approach to learn SPNs by simultaneously performing inference over both structure and parameters. We will leverage the decomposition of SPNs into a computational graph and a scope-function, as introduced in Section 3.2.1, and introduce a natural parameterisation for scope-functions applicable to tree-structured computational graphs. In particular, we will focus on so-called *tree-shaped region graphs*, which have been widely used in prior art, e.g. [10], [76]. This restriction allows an elegant encoding of the scope-function via categorical variables. As a consequence, Bayesian learning in SPNs becomes conceptually simple. In particular, we equip all parameters and latent variables, including the additional categorical variables for the scope-function, with appropriate priors and perform approximate Bayesian inference to estimate the posterior distribution of Bayesian SPNs. We will incrementally build up our Bayesian framework for principled learning in SPN by, first revising Bayesian parameter learning with a fixed scope-function, as introduced in [17]–[19], and subsequently introducing our fully Bayesian approach.

5.2 Preliminaries

Recall from Section 3.1.1, that sum nodes in an SPN can be interpreted as latent variables. Thus, we can understand an SPN as a compact representation of an exponentially large mixture over induced trees, c.f. Equation (3.5). Further recall that in this context $T(\boldsymbol{z}_i) = \mathcal{T}$ denotes a surjective map, which returns the induced tree determined by $\boldsymbol{z}_i$ for any value of $\boldsymbol{z}_i$.

Bayesian parameter learning

By equipping the sum weights and the leaf parameters with suitable prior distributions, we can easily derive the generative model of an SPN for Bayesian parameter learning. In this work, we will assume sum weights to be Dirichlet distributed and use, out of convenience, conjugate priors[21] for the parameters of the leaves. For ease of notation, we denote the appropriate prior for θ_{L} using $p(\theta_{\mathsf{L}} \mid \gamma)$, where γ compromises the parameters of the prior distribution. The resulting generative model for Bayesian parameter learning is written as

$$\begin{aligned} \boldsymbol{w}_{\mathsf{S}} &\sim \mathrm{Dir}(\boldsymbol{w}_{\mathsf{S}} \mid \alpha_1, \dots, \alpha_{K_{\mathsf{S}}}) && \forall \mathsf{S} \\ z_{\mathsf{S}} &\sim \mathrm{Cat}(z_{\mathsf{S}} \mid \boldsymbol{w}_{\mathsf{S}}) && \forall \mathsf{S} \\ \theta_{\mathsf{L}} &\sim p(\theta_{\mathsf{L}} \mid \gamma) && \forall \mathsf{L} \\ \boldsymbol{x} &\sim \prod_{\mathsf{L} \in T(\boldsymbol{z})} p(\boldsymbol{x}_{\mathsf{L}} \mid \theta_{\mathsf{L}}) \,. \end{aligned} \tag{5.1}$$

To learn an SPN using this generative model, we will employ Bayes' rule, c.f. Equation (3.12). In our previous approaches, c.f. Section 3.3, we have used a single estimate of our model parameters to perform future predictions. However, the correct parameters are usually unknown, and we should, therefore, account for the uncertainty about the model parameters. The conventional approach to account for our uncertainty is to integrate over all model parameters in the conditional distribution $p(\boldsymbol{x}^*, \boldsymbol{w}, \boldsymbol{\theta} \colon \mathcal{D})$, where $\boldsymbol{x}^*$ denotes an unseen datum, and perform predictions using the resulting marginal distribution. The resulting distribution is called the posterior predictive distribution and is given as

posterior predictive distribution

$$p(\boldsymbol{x}^* \mid \mathcal{D}) = \int_{\boldsymbol{w}} \int_{\boldsymbol{\theta}} p(\boldsymbol{x}^* \mid \boldsymbol{w}, \boldsymbol{\theta}, \mathcal{D})\, p(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}) \,\mathrm{d}\boldsymbol{w} \,\mathrm{d}\boldsymbol{\theta} \tag{5.2}$$

$$= \int_{\boldsymbol{w}} \int_{\boldsymbol{\theta}} p(\boldsymbol{x}^* \mid \boldsymbol{w}, \boldsymbol{\theta})\, p(\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}) \,\mathrm{d}\boldsymbol{w} \,\mathrm{d}\boldsymbol{\theta} \tag{5.3}$$

$$= \mathbb{E}_{\boldsymbol{w}, \boldsymbol{\theta} \mid \mathcal{D}} \left[p(\boldsymbol{x}^* \mid \boldsymbol{w}, \boldsymbol{\theta}) \right] , \tag{5.4}$$

where we assume that $\boldsymbol{x}^*$ is independent of $\mathcal{D}$. Note that the last line follows from the *law of the unconscious statistician* [123]. However, computing these integrals is often intractable, and we have to rely on approximations through either: variational inference, as done in [19], or Monte Carlo integration, as shown in [17]. In the course of this thesis, we will focus on the latter and leverage Markov chain Monte Carlo (MCMC) sampling to approximate the posterior predictive distribution. We refer to Murphy [124] for details on variational inference. In MCMC, the

Markov chain Monte Carlo

[21] A prior $p(\cdot)$ is conjugate for a likelihood if the posterior of $p(\cdot)$ is in the same family of distributions as $p(\cdot)$. Note that conjugate prior distributions are often used out of convenience as posterior inference simplifies in case of conjugacy.

integration problem is approximated using Monte Carlo integration, using samples drawn from the unnormalised posterior distribution. Monte Carlo integration aims to numerically integrate a function $f(X)$ of RV X, in order to compute moments of interest, e.g. the expected value

Monte Carlo integration

$$\mathbb{E}[X] = \int_x f(x)p(x)\,\mathrm{d}x \approx \frac{1}{S}\sum_{s=1}^{S} f(x_s)\,, \tag{5.5}$$

where we sample $x_s \sim p(X)$ accordingly. Note that Monte Carlo integration follows as a consequence of the *law of large numbers*, i.e. with $S \to \infty$ it is ensured that the sample mean $\frac{1}{S}\sum_{s=1}^{S} f(x_s)$ will converge to the expected value. However, as the posterior distribution is often not accessible in closed-form, we need to employ an algorithm to generate samples of our target distribution. A prominent class of algorithms, which construct (correlated) samples from the unnormalised posterior distribution in the form of a Markov chain, is called MCMC algorithms. Note that the constructed Markov chain has $p(X)$ as its stationary distribution. We will discuss a concrete example of an MCMC algorithm in Section 5.4.

5.3 Bayesian Sum-Product Networks

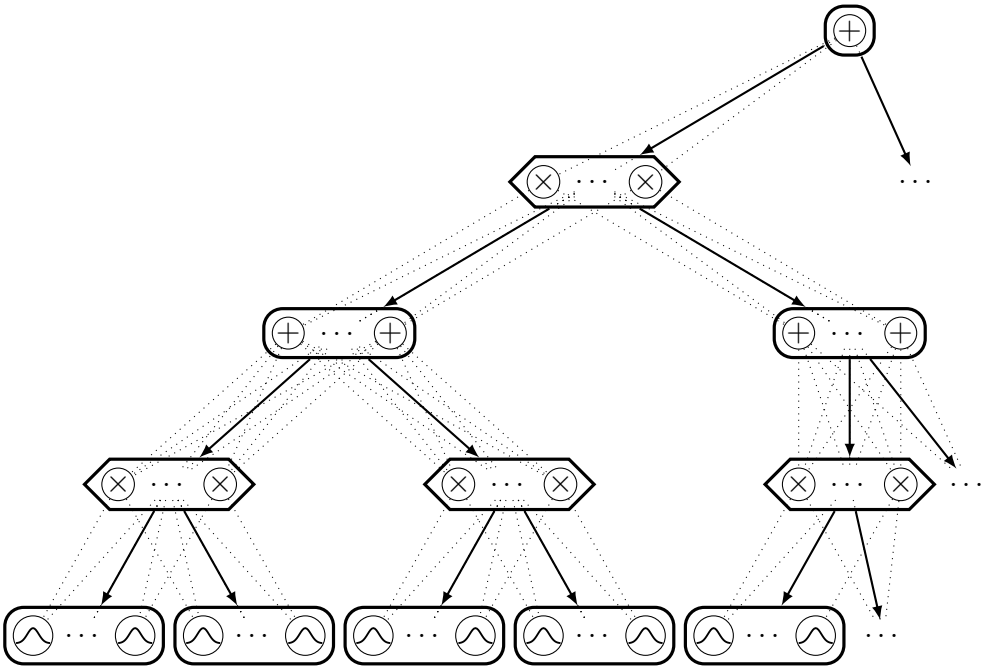


Figure 5.1: Illustration of a computational graph in form of a tree-shaped region graph of depth two. Regions contain a collection of sum nodes, while partitions contain all possible cross-products (Cartesian product) over the nodes in the respective child regions. Atomic regions contain a collection of leaf nodes with different parameters. Dotted lines illustrate edges in the induced SPN.

Given a computational graph $\mathcal{G}$, we wish to learn the scope-function ψ, in addition to the SPN's parameters $\boldsymbol{w}$ and $\boldsymbol{\theta}$. In general graphs,

representing ψ in an amenable form is somewhat involved as it requires ψ to ensure consistency of the scope assignments among nodes that share some of their children. Therefore, we will restrict our analysis to the class of SPNs whose computational $\mathcal{G}$ follows a *tree-shaped region graph*, i.e. each node in the region graph has at most one parent, as illustrated in Figure 5.1. Assuming a tree-shaped region graph leads to a natural encoding of ψ through additional latent variables. Recall, that a region graph is a tuple $(\mathcal{R}, \psi)$ where $\mathcal{R}$ is a connected DAG containing regions R and partitions P, c.f. Definition 3.11.

For a given region graph, the resulting SPN will inherit its scope-function ψ from the scope-function of the region graph. In particular, any SPN node introduced for a region (partition) gets the same scope as the region (partition) itself. It is easy to check that if the SPN's $\mathcal{G}$ follows $\mathcal{R}$, any proper scope-function corresponds to a proper scope-function of the SPN. Note that $\mathcal{G}$ is in general *not* tree-shaped, even if $\mathcal{R}$ is tree-shaped, with the exception that every region contains only a single node.

When the SPN follows a tree-shaped region graph, the scope-function can be encoded as follows: Let P be any partition and $R_1, \dots, R_{K_P}$ be its K_P children. For each data dimension $d \in \{1, \dots, D\}$, we introduce a discrete latent variable $Y_{P,d}$ with $1, \dots, K_P$ states. Intuitively, the latent variable $Y_{P,d}$ represents a decision to assign dimension d to a particular child, given that all partitions "above" have decided to assign d onto the path leading to P. Note that this path is unique for tree-shaped region graphs.

More formally, we define the scope-function of a tree-shaped region graph induced by a configuration $\boldsymbol{y}$ of $\boldsymbol{Y}$ as follows:

induced scope-function

Definition 5.1 (Induced scope-function)**.** *Let $\mathcal{R}$ be a tree-shaped region graph structure, and let $Y_{P,d}$ be defined as above. Further, let $\boldsymbol{Y} = \{Y_{P,d}\}_{P\in\mathcal{R},d\in\{1\dots D\}}$, and let $\boldsymbol{y}$ be any assignment for $\boldsymbol{Y}$. Let Q be a node in $\mathcal{R}$, then there exists a unique path Π from the root to Q (excluding Q). Therefore, the scope-function induced by $\boldsymbol{y}$ is defined as:*

$$\psi(Q)_{\boldsymbol{y}} \coloneqq \left\{ X_d \,\middle|\, \prod_{P\in\Pi} \mathbb{1}_{\{R_{y_{P,d}}\in\Pi\}} \right\}, \tag{5.6}$$

where $\prod_{P\in\Pi} \equiv 1$ if $\Pi = \emptyset$. From Equation (5.6) we can see that $\psi(Q)_{\boldsymbol{y}}$ contains X_d if for each partition in Π the child indicated by $y_{P,d}$ is also in Π.

Lemma 5.1. *For any tree-shaped region graph $\mathcal{R}$ and any assignments $\boldsymbol{y}$, the induced scope-function $\psi(\cdot)_{\boldsymbol{y}}$ is a scope-function according to Definition 3.11. Conversely, for any proper scope-function, there exists a $\boldsymbol{y}$ such that $\psi(\cdot)_{\boldsymbol{y}} \equiv \psi(\cdot)$.*

Proof. To proof the lemma, we will examine the individual requirements (c.f. Definition 3.11) one after another.

(1.) By Definition 5.1, each node Q that does not have any partition node in its unique path from the root has full scope, i.e. $\psi(Q)_{\boldsymbol{y}} = \boldsymbol{X}$. Since the root node is such a node, it has full scope.

(2.) For each child Q of partition P, the scope of Q is solely determined by the latent variables on the path Π from the root to Q. As $\mathcal{R}$ is a tree, the scope of P is determined trough the same latent variables, excluding the latent variable at P. Thus, we have $\psi(Q)_{\boldsymbol{y}} \subseteq \psi(P)_{\boldsymbol{y}}$, $\bigcup_{Q \in \text{ch}\, P} \psi(Q)_{\boldsymbol{y}} = \psi(P)_{\boldsymbol{y}}$ and $\bigcap_{Q \in \text{ch}\, P} \psi(Q)_{\boldsymbol{y}} = \emptyset$.

(3.) For each child Q of region R, the scope of Q is solely determined by all latent variables on the path Π from the root to Q. As $\mathcal{R}$ is a tree, the scope of R is determined trough the same latent variables and thus their scopes have to be the same, i.e. $\psi(Q)_{\boldsymbol{y}} = \psi(R)_{\boldsymbol{y}}$ and $\bigcup_{Q \in \text{ch}\, R} \psi(Q)_{\boldsymbol{y}} = \psi(R)_{\boldsymbol{y}}$.

□

Note that the relationship between $\boldsymbol{y}$ and $\psi(\cdot)_{\boldsymbol{y}}$ is similar to the relationship between $\boldsymbol{z}$ and $\mathcal{T}$, i.e. each $\psi(\cdot)_{\boldsymbol{y}}$ corresponds in general to many $\boldsymbol{y}$'s. Also note, that the encoding of the scope-function for general region graphs is more involved, since the path to each Q is not unique, and we need to ensure that consistency of the scope assignment, for all nodes that are shared throughout the network, is guaranteed. Assuming Dirichlet priors for each $Y_{P,d}$, we can define the extended generative model as:

$$
\begin{aligned}
\boldsymbol{w}_{\mathsf{S}} &\sim \text{Dir}(\boldsymbol{w}_{\mathsf{S}} \,|\, \alpha_1, \ldots, \alpha_{K_{\mathsf{S}}}) && \forall \mathsf{S} \\
z_{\mathsf{S}} &\sim \text{Cat}(z_{\mathsf{S}} \,|\, \boldsymbol{w}_{\mathsf{S}}) && \forall \mathsf{S} \\
\boldsymbol{v}_P &\sim \text{Dir}(\boldsymbol{v}_P \,|\, \beta_1, \ldots, \alpha_{K_{\mathsf{P}}}) && \forall P \\
y_{P,d} &\sim \text{Cat}(y_{P,d} \,|\, \boldsymbol{v}_P) && \forall d = 1, \ldots, D \;\; \forall P \\
\theta_{\mathsf{L},d} &\sim p(\theta_{\mathsf{L},d} \,|\, \gamma) && \forall d = 1, \ldots, D \;\; \forall \mathsf{L} \\
\boldsymbol{x} &\sim \prod_{\mathsf{L} \in T(\boldsymbol{z})} p(\boldsymbol{x}_{\boldsymbol{y}} \,|\, \theta_{\mathsf{L},\boldsymbol{y}}) \,.
\end{aligned}
\tag{5.7}
$$

The notation $\boldsymbol{x}_{\boldsymbol{y}}$ denotes the evaluation of L on the scope induced by $\boldsymbol{y}$. Figure 5.2 illustrates our generative model in plate notation in which directed edges indicate dependencies between variables.

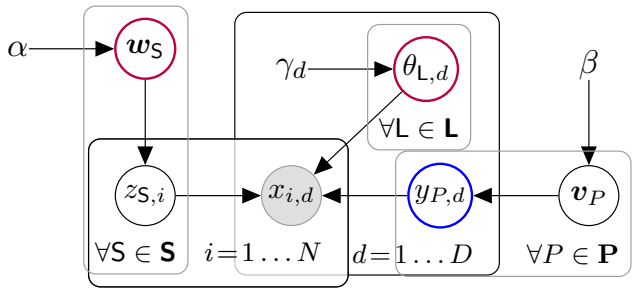


Figure 5.2: Plate notation of the generative model of Bayesian sum-product networks. Network parameters are highlighted in purple and latent variables encoding the scope-function are shown in blue.

5.4 Sampling-based Inference

Let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ be a set of observations, given our generative process in Equation (5.7), the posterior distribution of Bayesian SPNs can be written as

$$\begin{aligned} p(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y}, \boldsymbol{\theta} \mid \mathcal{D}) &= \frac{p(\mathcal{D} \mid \boldsymbol{w}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y}, \boldsymbol{\theta})\, p(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y}, \boldsymbol{\theta})}{p(\mathcal{D})} \\ &\propto p(\mathcal{D} \mid \boldsymbol{w}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y}, \boldsymbol{\theta})\, p(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{y}, \boldsymbol{\theta})\,. \end{aligned} \tag{5.8}$$

Gibbs sampling

To obtain samples from Equation (5.8) we will perform Gibbs sampling, which is one of the most popular MCMC algorithms and similar to coordinate ascent. The central idea behind Gibbs sampling is to update one parameter at a time, while conditioning on the state of all other parameters. Thus, Gibbs sampling constructs a Markov chain using axis-aligned moves and cannot exploit correlations between parameters well. However, by exploiting conjugacy relationships we can marginalise out model parameters. Such an approach reduces the variance of the sampler, thus increasing the effectiveness of the sampling, and is referred to as collapsed Gibbs sampling.

collapsed Gibbs sampling

We will now discuss our Gibbs sampling scheme, which alternates between i) updating the model parameters $\boldsymbol{w}, \boldsymbol{\theta}$ for a fixed encoding of the scope-function $\boldsymbol{y}$, and ii) updating the scope-function encoding $\boldsymbol{y}$ for a fixed set of parameters $\boldsymbol{w}, \boldsymbol{\theta}$. Note that the model parameters and the scope-function encoding may be initialised using random draws from their respective prior distribution or using a heuristic method such as k-means clustering.

5.4.1 Updating the Parameters

To update the model parameters, we follow the same procedure as introduced by Vergari, Molina, Peharz *et al.* [17], i.e. we first sample all latent assignments $\boldsymbol{z}_i$, which can be done in parallel; and subsequently updated all model parameters given the latent assignments. In particular, for a given set of parameters ($\boldsymbol{w}$ and $\boldsymbol{\theta}$), we draw each $\boldsymbol{z}_i$ independently using ancestral sampling in SPN, i.e. we sample $z_{\mathsf{S},i} \sim \sum_{\mathsf{N} \in \operatorname{ch} \mathsf{S}} w_{\mathsf{S},\mathsf{N}} \mathsf{N}(\boldsymbol{x}_i)$. Latent variables that are not visited during ancestral sampling will be drawn from the prior, i.e. $z_{\mathsf{S},i} \sim \operatorname{Cat}(z_{\mathsf{S},i} \mid \boldsymbol{w}_{\mathsf{S}})$. Once all $\boldsymbol{z}_i$ have been drawn, we can sample the weights and leaf node parameters from their posterior distribution. As we have chosen prior distributions for the leaf and sum node parameters from a conjugate family, we can obtain an analytic expression for their posterior distribution. Doing so allows us to generate all model parameters by sampling from the analytic form of their posterior distribution. In particular, we sample the weights for each sum node from the following posterior,

$$\boldsymbol{w}_{\mathsf{S}} \sim \operatorname{Dir}(\boldsymbol{w}_{\mathsf{S}} \mid \alpha + c_{\mathsf{S},1}, \ldots, \alpha + c_{\mathsf{S},K_{\mathsf{S}}})\,, \tag{5.9}$$

where $c_{\mathsf{S},k}$ represents the number of observations that have chosen the k^{th} child under S during ancestral sampling. In the case of leaf nodes, we use the respective posterior distribution as listed in Appendix C.1.

5.4.2 Updating the Structure

After having updated the model parameters, we sample the latent variables $Y_{P,d}$ using collapsed Gibbs sampling. For this, we marginalise out the $\boldsymbol{v}$ parameters in Equation (5.8) and draw each $y_{P,d}$ from the respective conditional.

Let $\mathcal{D}_{P,d} = \{x_{i,d} \,|\, P \in V(T(\boldsymbol{z}_i))\}_{i=1}^{N}$ denote the d^{th} dimension of each datum in the training set that reached partition P, and let $\boldsymbol{y}_P$ be the set of all dimension assignments at the partition. Further, let $\boldsymbol{y}_{P,-d} = \{y_{P,d'} \,|\, d' \neq d\}_{d=1}^{D}$ denote the exclusion of the d^{th} assignment from $\boldsymbol{y}_P$ and let $\boldsymbol{y}_{-P,d} = \{y_{\mathbf{P}',d} \,|\, P' \neq P\}_{P' \in \mathcal{R}}$ denote the assignments at all partitions but partition P. Then the probability of assigning the d^{th} dimension to the k^{th} child of partition P is,

$$\begin{aligned} p(y_{P,d} = k \,|\, \boldsymbol{y}_{P,-d}, \boldsymbol{y}_{-P,d}, \mathcal{D}_{P,d}, \boldsymbol{z}, \boldsymbol{\theta}, \beta) \\ \propto p(y_{P,d} = k \,|\, \boldsymbol{y}_{P,-d}, \beta)\, p(\mathcal{D}_{P,d} \,|\, y_{P,d} = k, \boldsymbol{y}_{-P,d}, \boldsymbol{z}, \boldsymbol{\theta})\,. \end{aligned} \tag{5.10}$$

The conditional prior in Equation (5.10) follows standard derivations, i.e.

$$p(y_{P,d} = k \,|\, \boldsymbol{y}_{P,-d}, \beta) = \frac{p(\boldsymbol{y}_P \,|\, \beta)}{p(\boldsymbol{y}_{P,-d} \,|\, \beta)}\,, \tag{5.11}$$

which we obtain by first finding the marginal likelihood of the Dirichlet-multinomial distribution. For this let $m_{P,k} = \sum_{d \in \psi(P)} \mathbb{1}_{\{y_{P,d}=k\}}$ and $m_P = \sum_{k=1}^{K} m_{P,k}$. Then we can obtain the marginal likelihood by integrating over the weights $\boldsymbol{v}$, i.e.

$$p(\boldsymbol{y}_P \,|\, \beta) = \int_{\boldsymbol{v}_P} p(\boldsymbol{y}_P \,|\, \boldsymbol{v}_P)\, p(\boldsymbol{v}_P \,|\, \beta)\, \mathrm{d}\boldsymbol{v}_P \tag{5.12}$$

$$= \int_{\boldsymbol{v}_P} \underbrace{\prod_{k=1}^{K} v_{P,k}^{m_{P,k}}}_{=p(\boldsymbol{y}_P \,|\, \boldsymbol{v}_P)} \underbrace{\frac{1}{B(\beta)} \prod_{k=1}^{K} v_{P,k}^{\beta_k - 1}}_{=p(\boldsymbol{v}_P \,|\, \beta)} \mathrm{d}\boldsymbol{v}_P \tag{5.13}$$

$$= \underbrace{\frac{1}{B(\beta)}}_{=\frac{\prod_k \Gamma(\beta_k)}{\Gamma(\sum_k \beta_k)}} \int_{\boldsymbol{v}_P} \prod_{k=1}^{K} v_{P,k}^{m_{P,k}} \prod_{k=1}^{K} v_{P,k}^{\beta_k - 1}\, \mathrm{d}\boldsymbol{v}_P \tag{5.14}$$

$$= \frac{1}{B(\beta)} \int_{\boldsymbol{v}_P} \prod_{k=1}^{K} v_{P,k}^{m_{P,k} + \beta_k - 1}\, \mathrm{d}\boldsymbol{v}_P \tag{5.15}$$

$$= \frac{\Gamma(\sum_k \beta_k)}{\Gamma(m_P + \sum_k \beta_k)} \prod_{k=1}^{K} \frac{\Gamma(m_{P,k} + \beta_k)}{\Gamma(\beta_k)}\,. \tag{5.16}$$

After having obtained the respective marginals, we can use the standard derivation of Gibbs conditionals for collapsed Gibbs sampling in mixture models [124, p. 845], i.e.

$$p(y_{P,d} = k \,|\, \boldsymbol{y}_{P,-d}, \beta) = \frac{m_{P,k,-d} + \beta_k}{m_P - 1 + \sum_k \beta_k}\,, \tag{5.17}$$

where $m_{P,k,-d} = \sum_{d' \in \psi(P) \setminus_d} \mathbb{1}_{\{y_{P,d'}=k\}}$, to update the scope function encoding. Note that the likelihood term in Equation (5.10) is computed by taking a product over the likelihood functions of each cross-product in partition P for dimension d, c.f. Section 3.2.2, as each term is independent.

5.4.3 Performing Predictions

Given a set of S posterior samples, we can now compute predictions for an unseen datum $\boldsymbol{x}^*$ using Monte Carlo integration, i.e.

$$p(\boldsymbol{x}^* \mid \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} \mathcal{S}(\boldsymbol{x}^* \mid \boldsymbol{y}^{(s)}, \boldsymbol{w}^{(s)}, \boldsymbol{\theta}^{(s)}) \,, \tag{5.18}$$

where $\mathcal{S}(\boldsymbol{x}^* \mid \boldsymbol{y}^{(s)}, \boldsymbol{w}^{(s)}, \boldsymbol{\theta}^{(s)})$ denotes the SPN of the s^{th} posterior sample. We can interpret the resulting distribution as an SPN with S children, each one being a sub-SPNs with different parameters and different structure.

The central object, when performing predictions this way is arguably the posterior distribution. Recall that the posterior distribution reflects our posterior beliefs, after having observed data, and is directly derived from our prior assumption. To guarantee consistency of the Bayesian approach[22], i.e. when performing posterior inference the posterior distribution will concentrate on the true model, we require that the true model is in the interior of the hypothesis space and has a positive prior probability. However, often we have situations in which the model complexity should increase with the increase of observed data, e.g. streaming data or topic modelling. Bayesian nonparametric models promise a solution to this problem by using infinite-dimensional priors. We will review extensions of SPNs to nonparametric formulations in the next section.

5.5 Nonparametric Sum-Product Networks

Dirichlet process

In Bayesian nonparametrics, we "expand" the prior distributions to infinite-dimensional spaces, facilitated in the form of a random measure[23]. One such prior is the Dirichlet process (DP), which can be understood as the infinite-dimensional generalisation of a Dirichlet distribution that has Dirichlet distributed finite-dimensional marginals. The DP is often also described as a distribution over distributions. An essential property of the DP, is that draws from a DP are almost surely discrete, i.e. we obtain a discrete distribution when we draw from a DP. We refer to Hjort, Holmes, Müller *et al.* [125] for details on the DP and other nonparametric priors. In this section, we will make use of the

[22] Bayesian consistency under the true model is an implication of the Bernshteĭn-von-Mises theorem. See Vaart [80] for details.

[23] We will not require knowledge about random measures throughout this thesis. In short, a random measure is a random variable on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, which takes values in a measurable space $(M, \mathcal{M})$ for which M is the set of all finitely bounded measures on a metric space $(\mathbb{R}^D, \mathcal{B}(\mathbb{R}^D))$ and $\mathcal{M}$ is the minimal σ-algebra on M. See Hjort, Holmes, Müller *et al.* [125] for details.

DP to i) formulate infinite SPTs and ii) introduce a more efficient non-parametric formulation in the form of an infinite mixture over Bayesian SPNs.

5.5.1 Infinite Sum-Product Trees

In order to define the generative process for infinite Sum-Product Trees (iSPTs) [99], the following simplifying but not very restrictive assumptions are made:

1. all leaf distributions are univariate,
2. all product nodes have exactly two children,
3. sums and products occur in an alternating fashion and,
4. the root node is a sum.

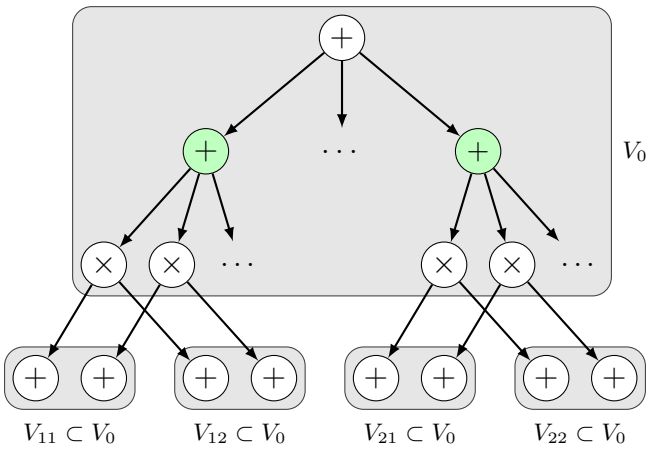


Figure 5.3: Schematic illustration of an iSTP with additional group nodes, indicated in green. Gray boxes indicate the scope of nodes within a box, denoted as $V_0, V_{11}, V_{12}, V_{21}, V_{22}$ respectively. Note that the generative process of iSPTs guarantees that $V_{11} \cup V_{12} = V_0$, $V_{21} \cup V_{22} = V_0$, $V_{11} \cap V_{12} = \emptyset$, $V_{21} \cap V_{22} = \emptyset$, and $V_i \neq \emptyset$ for all $i \in \{0, 11, 12, 21, 22\}$.

Additional to those assumptions, we augment an SPN structure with so-called group nodes. Each group node, which is essentially an additional sum node, groups product nodes that share the same partition of their scope together. Figure 5.3 illustrates the basic structure of iSPTs with additional group nodes, indicated in green. Note that the number of group nodes, i.e. the children of sum S with scope $\psi(\mathsf{S})$, is equal to the Stirling number of the second kind[24], i.e. $\left\{ {|\psi(\mathsf{S})| \atop 2} \right\}$. In order to construct product nodes and their respective partition of the scope, we add $\left\{ {|\psi(\mathsf{S})| \atop 2} \right\}$ many group nodes under each sum nodes, each of which representing a different partition of the current scope, and draw weights to the group nodes from a Dirichlet distribution. Once we obtained the weights, we can sample cluster assignments of the observations at each group node

[24] The Stirling number of the second kind $\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$ counts the number of possible partitions of a set of n objects into k non-empty subsets.

Chinese Restaurant process

and subsequently construct product nodes with the now deterministically defined partition of the scope. To represent group nodes we use the so-called Chinese Restaurant Process (CRP), which represents the marginal probabilities of a DP in the form of a random assignment of the observations to groups (clusters). Let $\alpha > 0$ be the concentration parameter of the DP and H its base distribution, i.e. draws from the DP concentrate on H once $\alpha \to \infty$. Further, let H be defined on Θ and let $\theta_1, \theta_2, \cdots \in \Theta$ be an i.i.d. infinitely exchangeable sequence[25] drawn according to

$$\boldsymbol{x}_i \sim p(\boldsymbol{x}_i \,|\, \theta_i) \qquad \theta_1, \theta_2, \ldots \sim G \qquad G \sim \mathrm{DP}(\alpha, H)\,. \tag{5.19}$$

Since G is almost surely discrete, values θ_i will be repeated. We will, therefore, use $\theta_1^*, \theta_2^*, \ldots$ to denote the unique values among those. By integrating over G, we obtain the marginal distribution in form of consecutive conditional distributions known as the Blackwell-MacQueen's urn scheme [126] or the CRP, i.e.

$$\theta_i \sim \frac{1}{\alpha + i - 1}\left(\alpha H + \sum_{k=1}^{K} n_k \delta_{\theta_k^*}\right), \tag{5.20}$$

where $\delta_{\theta_k^*}$ is a Dirac delta on θ_k^* and n_k is the number of observations that are assigned to cluster k. We see from Equation (5.20) that the probability that θ_i is equal to θ_k^* is proportional to the number of observations assigned to cluster k, i.e. n_k. In other words, the larger n_k is, the more likely it is that n_k will grow with increasing i, i.e. a DP has a rich-get-richer property. Note that in the following, we will directly use the conditional probability of assigning observation i onto cluster k, i.e.

$$p(z_i = k \,|\, z_1, \ldots, z_{i-1}) = \begin{cases} \frac{n_k}{i-1+\alpha} & \text{if } n_k > 0 \\ \frac{\alpha}{i-1+\alpha} & \text{otherwise} \end{cases}, \tag{5.21}$$

as done in the nested CRP [127].

The generative process of iSPTs can be outlined as follows:

1. If the scope of the current node S is multivariate, then:

$$c_{\mathsf{S}} \sim \mathrm{Cat}(c_{\mathsf{S}} \,|\, \boldsymbol{w}_{\mathsf{S}}) \qquad \boldsymbol{w}_{\mathsf{S}} \sim \mathrm{Dir}(\boldsymbol{w}_{\mathsf{S}} \,|\, \alpha_1, \ldots, \alpha_{\left\{\frac{|\psi(\mathsf{S})|}{2}\right\}}) \tag{5.22}$$

$$z_{c_{\mathsf{S}}} \sim \mathrm{CRP}(z_{c_{\mathsf{S}}} \,|\, \beta)\,, \tag{5.23}$$

   for the selected product node $z_{c_{\mathsf{S}}}$: split the scope into partition c_{S} and apply the process recursively for all children.

2. otherwise $d = \psi(\mathsf{S})$ and:

$$x_d \sim p(x_d \,|\, \theta_{c_{\mathsf{S}}}) \qquad \theta_i \sim p(\theta_i \,|\, \gamma) \qquad c_{\mathsf{S}} \sim \mathrm{CRP}(c_{\mathsf{S}} \,|\, \beta)\,. \tag{5.24}$$

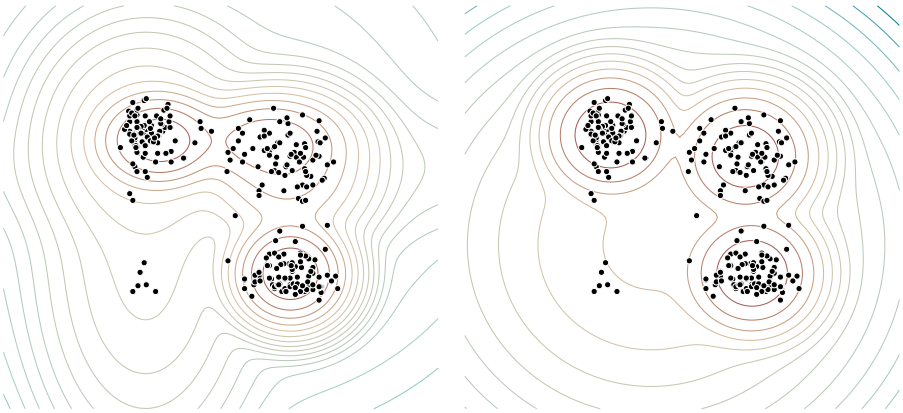
[25] A sequence is exchangeable if the probability of drawing the sequence does not change under arbitrary permutations, i.e. $p(\theta_1, \theta_2, \ldots) = p(\theta_{\sigma(1)}, \theta_{\sigma(2)}, \ldots)$ where σ is an arbitrary permutation.

Experiments

To assess the performance of iSPTs we compare the log posterior predictive probability of iSPTs with Gaussian leaves to infinite Gaussian mixtures models (iGMMs) [128] on the Old Faithful geyser dataset [129], the Chemical Diabetes dataset[26] [130] and the Iris dataset [131]. In each experiment, we initialise iSPTs and iGMMs using a sequential construction and performed posterior inference using the Gibbs sampler proposed by Neal [132]. To account for different choices of the CRPs hyperparameters, we sampled the concentration parameter from independent Gamma priors, as proposed by Escobar and West [133].

Figure 5.4 shows the resulting density function for the iSPT and the iGMM on the Old Faithful dataset. We see that the iSPT is capable of recovering the correlations between input dimensions and models the data distribution (visually) more accurately. Note that, even though both approaches are effectively infinite mixtures over Gaussians with diagonal covariance matrix, the iSPT has additional flexibility through its hierarchical structure.

Table 5.1 shows the average 10-fold cross-validation log posterior predictive probabilities [134] and Mann-Whitney U test values[27] for iGMMs and iSPTs on all three datasets. We estimated the posterior distribution of each model using 1k MCMC samples obtained as described before. We see that iSPTs obtain significantly ($p < 0.01$) better results on all datasets. However, note that posterior inference in iSPTs comes with a higher cost than in iGMMs. Thus, making it infeasible to apply iSPTs on high-dimensional data. In fact, due to the explicit construction of all possible two-partitions, i.e. the group nodes, iSPTs scale poorly with the data dimensionality and are limited to low-dimensional data domains.



(a) iSPT (b) iGMM

Figure 5.4: Density function of an iSPT and an iGMM on the Old Faithful geyser dataset after 1k iterations. Both models use the same hyperparameters and base distribution.

[26] We used the features: glucose area, insulin area and insulin resistance.

[27] The Mann-Whitney U test is a nonparametric significance test for numerical data with a natural order, such as ordinal data, which does not require that the compared populations follow a normal distribution, as assumed by the t-test.

Dataset	iGMM	iSPT	p-value
Old Faithful	-1.737	-1.700	< 0.01
Chemical Diabetes	-3.022	-2.879	< 0.01
Iris	-3.943	-3.744	< 0.01

Table 5.1: Average 10-fold cross-validation log posterior predictive probability and Mann-Whitney U test p-values.

5.5.2 Infinite Mixture of Bayesian Sum-Product Networks

To overcome the limitations of the generative process for iSPTs, we will use a natural extension of our formulation for Bayesian SPNs. In particular, we will use the stick-breaking construction [135] for DPs and define an infinite mixture of Bayesian SPNs. Note that it is also possible to replace each Dirichlet prior in Equation (5.7) with a DP prior. By using a nonparametric prior for at each sum and product node, we can obtain a fully specified generative process for nonparametric tree-shaped region graphs with potentially infinitely many regions and partitions. However, due to the difficulty of efficient posterior inference in nonparametric trees, e.g. [136], [137], we will restrict our approach to a nonparametric mixture over finite SPNs.

stick-breaking construction

The stick-breaking construction defines a random draw from a DP with base measure H as:

$$\begin{aligned} &\theta_k \sim H \qquad\qquad v_k \sim \mathrm{Beta}\left(v_k \,|\, 1,\ \alpha\right) \\ &G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k} \quad \pi_k = v_k \prod_{j=1}^{k-1} (1 - v_j) \qquad \pi_1 = v_1 \,. \end{aligned} \tag{5.25}$$

In order to perform inference when using a stick-breaking construction we need to evaluate conditionals for potentially infinitely many components. One way to approach this, is to introduce slice variables $u_i \sim \mathrm{U}\left(0, \pi\right)$, such that $\sum_{k=1}^{\infty} \mathbb{1}_{\{u_i \leq \pi_k\}} p(\boldsymbol{x}_i \,|\, \theta_k)$ has $\sum_{k=1}^{\infty} \pi_k p(\boldsymbol{x}_i \,|\, \theta_k)$ as its marginal distribution. The resulting inference algorithm, introduced by Walker [138], can be understood to dynamically truncate the infinite mixture and allowing us to sample from a finite number of conditionals. To apply the slice sampling approach for DP mixtures to large-scale data, Ge, Chen, Wan *et al.* [52] suggested an extension of the original slice sampler that allows the computations to be distributed. In the distributed slice sampler, latent assignments $p(z_i = k \,|\, \boldsymbol{x}_i, u_i) = \mathbb{1}_{\{u_i \leq \pi_k\}} f(\boldsymbol{x}_i \,|\, \theta_k)$ can be drawn independently [52]. Subsequently, the truncated infinite collection of weights will be drawn from a finite Dirichlet distribution, i.e.

$$m_k = \begin{cases} \sum_{i=1}^{N} \mathbb{1}_{\{z_i = k\}} & \text{if } \exists i\, \mathbb{1}_{\{z_i = k\}} \\ \alpha / M & \text{otherwise} \end{cases} \tag{5.26}$$

$$\pi \sim \mathrm{Dir}(m_1, \ldots, m_K, \alpha/M)\,, \tag{5.27}$$

where M denotes the total number of empty clusters plus one. Using the generative model in Equation (5.7) as base distribution H allows us

to formulate infinite mixtures of Bayesian SPNs, which can readily be inferred using the distributed slice sampler.

5.6 Experiments

We assessed the performance of Bayesian SPNs and infinite mixtures of Bayesian SPNs on discrete [36] and heterogeneous data [17] as well as on three datasets with missing values. We constructed the computational graph $\mathcal{G}$ using the algorithm described in Section A.1 and used a grid search over the parameters of the graph to select the best performing computational graph. Since the Bayesian framework is protected against overfitting, we combined training and validation sets and followed classical Bayesian model selection [139], i.e. using the Bayesian model evidence, to select the best performing computational graph. The grid search was defined over: the number of nodes per region $I \in [5, 10]$, the number of nodes per atomic region $I \leq J \in [5, 10]$, the number of partitions under a region $M \in [2, 4, 8]$, and the number of consecutive region-partition layers $L \in [1, 2]$. Further, we used 5×10^3 burn-in steps and estimated the predictive distribution using 10^4 samples from the posterior. Note that we used $\alpha = 1.0$ as concentration parameter for all sum nodes and $\beta = 10.0$ as concentration parameter for all product nodes. Details on the selected parameters and the runtime for each dataset are listed in Appendix C.3.

Table 5.4 lists the test log-likelihood scores of state-of-the-art (SOTA) structure learners, i.e. LearnSPN [36], LearnSPN with parameter optimisation (CCCP) [15], ID-SPN [38], random region graphs (RAT-SPN) [10] and the results obtained using Bayesian SPNs (BSPN) and infinite mixtures of Bayesian SPN (BSPNs$^\infty$) on discrete datasets. In addition we list the best-to-date (BTD) results, collected based on the most recent works on structure learning for SPNs [140], PSDDs [11] and CNets [102], [103]. In many cases, we observe an improvement over LearnSPN with additional parameter learning and often obtain results comparable to ID-SPN or sometimes outperform BTD results. Note that ID-SPN uses a more expressive formulation by utilizing Markov networks as leaves and that the BTD results are often obtained by large ensembles of structures. Significant differences to the best SOTA approach under the Mann-Whitney-U-Test [141] with $p < 0.01$ are underlined. An extended results table, including the test result for each pair, is shown in Appendix C.2.

Additionally, we conducted experiments on heterogeneous data, we refer to Vergari, Molina, Peharz *et al.* [17] and Molina, Vergari, Di Mauro *et al.* [40] for details on the datasets. We compared Bayesian SPNs and infinite mixtures of Bayesian SPNs against mixed SPNs (MSPN) [40] and ABDA [17]. Similar to the approach in ABDA, we used mixtures over parametric families at the leaves, see Table 5.2 for a listing of the likelihood and prior constructions used in the experiments. The analytic expressions of the respective posterior distributions are listed in Appendix C.1. A critical difference between Bayesian SPNs and ABDA is that ABDA performs inference on a pre-defined SPN structure, while Bayesian SPNs perform inference over the structure, parameters and the

parametric families jointly.

Datatype	Likelihood	Prior
Continuous	$\mathcal{N}(\mu, \sigma^2)$	$\sigma^2 \sim \Gamma^{-1}(2.0, 3.0)$
		$\mu \sim \mathcal{N}(\tilde{\mu}, \sigma^2)$
Continuous	$\mathrm{Exp}\,(\lambda)$	$\lambda \sim \Gamma(1.0, 1.0)$
Discrete	$\mathrm{Poisson}\,(\lambda)$	$\lambda \sim \Gamma(1.0, 1.0)$
Discrete	$\mathrm{Cat}(\boldsymbol{w})$	$\boldsymbol{w} \sim \mathrm{Dir}(0.1, \ldots, 0.1)$
Discrete	$\mathrm{B}\,(p)$	$p \sim \mathrm{Beta}\,(0.5,\ 0.5)$

Table 5.2: Likelihood and prior distributions used for heterogeneous data experiments.

Note that we further assume that the distribution of each leaf factorises, i.e.

$$\mathsf{L}(\boldsymbol{x}) = \prod_{X_d \in \psi(\mathsf{L})} \sum_{k=1}^{K} w_k \, p_{X_d}(x_d \,|\, \theta_{\mathsf{L},d,k})\,, \tag{5.28}$$

where $k = 1, \ldots, K$ indexes the k^{th} parametric form. We used a symmetric Dirichlet prior $\boldsymbol{w} \sim \mathrm{Dir}(\alpha_1, \ldots, \alpha_K)$ with $\alpha_k = \frac{0.1}{K}$ at each leaf, to enforces that few components are selected.

Table 5.3 lists the test log-likelihood scores of all approaches, indicating that our approaches perform comparable to structure learners tailored to heterogeneous datasets and sometimes outperform MSPNs and ABDA[28]. Surprisingly, we obtain, with a large margin, better test scores for `Autism`, which might indicate that existing approaches overfit the training data, while our formulation naturally penalises complex models.

Dataset	MSPN	ABDA	BSPN	BSPN$^\infty$
Abalone	**9.73**	2.22	3.92	3.99
Adult	−44.07	−5.91	**−4.62**	−4.68
Australian	−36.14	**−16.44**	−21.51	−21.99
Autism	−39.20	−27.93	**−0.47**	−1.16
Breast	−28.01	−25.48	**−25.02**	−25.76
Chess	−13.01	−12.30	**−11.54**	−11.76
Crx	−36.26	**−12.82**	−19.38	−19.62
Dermatology	−27.71	−24.98	**−23.95**	−24.33
Diabetes	−31.22	**−17.48**	−21.21	−21.06
German	−26.05	**−25.83**	−26.76	−26.63
Student	−30.18	**−28.73**	−29.51	−29.9
Wine	**−0.13**	−10.12	−8.62	−8.65

Table 5.3: Average test log-likelihoods on heterogeneous datasets using MSPNs, ABDA, Bayesian SPN (BSPN) and infinite mixtures of SPNs (BSPN$^\infty$). Overall best result is indicated in bold.

[28] We did not apply a significance test as the results of the prior art is not openly available.

We further evaluated LearnSPN, ID-SPN and Bayesian SPNs on three discrete datasets with artificially introduced missing values in the training and validation set. In particular, we incrementally increased the number of observations having 50% of their features missing completely at random [142]. We evaluated LearnSPN and ID-SPN by: i) removing all observations with missing values or ii) using k-Nearest Neighbour (k-NN) imputation [143] (denoted with an asterisk). Note that we selected k-NN imputation as it arguably provides a stronger baseline than simple mean imputation, while being computationally more demanding. All methods have been trained using the full training set, i.e. training and validation set combined, and were evaluated using default parameters to ensure a fair comparison across methods and levels of missing values. In particular, we evaluate their performance in the cases of 20%, 40%, 60% or 80% of all observations having 50% missing values. We used the following default parameters for each approach: (LearnSPN): cluster penalty = 0.6, significance threshold = 10 as described in [36], (ID-SPN): the default settings described in [38], and for (Bayesian SPN): $I = 5$ nodes per region, $J = 10$ nodes per atomic region, $R = 8$ partitions under a region, and a depth of $L = 1$.

Figure 5.5 shows that our approach is consistently more robust against missing values than learnSPN and ID-SPN, which both quickly degenerate in performance with an increasing amount of missing values (even with additional k-NN imputation).

Dataset	LearnSPN	RAT-SPN	CCCP	ID-SPN	BSPN	BSPN$^\infty$	BTD
NLTCS	−6.11	−6.01	−6.03	−6.02	**−6.00**	−6.02	−5.97
MSNBC	−6.11	−6.04	−6.05	−6.04	−6.06	**−6.03**	−6.03
KDD	−2.18	−2.13	−2.13	−2.13	**−2.12**	−2.13	−2.11
Plants	−12.98	−13.44	−12.87	**−12.54**	−12.68	−12.94	−11.84
Audio	−40.50	−39.96	−40.02	−39.79	**−39.77**	−39.79	−39.39
Jester	−53.48	−52.97	−52.88	−52.86	**−52.42**	−52.86	−51.29
Netflix	−57.33	−56.85	−56.78	−56.36	**−56.31**	−56.80	−55.71
Accidents	−30.04	−35.49	−27.70	**−26.98**	−34.10	−33.89	−26.98
Retail	−11.04	−10.91	−10.92	−10.85	−10.83	**−10.83**	−10.72
Pumsb-star	−24.78	−32.53	−24.23	**−22.41**	−31.34	−31.96	−22.41
DNA	−82.52	−97.23	−84.92	**−81.21**	−92.95	−92.84	−81.07
Kosarak	−10.99	−10.89	−10.88	**−10.60**	−10.74	−10.77	−10.52
MSWeb	−10.25	−10.12	−9.97	**−9.73**	−9.88	−9.89	−9.62
Book	−35.89	−34.68	−35.01	−34.14	**−34.13**	−34.34	−34.14
EachMovie	−52.49	−53.63	−52.56	−51.51	−51.66	**−50.94**	−50.34
WebKB	−158.20	−157.53	−157.49	**−151.84**	−156.02	−157.33	−149.20
Reuters-52	−85.07	−87.37	−84.63	**−83.35**	−84.31	−84.44	−81.87
20 Newsgroup	−155.93	−152.06	−153.21	**−151.47**	−151.99	−151.95	−151.02
BBC	−250.69	−252.14	**−248.60**	−248.93	−249.70	−254.69	−229.21
AD	−19.73	−48.47	−27.20	**−19.05**	−63.80	−63.80	−14.00

Table 5.4: Average test log-likelihoods on discrete datasets using SOTA, Bayesian SPNs (BSPN) and infinite mixtures of SPNs (BSPN$^\infty$). Significant differences are underlined. Overall best result is in bold. In addition we list the best-to-date (BTD) results obtained using SPNs, PSDDs or CNets.

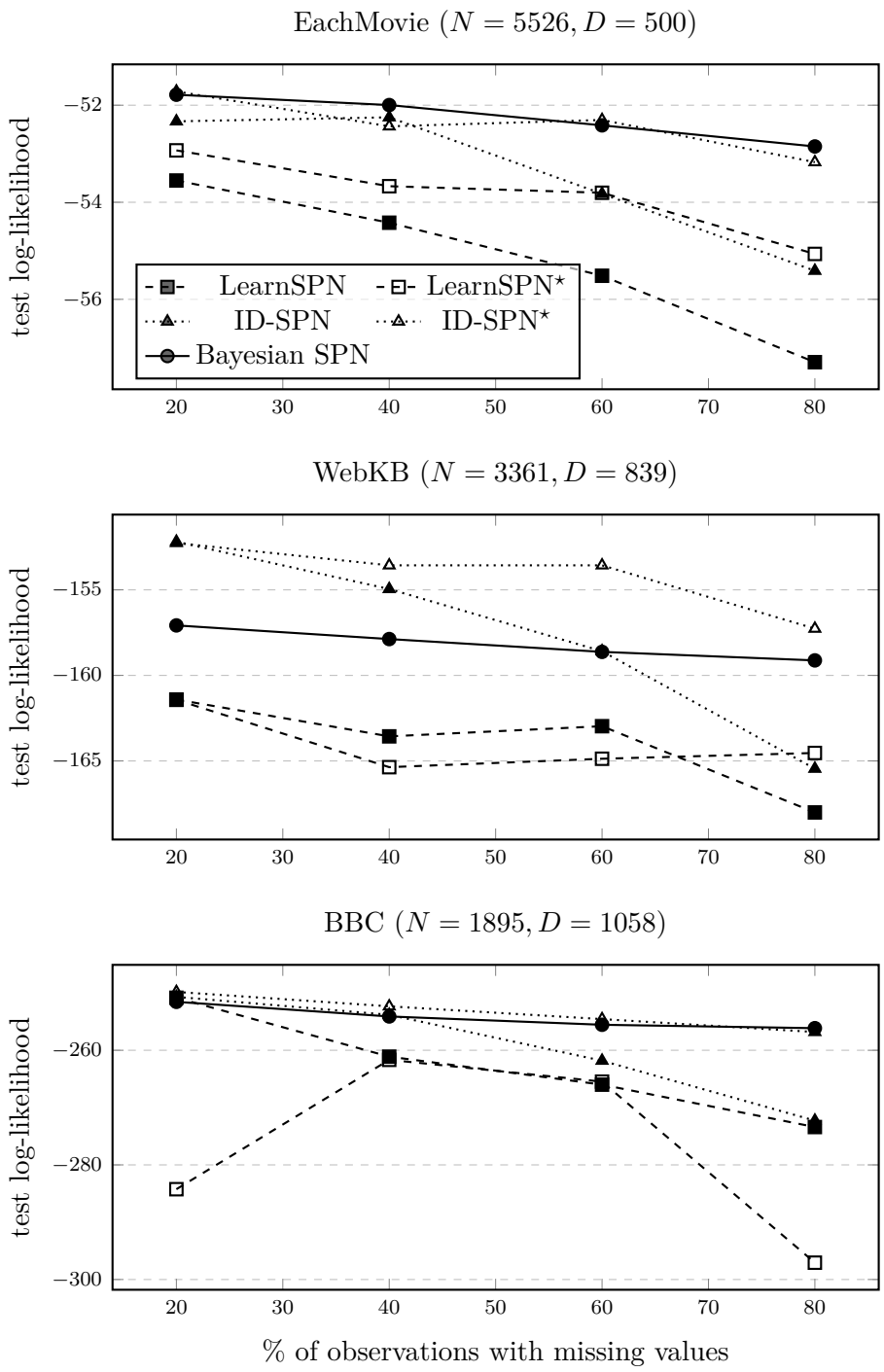


Figure 5.5: Performance under missing values for discrete datasets with increasing dimensionality (D). Results for LearnSPN are shown in dashed lines, results for ID-SPN in dotted lines and our approach is indicated using solid lines. Filled symbols indicate that the method does not use any additional imputation method.

6 Sum-Product Networks over Gaussian Processes

6.1 Motivation

So far, we have only considered modelling tasks in which the number of RVs is finite. However, many practical applications deal with situations in which the number of RVs is potentially infinite, e.g. daily forecasting of the number of expected COVID-19 infections. In such cases, stochastic process models that can capture uncertainties and model complex relationships in time series data are indispensable. A Gaussian Process (GP) is an example of such a stochastic process model, allowing for probabilistic non-linear regression analysis. One remarkable property of GPs is that they allow exact posterior inference, i.e. we can obtain the posterior mean and variance of a GP in analytical form. However, computing the posterior distribution comes with high computational and memory costs. In fact, the computation scales cubic in the number of observations N, i.e. $\mathcal{O}(N^3)$, and has memory requirement of $\mathcal{O}(N^2 + N\,D)$, where D is the dimensionality of the input [50]. Therefore, GPs are often limited to small data domains or require approximation schemes if used in large scale applications. The most common strategies to reduce the computational and memory costs are based on either variational approximations to the GP posterior or based on local GP experts [144].

The first approach is undoubtedly the more dominant one as it allows for straightforward implementation using differential programming [145]. In this case, the posterior of a GP is represented by Q *inducing points*, which are treated as variational parameters, and learned by minimising the Kullback-Leibler divergence[29] of the approximate posterior from the full posterior. Variational approximations reduce the computational burden to $\mathcal{O}(N\,Q^2)$ [146]. As shown by Burt, Rasmussen and Wilk [147], asymptotically the number of inducing points has to increase with a polylogarithmic rate, i.e. $\mathcal{O}(\log(N)^D)$, in order to guarantee convergence with high probability. In the non-asymptotic regime, however, variational approximation may struggle in producing a good sparse approximation.

KL divergence

Approximations based on local experts, on the other hand, use a *divide-and-conquer* strategy and partition the input space (or the dataset) into subsets, each modelled with an individual GP expert. For K experts, each with $M << N$ observations, the computational and memory

[29] The Kullback-Leibler (KL) divergence of distribution $q(x)$ from distribution $p(x)$, often denoted as $D_{\mathrm{KL}}(p(x), q(x))$, measures the information lost when using $q(x)$ to approximate $p(x)$.

costs are typically reduced to $\mathcal{O}(K\,M^3)$ and $\mathcal{O}(K(M^2 + M\,D))$, respectively. Prominent examples include the Naive-Local-Experts model (NLE) [148], [149], which naively models each partition of the input space with an independent GP, Products-of-Experts (PoE) [150], [151], which aggregate predictive distributions from experts using a product operation, and the Mixture-of-Experts (MoE) [151], [152], which dynamically distribute observations to experts.

Naive-Local-Experts
Product-of-Experts
Mixture-of-Experts

All these local-expert approaches have different advantages and disadvantages. The NLE model allows exact posterior inference, which reduces to independent GP inference at each expert, but introduces hard discontinuities in the input space. In case the partition is not well-supported by the data, NLE models can, therefore, result in high generalisation errors [144]. PoE approaches result in a natural strategy to distribute the data, but have been shown to result in sub-optimal rates of the posterior contraction [153]. Further, the combination of local experts using a product aggregation is known to be Kolmogoroff inconsistent[30] [154]. Even in the case of the Bayesian committee machine (BCM) [151], where the PoE approach is justified as an approximation to Bayesian posterior inference, the introduced approximation error is hard to analyse. Finally, MoE models specify a sound stochastic process model, but do not permit tractable posterior inference and rely on approximate inference techniques for posterior inference.

Bayesian committee machine

In this chapter, we will introduce a natural combination of SPNs and GPs, called Deep Structured Mixtures of GPs (DSMGPs), as an attractive alternative to previous local-expert approaches. DSMGPs can be understood as an extension of SPNs to the stochastic process case by equipping SPNs with Gaussian measures, which correspond to GPs [155], as leaves. Equivalently, we can interpret DSMGPs as a hierarchical structured mixture over a large number of NLEs. In particular, the posterior of DSMGPs can be naturally understood as Bayesian model averaging over an *exponentially large mixture* of NLEs, i.e. combinatorial in the states of the latent variables of the SPN [16], [19]. The crucial key advantage of DSMGPs is that posterior inference can be performed *exact* and *efficiently*, i.e. DSMPGs inherit tractable inference from SPNs and exact computations from GPs.

6.2 Preliminaries

Before introducing DSMGPs, we will first review the relevant concepts related to stochastic processes and provide a brief introduction to Gaussian process regression.

A common approach to define a stochastic process is via its marginal distribution. For this, consider a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and a measurable space $(E, \mathcal{E})$. A stochastic process is a family of RVs $\{F_t\}_{t\in I}$, with $F_t \colon (\Omega, \mathcal{A}) \mapsto (E, \mathcal{E})$ for all $t \in I$, indexed by an *arbitrary index set* I. In many applications the index set I is interpreted as time and x_t represents the value observed at time t. However, in the context of this thesis,

stochastic process

[30] We refer to approaches as Kolmogoroff inconsistent if there does not exist a probability measure and, therefore, the stochastic process is not well-defined. See Daniell-Kolmogoroff theorem in Section 6.2.

we will assume I to generally not represent time.

In case I is finite, we can see that the stochastic process can be understood as a random vector. In this case, we can readily define the stochastic process as a map

$$\boldsymbol{F}\colon \Omega \times I \mapsto E\,, \qquad (\omega, t) \mapsto F_t(\omega)\,, \tag{6.1}$$

and see that the resulting probability measure is in fact a product measure on a product measurable space. However, often cases, I is uncountable, e.g. $I = \mathbb{R}$, and it is sometimes not obvious how to even construct such a process. Fortunately, as pointed out by Kolomgoroff, a stochastic process with infinitely many RVs is an idealistic construction and it is sufficient to show that such a process exists.

Daniell-Kolmogoroff theorem

Theorem 6.1 (Daniell-Kolmogoroff)**.** *Let $i \in I$ be an uncountable index set, and let $(\Omega_i, \mathcal{B}(\Omega_i))$ be some Borel measurable spaces. Further, let $\mathcal{E}(I)$ be the set of all non-empty, finite subsets of I. If there exist a consistent family*[31] *of probability measures $\{\mathbb{P}_j\}_{j\in\mathcal{E}(I)}$, then there exists an unique probability measure $\mathbb{P}$ on the product measurable space defined as*

$$(\Omega, \mathcal{B}(\Omega)) \coloneqq \left(\bigtimes_{i\in I} \Omega_i, \bigotimes_{i\in I} \mathcal{B}(\Omega_i)\right)\,, \tag{6.2}$$

such that every probability measure $\mathbb{P}_j$ is a push-forward measure for the projection $\pi_j\colon \Omega \mapsto \Omega_j$, i.e. $\mathbb{P}_j = \mathbb{P} \circ {\pi_j}^{-1}$.

The Daniell-Kolmogoroff theorem guarantees that a consistent collection of finite distributions defines a stochastic process and that there exist a unique probability measure on an uncountable product measurable space. We refer to the original work by Kolmogoroff [54] for details.

6.2.1 Gaussian Process Regression

Gaussian process

A Gaussian Process (GP) is defined as a collection of RVs $\boldsymbol{F}$[32] indexed by an arbitrary (uncountable) input space $\mathcal{X}$, where any finite subset of $\boldsymbol{F}$ has a joint Gaussian distribution [50]. Further, by the Daniell-Kolmogoroff theorem we also require that any two overlapping finite marginal distributions are consistent, i.e. if we have $(x_1, x_2) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ then we also have $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{1,1})$ with $\Sigma_{1,1}$ being a sub-matrix of Σ. Note that GPs can naturally be interpreted as distributions over functions $f\colon \mathcal{X} \mapsto \mathbb{R}$. Therefore, GPs are a natural choice as prior distribution in probabilistic regression analysis.

mean-function

A GP is uniquely specified by: i) a *mean-function* $\mu\colon \mathcal{X} \mapsto \mathbb{R}$, which is often assumed to be $\mu(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$, and denoted as a *zero* mean-function or a centred process in this case; and ii) a *kernel-function* $\kappa\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Given a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ and $\boldsymbol{y} = \{y_i\}_{i=1}^N$. Note that we intensionally overload the symbol $\boldsymbol{X}$ to indicate the set of observed covariates, while we use $\boldsymbol{y}$

kernel-function

[31] A family of probability measures on the space $(\Omega_J, \mathcal{A}_J)$ with $J \subset I$ is called projective or consistent if $\mathbb{P}_L = \mathbb{P}_J \circ (\pi_L^J)^{-1}$ for all $L \subset J \subset I$ where $\pi_L^J\colon \Omega_J \mapsto \Omega_L$ is the projection of the components of the index set J onto L.

[32] We use the notation $\boldsymbol{F}$ to indicate the connection to random functions.

to indicate the set of observed response values or outputs. Further, let $\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}}$ denote the $N \times N$ kernel matrix defined by $[\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}}]_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and let $\boldsymbol{m}_{\boldsymbol{X}}$ be the respective mean values, i.e., $[\boldsymbol{m}_{\boldsymbol{X}}]_i = \mu(\boldsymbol{x}_i)$.

In the context of this work, we will assume the observed outputs $y_i \in \mathbb{R}$ at a given input location $\boldsymbol{x}_i \in \mathcal{X}$ to be noisy. Therefore, we obtain a generative process that can be outlined as follows

$$\boldsymbol{y} \,|\, \boldsymbol{X}, \boldsymbol{f}, \sim \mathcal{N}(\boldsymbol{f}, \mathrm{I}\,\sigma^2_{\mathrm{ns}}) \qquad\qquad \boldsymbol{f} \sim \mathrm{GP}(\boldsymbol{m}_{\boldsymbol{X}}, \boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}})\,, \tag{6.3}$$

where I denotes an identity matrix and σ^2_{ns} is the noise variance – assuming a Gaussian likelihood. Consequently, the posterior predictive distribution for an unseen location $\boldsymbol{x}^*$ has a posterior mean of

$$\mu_{\mathcal{D}}(\boldsymbol{x}^*) = \boldsymbol{k}_{\boldsymbol{x}^*,\boldsymbol{X}} \left(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \mathrm{I}\,\sigma^2_{\mathrm{ns}}\right)^{-1} \boldsymbol{y}\,, \tag{6.4}$$

and a posterior variance given as

$$\sigma^2_{\mathcal{D}}(\boldsymbol{x}^*) = \boldsymbol{k}_{\boldsymbol{x}^*,\boldsymbol{x}^*} - \boldsymbol{k}_{\boldsymbol{x}^*,\boldsymbol{X}} \Bigg(\underbrace{\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \mathrm{I}\,\sigma^2_{\mathrm{ns}}}_{=\boldsymbol{C}_{\boldsymbol{X},\boldsymbol{X}}} \Bigg)^{-1} \boldsymbol{k}_{\boldsymbol{X},\boldsymbol{x}^*}\,. \tag{6.5}$$

However, computing Equation (6.5) requires the inversion of the $N \times N$ matrix $\boldsymbol{C}_{\boldsymbol{X},\boldsymbol{X}}$, which scales cubic in N when solved via the Cholesky decomposition [156].

Note that there is an intimidate relationship between GPs, whose function draws are almost surely from a particular function space, and Gaussian measures defined on the same function space. In fact, this relationship is one-to-one for the space of continuously differentiable functions on any real interval, and for L_2-spaces defined on arbitrary measurable spaces [155]. We will make use of this equivalence to describe DSMGPs as a hierarchical mixture, realised as an SPN, over Gaussian measures.

Gaussian measure

6.3 Deep Structured Mixture of Gaussian Processes

Intuitively, a Deep Structured Mixture of GPs (DSMGPs) can be thought of as an "SPN over GPs". Formally, this is most naturally defined via the correspondence of Gaussian measures on a function space of interest and GPs, which almost surely realise in this function space [155].

Deep Structured Mixture of Gaussian Processes

Definition 6.1 (Deep Structured Mixture of Gaussian Processes)**.** *Given a measurable input space $(\mathcal{X}, \Sigma)$, let $(\mathcal{F}, \Sigma_{\mathcal{F}})$ be a measurable function space of real-value functions defined on the (uncountable) set $\mathcal{X}$, i.e. $\mathcal{F} \subset \mathbb{R}^{\mathcal{X}}$, and equipped with a suitable sigma algebra $\Sigma_{\mathcal{F}}$. Then a Deep Structured Mixture of GPs (DSMGP) is defined as an SPN $(\mathcal{G}, \psi, \boldsymbol{w}, \theta)$, where $\mathcal{G}$ is a computational graph (as in Section 3.2.1), ψ is a scope-function $\psi\colon \mathsf{N} \mapsto \Sigma$, $\boldsymbol{w}$ is a set of sum weights, and θ is a set of GP parameters. When N is the root of $\mathcal{G}$, then $\psi(\mathsf{N}) = \mathcal{X}$; additionally, ψ satisfies the conditions 2-4) in Definition 3.10.*

Furthermore, a DSMGP is recursively defined as follows:

1. *A leaf node L computes a Gaussian measure, corresponding to the GP on $\psi(\mathsf{L})$, parametrised by θ_{L}.*

2. *A product node* P *computes a product measure of its children.*

3. *A sum node* S *computes a convex combination (determined by its sum-weights) of the measures computed by its children.*

Definition 6.1 is mathematically elegant as it directly extends the probability measure of SPNs over finitely many RVs, c.f. Section 3.1.2, to infinitely many RVs using Gaussian measures and, by the Daniell-Kolmogoroff theorem, establishes SPNs as a stochastic process model. On the other hand, this definition might obscure how to work with DSMGPs in practice. Recall that any marginal distribution of a Gaussian process is joint Gaussian distributed. In case of an NLE model, the kernel matrix of the posterior distribution is naturally a block-diagonal matrix. Similarly, each component of a DSMGP yields a Gaussian distribution with block-diagonal covariance-structure. Consequently, the marginal distribution of a DSMGP yields a finite – albeit large – mixture of Gaussians with block-diagonal covariance-structure determined by the scope-function.

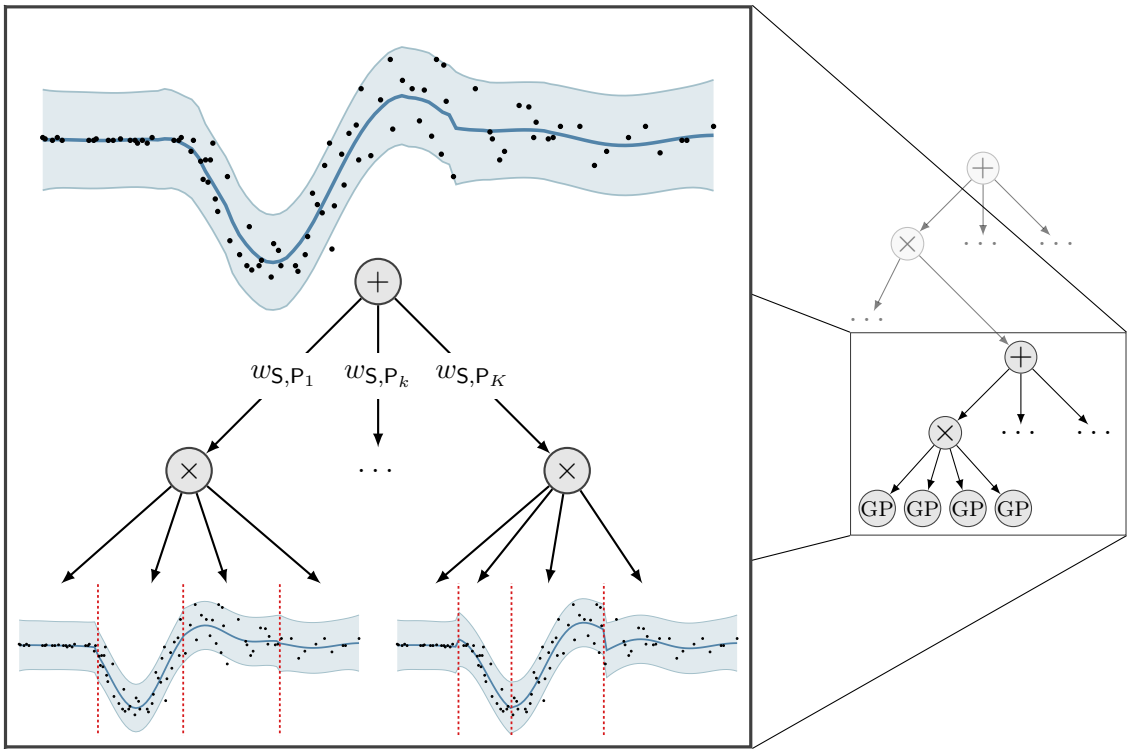


Figure 6.1: Illustration of a deep structured mixture of Gaussian processes. Vertical lines (in red) illustrate the independence assumptions in the input domain (NLE model hypotheses) of each product node. Children of product nodes are either Gaussian process experts or sum nodes.

The structure $(\mathcal{G}, \psi)$ of a DSMGP is either pre-defined or learned using posterior inference [157]. For simplicity, we assume that $\mathcal{G}$ is tree-shaped, i.e. each node has at most one parent, and pre-specify ψ by fixing a random partition of the input space at each product node. When using DSMGPs as a prior over functions, we assume all sum node weights to be uniform, i.e. $w_{\mathsf{S},\mathsf{N}} = 1/K_{\mathsf{S}}$ where K_{S} is the number of children under S. Intuitively, each sum node represents a prior over partitions of the input domain, where split-positions of each partition mark statistical

independence in the input domain. In a DSMGP, split-positions are selected hierarchically, following the same hierarchy as sum nodes in the computational graph. This mechanism is illustrated in Figure 6.1.

Note that DSMGPs are particularly well suited if some areas of the input domain can be expected to be approximately independent. Performing posterior inference in DSMGPs, which can be done *exactly*, updates our belief about the respective split-positions. Thus, in DSMGPs, we automatically infer, from a rich set of choices, those independence assumptions that are well supported by the data. Intuitively, we can understand DSMGPs to implicitly perform exact Bayesian model averaging over a rich set of NLE models.

6.3.1 Exact Posterior Inference

Posterior inference in DSMGPs combines exact posterior inference in GP experts, defined over a subspace of $\mathcal{X}$, with tractable computations in SPNs. This is a crucial advantage over PoE approaches, which do not define a sound probabilistic model, and over MoE approaches, which are often inherently intractable.

Theorem 6.2. *Let $\mathcal{S} = (\mathcal{G}, \psi, \boldsymbol{w}, \theta)$ be a DSMGP on the measurable space $(\mathcal{X}, \Sigma)$, with $\mathcal{X}$ being an input space and Σ a σ-algebra over $\mathcal{X}$. Then, computing the unnormalised posterior distribution of $\mathcal{S}$ simplifies to tractable posterior inference at the leaves.*

Proof. Let $\mathcal{D}$ denote a training set. Then, under the usual i.i.d. assumption, the unnormalised posterior is given as

$$p(\boldsymbol{f} \mid \mathcal{D}) \propto \prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} \underbrace{p(y_i \mid f_i)}_{\text{likelihood}} \underbrace{p(f_i \mid \boldsymbol{x}_i)}_{\text{prior}} . \tag{6.6}$$

If the DSMGP is a leaf L, i.e. it is a Gaussian measure induced by the GP at L, then the computation of the posterior follows the standard computation in GPs [50, Eq. 2.7] and can be performed exactly.

In case the DSMGP is a sum node S, the likelihood terms can be "pulled" over the summation at the sum node, i.e.

$$\begin{aligned} p_{\mathsf{S}}(\boldsymbol{f} \mid \mathcal{D}) &\propto \prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} p(y_i \mid f_i) \sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}}\, p_{\mathsf{N}}(f_i \mid \boldsymbol{x}_i) \\ &= \sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} \prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} p(y_i \mid f_i)\, p_{\mathsf{N}}(f_i \mid \boldsymbol{x}_i) , \end{aligned} \tag{6.7}$$

and posterior inference simplifies to inference at the children of S.

Finally, in case the DSMGP is a product node P, we can swap the product over observations with the product over children and "pull" the likelihood terms down to the respective children, i.e.

$$\begin{aligned} p_{\mathsf{P}}(\boldsymbol{f} \mid \mathcal{D}) &\propto \prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} p(y_i \mid f_i) \prod_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} p_{\mathsf{N}}(f_i \mid \boldsymbol{x}_i) \\ &= \prod_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} \Bigg(\underbrace{\prod_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{(\mathsf{N})}}}_{\bigcup_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} \mathcal{D}_{(\mathsf{N})} = \mathcal{D}} p(y_i \mid f_i)\, p_{\mathsf{N}}(f_i \mid \boldsymbol{x}_i) \Bigg) , \end{aligned} \tag{6.8}$$

where $\mathcal{D}_{(\mathsf{N})}$ denotes the subset of observations that are within the domain of node N and $\cap_{\mathsf{N}\in\mathrm{ch}(\mathsf{P})}\mathcal{D}_{(\mathsf{N})} = \emptyset$. Therefore, posterior inference simplifies to inference at the children of the product node P using subsets of $\mathcal{D}$.

Inductively repeating this argument for all internal nodes, we see that we obtain the unnormalised posterior by multiplying each leaf with its local likelihood. Therefore, the unnormalised posterior of a DSMGP is obtained by performing inference on the leaves, which can be done exactly [50, Eq. 2.7]. □

We can obtain the *normalised* posterior, i.e. $p(\boldsymbol{f} \mid \mathcal{D}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{f})\, p(\boldsymbol{f} \mid \boldsymbol{X})}{p(\boldsymbol{y} \mid \boldsymbol{X})}$, by re-normalising the unnormalised posterior of the DSMGP using a bottom-up propagation of the marginal likelihood of each expert. An efficient approach to renormalise any SPN without changing its distribution has been introduced by Peharz, Tschiatschek, Pernkopf *et al.* [71]. An adaptation of [71, Alg. 1] for DSMGPs is illustrated in Appendix D.2.2.

6.3.2 Predictions

The predictive posterior distribution of a DSMGP for an unseen datum $\boldsymbol{x}^*$ is naturally a mixture distribution and, therefore, can be multimodal. For practical reasons, it is often useful to project the posterior of a DSMGP to the closest GP, i.e. the GP with minimal KL divergence from the DSMGP. This can be done by computing the first and second moments of the resulting mixture distribution, see Rasmussen and Williams [50, Eq. A.24]. Let L be the set of all GP leaves in a DSMGP and let $\tau_j \colon \mathcal{X} \to \{\mathsf{L} \in V(\mathcal{T}_j)\}$ be a bijective map from the input domain to the set of leaves in the j^{th} induced tree $\mathcal{T}_j$, similar to a gating function. Then we can write the mean (first moment) as

$$\mu_{\mathcal{D}}(\boldsymbol{x}^*) = \sum_{j=1}^{K} \prod_{(\mathsf{S},\mathsf{N})\in E(\mathcal{T}_j)} w_{\mathsf{S},\mathsf{N}}\, \mu_{\tau_j}(\boldsymbol{x}^*)\,, \tag{6.9}$$

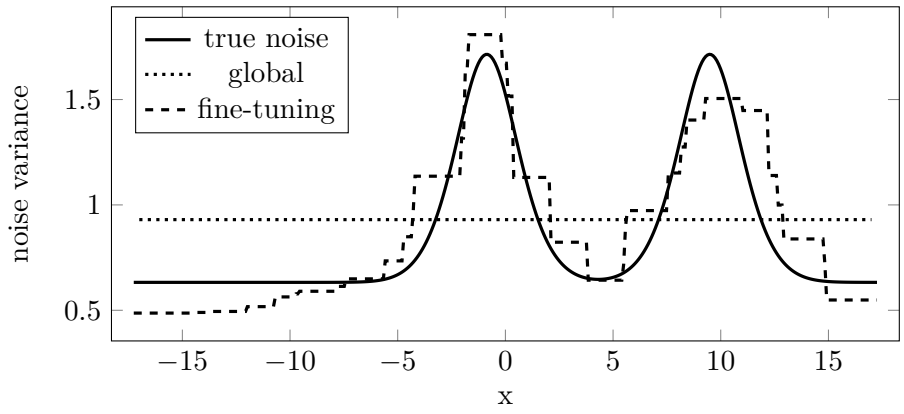
and the variance (second moment) as

$$\sigma^2_{\mathcal{D}}(\boldsymbol{x}^*) = \sum_{j=1}^{K} \prod_{(\mathsf{S},\mathsf{N})\in E(\mathcal{T}_j)} w_{\mathsf{S},\mathsf{N}} \left(\mu^2_{\tau_j}(\boldsymbol{x}^*) + \sigma^2_{\tau_j}(\boldsymbol{x}^*) - \mu^2_{\mathcal{D}}(\boldsymbol{x}^*)\right)\,, \tag{6.10}$$

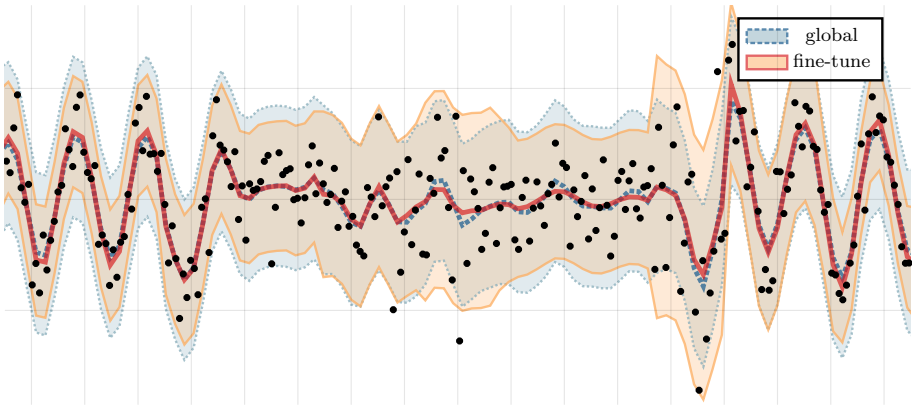
where we use $\mu_{\tau_j}(\boldsymbol{x}^*)$ and $\sigma^2_{\tau_j}(\boldsymbol{x}^*)$ as short-hand notation for the mean and variance of the predictive distribution of the GP allocated at leaf $\tau_j(\boldsymbol{x}^*)$. Both moments can be computed efficiently in DSMGPs using a two step procedure, i.e. first we select the respective leaves using a downward path, and then use an upward path to compute the respective moments.

6.3.3 Hyperparameter Optimisation

We can optimise the hyperparameters, i.e. noise variance and the kernel parameters, of a DSMGP by maximising the *log marginal likelihood* given



(a) Noise parameter after *global* hyperparameter optimisation or *fine-tuning*.



(b) Posterior mean and variance after *global* hyperparameter optimisation or *fine-tuning*.

Figure 6.2: Noise parameter (a) and posterior mean and variance (b) of an DSMGP after global hyperparameter optimisation and fine-tuning on a synthetic dataset generated from a stochastic process with heteroscedastic noise following the "true noise" function illustrated in (a).

the data $\mathcal{D}$. Let us assume a *zero* mean-function, then the log marginal likelihood of a GP at leaf L is computed for the observations that fall into the subspace $\mathcal{X}_\mathsf{L}$ of the input domain. Let $\mathcal{D}_{(\mathsf{L})} = \{(\boldsymbol{x}_i, y_i) \in \mathcal{D} \,|\, x_i \in \mathcal{X}_\mathsf{L}\}$ denote the respective N_L observations and let $\boldsymbol{X}_{(\mathsf{L})}$ and $\boldsymbol{y}_{(\mathsf{L})}$ be the inputs and the observed outputs contained in $\mathcal{D}_{(\mathsf{L})}$.

Then the log marginal likelihood is given as

$$\log p(\boldsymbol{y}_{(\mathsf{L})} \,|\, \boldsymbol{X}_{(\mathsf{L})}) = -\frac{1}{2}\left((\boldsymbol{y}_{(\mathsf{L})}^T \boldsymbol{C}^{-1} \boldsymbol{y}_{(\mathsf{L})}) + \log|\boldsymbol{C}| + N_\mathsf{L} \log 2\pi\right), \quad (6.11)$$

where $\log|\boldsymbol{C}|$ denotes the log determinant of $\boldsymbol{C}$. Consequently, because the DSMGP is a mixture of Gaussian measures, the log marginal likelihood of a DSMGP is written as

$$\log p(\boldsymbol{y} \,|\, \boldsymbol{X}) = \mathrm{L} \sum_{j=1}^{K} \mathrm{E}\left(\log p(\mathcal{T}_j) + \sum_{\mathsf{L} \in V(\mathcal{T}_j)} \underbrace{\log p(\boldsymbol{y}_{(\mathsf{L})} \,|\, \boldsymbol{X}_{(\mathsf{L})})}_{=\text{Equation (6.11)}}\right), \quad (6.12)$$

where $p(\mathcal{T}_j) = \prod_{(\mathsf{S},\mathsf{N}) \in E(\mathcal{T}_j)} w_{\mathsf{S},\mathsf{N}}$ is the probability of the j^{th} induced

tree and $\mathrm{L} \overset{K}{\underset{j=1}{\Sigma}} \mathrm{E}$ denotes the log-sum-exp operation. Note that Equation (6.12) can be computed efficiently using a single upward evaluation through the model. To optimise the hyperparameters we perform gradient-based optimisation according to the partial derivatives of Equation (6.12) w.r.t. the parameters, c.f. Section 3.3. Let $p(\mathsf{L})$ denote the probability of leaf L, i.e. in case of tree-shaped DSMGPs $p(\mathsf{L})$ is the product of the weights on the unique path to L, then the partial derivative for the parameters θ, which are shared across all leaves, can be written as

$$\frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{X})}{\partial \theta} = \sum_{\mathsf{L} \in \mathcal{S}} \frac{1}{p(\boldsymbol{y} \mid \boldsymbol{X})} \frac{\partial p(\mathsf{L})\, p(\boldsymbol{y}_{(\mathsf{L})} \mid \boldsymbol{X}_{(\mathsf{L})})}{\partial \theta} \,. \tag{6.13}$$

After applying the log transformation, i.e. $\frac{\partial f(x)}{\partial x} = f(x) \frac{\partial \log f(x)}{\partial x}$, we obtain the following partial derivatives,

$$\frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{X})}{\partial \theta} = \sum_{\mathsf{L} \in \mathcal{S}} \underbrace{\frac{p(\mathsf{L})\, p(\boldsymbol{y}_{(\mathsf{L})} \mid \boldsymbol{X}_{(\mathsf{L})})}{p(\boldsymbol{y} \mid \boldsymbol{X})}}_{= \frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{X})}{\partial \mathsf{L}}} \frac{\partial \log p(\boldsymbol{y}_{(\mathsf{L})} \mid \boldsymbol{X}_{(\mathsf{L})})}{\partial \theta} \,, \tag{6.14}$$

which admit a numerically stable optimisation of the hyperparameters.

In the case of non-stationary data, we can optionally fine-tune the hyperparameters of each expert after the optimisation. For this purpose, let $\#\mathbf{L}$ denote the cardinality of $\mathbf{L}$ and let $\boldsymbol{S} \in \mathbb{R}^{\#\mathbf{L} \times \#\mathbf{L}}$ be a similarity matrix. Further, let $\boldsymbol{S}$ contain similarity values, i.e. $0 \leq (s_{\mathsf{L},\mathsf{L}'}) \leq 1$ and $s_{\mathsf{L},\mathsf{L}'} = 1$ if $\mathsf{L} = \mathsf{L}'$, between all pairs of leaves $(\mathsf{L}, \mathsf{L}')$ with $\mathsf{L}, \mathsf{L}' \in \mathbf{L}$.

A natural choice for S is a matrix of normalised overlap values, i.e.

$$s_{\mathsf{L},\mathsf{L}'} = \frac{1}{\#\mathcal{D}_{(\mathsf{L})}} \sum_{\boldsymbol{x}_i \in \mathcal{D}_{(\mathsf{L})}} \mathbb{1}_{\{\boldsymbol{x}_i \in \mathcal{D}_{(\mathsf{L}')}\}} \,, \tag{6.15}$$

where $\#\mathcal{D}_{(\mathsf{L})}$ is the cardinality of $\mathcal{D}_{(\mathsf{L})}$. Given such a similarity matrix and provided that each leaf has it's own parameters, denoted as θ_{L}, we can compute the gradients for θ_{L} as

$$\frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{X})}{\partial \theta_{\mathsf{L}}} = \sum_{\mathsf{L}' \in \mathcal{S}} s_{\mathsf{L},\mathsf{L}'} \frac{\partial \log p(\boldsymbol{y} \mid \boldsymbol{X})}{\partial \mathsf{L}} \frac{\partial \log p(\boldsymbol{y}_{(\mathsf{L}')} \mid \boldsymbol{X}_{(\mathsf{L}')}, \theta_{\mathsf{L}'} = \theta_{\mathsf{L}})}{\partial \theta_{\mathsf{L}'}} \,, \tag{6.16}$$

where $p(\boldsymbol{y}_{(\mathsf{L}')} \mid \boldsymbol{X}_{(\mathsf{L}')}, \theta_{\mathsf{L}'} = \theta_{\mathsf{L}})$ denotes the probability distribution conditioned on $\theta_{\mathsf{L}'}$ taking the value θ_{L}. Therefore, $\boldsymbol{S}$ constraints hyperparameters of "similar" leaves to take similar values. Note that Equation (6.16) reduces to Equation (6.14) if $\boldsymbol{S}$ is a matrix of ones and reduces to independent hyperparameter optimisation if $\boldsymbol{S}$ is the identity matrix.

Figure 6.2 illustrates the effects of fine-tuning on a synthetic dataset with heteroscedastic noise [158]. In contrast to global hyperparameter optimisation (Equation (6.14)), fine-tuning allows DSMGPS to capture heteroscedasticity by obtaining an individual noise parameter for each leaf. Therefore, DSMGPs with fine-tuned hyperparameters are more flexible than traditional GPs.

6.3.4 Shared Cholesky Decomposition

Recall that the posterior variance of a GP is calculated according to Equation (6.5), requiring us to invert the matrix $\boldsymbol{C}$. The standard approach to compute the inverse of a Hermitian positive-definite matrix is to first decompose the matrix into a product of an upper triangular matrix $\boldsymbol{U}$ and its conjugate transpose $\boldsymbol{U}^*$, i.e. $\boldsymbol{A} = \boldsymbol{U}^* \boldsymbol{U}$[33]. In case of a kernel matrix (real-valued symmetric positive-definite), the decomposition has a unique solution and the decomposition can be written as $\boldsymbol{K} = \boldsymbol{U}^T \boldsymbol{U}$. This decomposition is called the Cholesky decomposition and can be written as

$$\boldsymbol{K} = \begin{bmatrix} u_{1,1} & & \\ u_{2,1} & u_{2,2} & \\ u_{3,1} & u_{3,2} & u_{3,3} \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ & u_{2,2} & u_{2,3} \\ & & u_{3,3} \end{bmatrix} \tag{6.17}$$

$$= \begin{bmatrix} u_{1,1}^2 & u_{1,1}u_{1,2} & u_{1,1}u_{1,3} \\ \dots & u_{2,1}^2 + u_{2,2}^2 & u_{1,3}u_{1,2} + u_{2,3}u_{2,2} \\ \dots & \dots & \sum_{i=1}^{3} u_{3,i}^2 \end{bmatrix}, \tag{6.18}$$

which resembles the Cholesky–Banachiewicz algorithm [159]. Note that we can decompose $\boldsymbol{C}$ to compute its inverse using the same approach, i.e. $\boldsymbol{C}^{-1} = \boldsymbol{U}^{-1}(\boldsymbol{U}^T)^{-1}$.

DSMGPs with multiple children under a sum node naturally have overlapping local GPs, i.e. GPs share parts of their input domain. This property can be utilised to share solutions of the Cholesky decompositions, which can speed up computations.

Let us consider the case in which two leaves, denoted as L and L', are such that $\mathcal{X}_{\mathsf{L}'} \subset \mathcal{X}_{\mathsf{L}}$ and $\#\boldsymbol{X}_{(\mathsf{L}')} < \#\boldsymbol{X}_{(\mathsf{L})}$. We will discuss two scenarios in which the kernel matrix of L' can be obtained in an efficient and numerically stable way. In the first scenario the kernel matrix $\boldsymbol{K}_{\boldsymbol{X}_{(\mathsf{L}')},\boldsymbol{X}_{(\mathsf{L}')}}$ is a sub-matrix of the kernel matrix of L and $[\boldsymbol{K}_{\boldsymbol{X}_{(\mathsf{L}')},\boldsymbol{X}_{(\mathsf{L}')}}]_{1,1} = [\boldsymbol{K}_{\boldsymbol{X}_{(\mathsf{L})},\boldsymbol{X}_{(\mathsf{L})}}]_{1,1}$. The upper triangular matrix of the Cholesky decomposition for the kernel matrix of L' is a sub-matrix of the decomposition for the kernel matrix of L in this case.

Let $\boldsymbol{U}_{\mathsf{L}}$ and $\boldsymbol{U}_{\mathsf{L}'}$ denote the upper triangular matrices of the Cholesky decomposition for the respective kernel matrices. Then,

$$\boldsymbol{U}_{\mathsf{L}} = \begin{bmatrix} \boldsymbol{U}_{\mathsf{L}'} & \boldsymbol{v}^T \\ \boldsymbol{v} & \tilde{\boldsymbol{U}} \end{bmatrix}, \tag{6.19}$$

where the vector $\boldsymbol{v} \in \mathbb{R}^P$ and $\tilde{\boldsymbol{U}} \in \mathbb{R}^{P \times P}$ with P being the additional dimensions contained in $\boldsymbol{U}_{\mathsf{L}}$. Thus, we can copy the respective sub-matrix to obtain $\boldsymbol{U}_{\mathsf{L}'}$.

In the second scenario both kernel matrices share the last column and row and we have

$$\boldsymbol{K}_{\boldsymbol{X}_{(\mathsf{L})},\boldsymbol{X}_{(\mathsf{L})}} = \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & \dots & k_{1,N_{\mathsf{L}}} \\ k_{2,1} & k_{2,2} & k_{2,3} & \dots & k_{2,N_{\mathsf{L}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{N_{\mathsf{L}},1} & k_{N_{\mathsf{L}},2} & k_{N_{\mathsf{L}},3} & \dots & k_{N_{\mathsf{L}},N_{\mathsf{L}}} \end{bmatrix}, \tag{6.20}$$

[33] The Cholesky decomposition can equivalently be defined as a factorisation into a lower triangular matrix and its conjugate transpose.

and in case of L' we have

$$\boldsymbol{K}_{\boldsymbol{X}_{(\mathsf{L}')},\boldsymbol{X}_{(\mathsf{L}')}} = \begin{bmatrix} k_{1,1} & 0 & k_{1,3} & \dots & k_{1,N_\mathsf{L}} \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{N_\mathsf{L},1} & 0 & k_{N_\mathsf{L},3} & \dots & k_{N_\mathsf{L},N_\mathsf{L}} \end{bmatrix}. \tag{6.21}$$

In such a scenario, we can formulate the solution of the Cholesky decomposition $\boldsymbol{U}^T\boldsymbol{U}$ for the sub-matrix $[\boldsymbol{K}_{\boldsymbol{X}_{\mathsf{L}'},\boldsymbol{X}_{\mathsf{L}'}}]_{3:N_\mathsf{L},3:N_\mathsf{L}}$ in terms of a system of linear equations. The solution can efficiently be obtained by performing rank-1 updates, which scales quadratic with the size P of $\boldsymbol{U}$, i.e. $\mathcal{O}(P^2)$. We refer to Seeger [160] for details on a numerical stable algorithm to perform rank-1 updates or downdates.

Note that other scenarios are either a combination of the two discussed scenarios or can be solved by continuing the Cholesky decomposition after applying rank-1 updates. Scenarios that are not covered by those combinations do not result in numerically stable solutions, when solved using rank-1 updates or downdates, and should be solved explicitly. We refer to Seeger [160] for a detailed discussion. The outlined approach can readily be applied in case of noisy observations, i.e. to compute the inverse of $\boldsymbol{C}$.

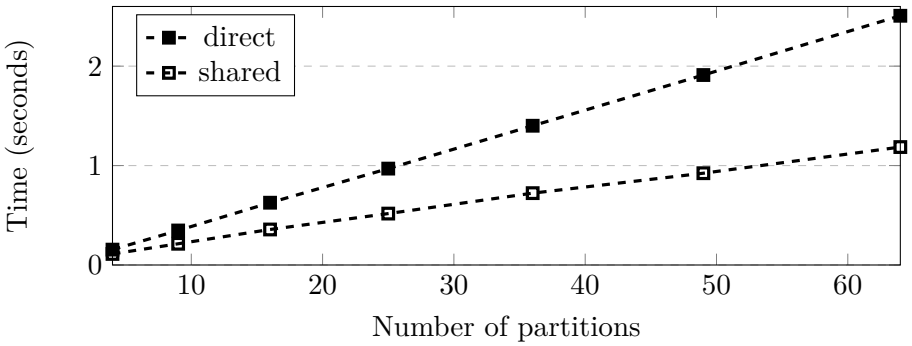


Figure 6.3: Time required to solve the Cholesky decomposition of a DSMGP on a synthetic dataset. The runtime for a direct approach without sharing solutions is denoted as (direct) and the shared approach using (shared). We see that sharing Cholesky decompositions improves the runtime by a factor of two.

We empirically evaluated the performance gains through sharing solutions of the Cholesky decompositions, shown in Figure 6.3. The plot compares the runtime, measured on an i7-6900k CPU @ 3.2 GHz, for a synthetic dataset consisting of $1k$ observations against an increasing number of partitions. We see that sharing Cholesky decompositions reduces the runtime by a factor of two, allowing us to explore twice as many partitions of the input space without additional computational costs.

6.4 Experiments

To assess the performance of DSMGPs, we first compared the approximation error against existing PoE approaches. Subsequently, we evaluated

the predictive performance of DSMGPs compared against state-of-the-art on various benchmark datasets.

To construct the DSMGP structure for each experiment, we used the algorithm outlined in Appendix D.2.1. In short, we construct a hierarchical structure consisting of sum nodes with K_{S} children and uniform weights, i.e. $w_{\mathsf{S},\mathsf{N}} = \frac{1}{K_{\mathsf{S}}}$, and product nodes with K_{P} children by alternating between sum and product nodes. This process terminates and constructs a leaf node once we reached R many repetitions – consecutive sum and product nodes – or the number of observations in the subspace of the input domain is smaller than a pre-defined minimum M. Finally, we equip each GP leaf with a Squared Exponential (SE) kernel-function with Automatic Relevance Detection (ARD) and a *zero* mean-function. Note that we used the same kernel and mean-function for all other methods. To obtain suitable hyperparameters, we performed global hyperparameter optimisation for each model using RMSprop run for $1k$ iterations. We intensionally refrained from performing local fine-tuning of DSMGPs in all experiments, to allow for a fair comparison.

6.4.1 Approximation Error

In the first experiment, we compared the approximation error of DSMGPs against popular expert-based approaches on the motorcycle dataset [161]. Figure 6.6 shows the posterior distribution of a generalize PoE (gPoE) [150], a robust Bayesian Committee Machine (rBCM) [162] and our DSMGP superimposed with the posterior of an exact GP. All models use the same kernel-function as the exact GP and distribute the input domain or the dataset onto local experts with $M = 7$ observations. Note that the gPoE and the rBCM algorithms result in over-conservative predictions or wrong estimates of the mean, specifically, in transition areas. DSMGP, on the other hand, provide an accurate representation of the uncertainties and mean in all regions of the input domain, when used as an approximation to a GP. Note that DSMGPs do not suffer from severe discontinuities and can exploit discontinuities in data when appropriate.

Figure 6.4 quantitatively compares the approximation error on the Kin40k dataset [163], in terms of the Root Mean Squared Error (RMSE), i.e.

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} (\hat{y}_j - y_j)^2}\,, \tag{6.22}$$

where $\hat{y}_j$ is the prediction for the j^{th} test datum and y_j denotes the correct outcome. The DSMGP was constructed using $K_{\mathsf{S}} = 4$, $R = 2$ and $K_{\mathsf{P}} = \sqrt[R]{\frac{N}{M}}$. We see that DSMGPs consistently obtain a lower approximation error than existing approaches, independent of the number of observations per expert.

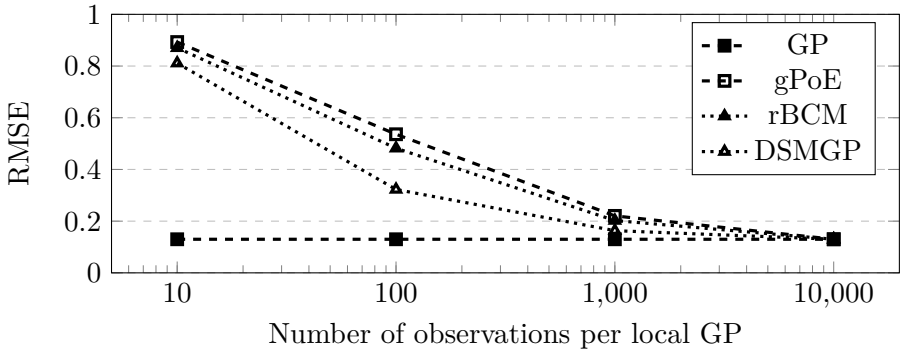


Figure 6.4: Approximation error on the Kin40k dataset.

6.4.2 Quantitative Evaluation

To compare the performance of DSMGPs, we assessed their the predictive performance against an exact GP, linear regression (LR), constant regression (Conts), gPoE, rBCM[34], sparse variational GPs (SVGPs)[35] [164] and structured kernel interpolation (KISS) [165] on the benchmark datasets listed in Table 6.1. Where available, we used the existing training and test set split, and otherwise randomly split the dataset into 70% for training and 30% for testing. We pre-processed each dataset to have zero mean and unit variance – in the inputs and outputs – and used a *zero* mean-function for each approach.

Table 6.1: Statistics of benchmark datasets. For each dataset we list size of the training set N, the test set N_{test}, the input dimensionality D, and output dimensionality P.

Dataset	N	N_{test}	D	P
Airfoil	1,052	451	5	1
Parkinsons	4,112	1,763	16	2
Kin40k	10,000	30,000	8	1
House	15,949	6,835	16	1
Protein	32,011	13,719	9	1
Year	360,742	154,603	90	1
Flight	500,000	200,000	8	2

In the experiments we used $Q = 100$ inducing points and consistently used $M = 100$ observations per expert for each expert-based approach. DSMGPs additionally use $K_{\mathsf{S}} = 4$, while the hyperparameters are estimated on an DSMGP with $K_{\mathsf{S}} = 1$. In case of KISS GPs, we chose the grid size according to the number of data points and used an additive kernel decomposition, as KISS GPs scale exponentially with the dimensionality of the input domain.

To assess the performance of each technique, we followed the approach

[34] We used the implementation in `https://github.com/jopago/GPyBCM`.
[35] We used the implementation in GPyTorch: `https://gpytorch.ai`.

by prior art and computed the Mean Absolute Error (MAE) and the Negative Log Predictive Density (NLPD), i.e.

$$\mathrm{MAE} = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} |\hat{y}_j - y_j| \tag{6.23}$$

$$\mathrm{NLPD} = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} -\log p(y_j \,|\, \boldsymbol{x}_j, \mathcal{D}) \,, \tag{6.24}$$

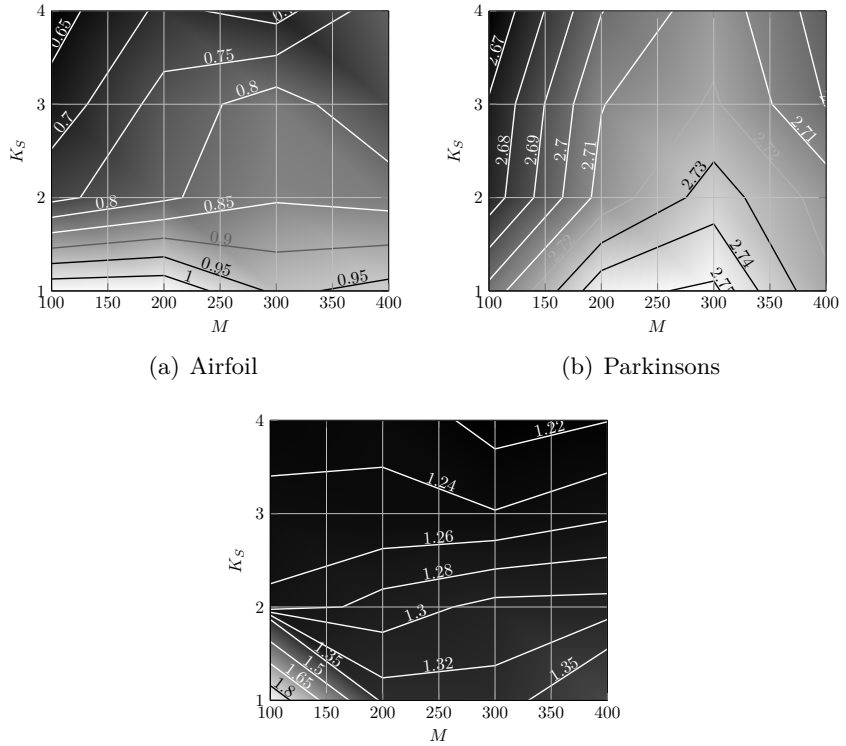
where $\mathcal{D}$ is the training set. Table 6.2 reports the MAE and the NLPD on each dataset. Note that the NLPD for *LR* and *Const* are computed using the inferred noise as the variance of the predictive distribution. We see that DSMGPs consistently outperform other expert-based approaches and often perform competitively or outperform SVGPs. Further, our model consistently captures predictive uncertainties better than previous expert-based approaches resulting in low NLPDs. Note that DSMGPs often have a lower approximation error than SVGPs, when compared to the NLPD and the MAE of an exact GPs.

Dataset		Const	LR	GP	SVGP	KISS	gPoE	rBCM	**DSMGP**
Airfoil	MAE	0.82	0.53	0.50	**0.32**	0.51	0.35	0.34	**0.32**
	NLPD	1.43	1.05	0.99	0.59	1.00	0.72	3.21	**0.57**
Parkin.	MAE	0.85	0.82	0.78	**0.68**	0.78	0.84	0.80	0.74
	NLPD	2.88	2.79	2.73	**2.52**	2.73	4.49	3.80	2.66
Kin40K	MAE	0.81	0.81	0.79	**0.25**	0.79	0.80	0.43	0.78
	NLPD	1.42	1.42	1.39	**0.37**	1.39	2.68	4.14	1.38
House	MAE	0.62	0.49	NA	**0.39**	0.43	0.50	0.40	**0.39**
	NLPD	1.45	1.30	NA	**1.06**	1.10	4.61	4.58	1.11
Protein	MAE	0.89	0.71	NA	0.57	0.64	0.82	0.70	**0.55**
	NLPD	1.41	1.25	NA	**1.11**	1.19	2.38	4.57	**1.11**
Year	MAE	0.74	0.73	NA	**0.57**	NA	0.74	0.74	0.72
	NLPD	1.41	1.39	NA	**1.21**	NA	3.78	1.49	1.38
Flight	MAE	0.56	**0.54**	NA	**0.54**	NA	0.56	0.56	**0.54**
	NLPD	2.87	2.85	NA	**2.80**	NA	8.05	11.51	2.84

Table 6.2: Mean Absolute Error (MAE) and Negative Log Predictive Density (NLPD) of state-of-the-art approaches and DSMGPs on benchmark datasets with $1.5K$ to $500K$ observations. The overall best performance is indicated in bold font and the best NLPD obtained by an expert-based approach is underlined. Smaller values are better.

To assess the effect of the number of children under each sum node (K_{S}) and the minimum number of observations per expert (M) on the performance of DSMGPs, we trained DSMGPs using different settings and computed the average NLPD for each setting on the test set for three datasets. Figure 6.5 shows the respective results in the form of contour plots. We can see that low performance, due to small M, can often be compensated by an increased number of children per sum node.

Additionally, we assessed the runtime for hyperparameter optimisation and compared the timings against existing PoE approaches. Table 6.3



(a) Airfoil

(b) Parkinsons

(c) Kin40k

Figure 6.5: Average test NLPD scores for DSMGPs trained using different number of children under each sum node K_S and different number of observations per expert M. Each test NLPD score has been computed based on 10 independent reruns.

lists the resulting timings, indicating that optimising DSMGPs is competitive to prior work when trained as described above. These timings can be improved by implementing the mentioned algorithms using a distributed framework or by utilising the GPU.

Dataset	GP	gPoE	rBCM	Ours
Airfoil	0.28s	0.05s	0.05s	0.06s
Parkin.	42.61s	1.21s	1.30s	1.27s
Kin40k	107.65s	0.86s	0.87s	0.89s
House	NA	2.55s	2.55s	2.59s
Protein	NA	2.69s	2.70s	2.53s
Year	NA	28.82s	28.90s	22.17s

Table 6.3: Average runtime (seconds) of an iteration of hyperparameter optimisation on an i7-6900k CPU @ 3.2 GHz.

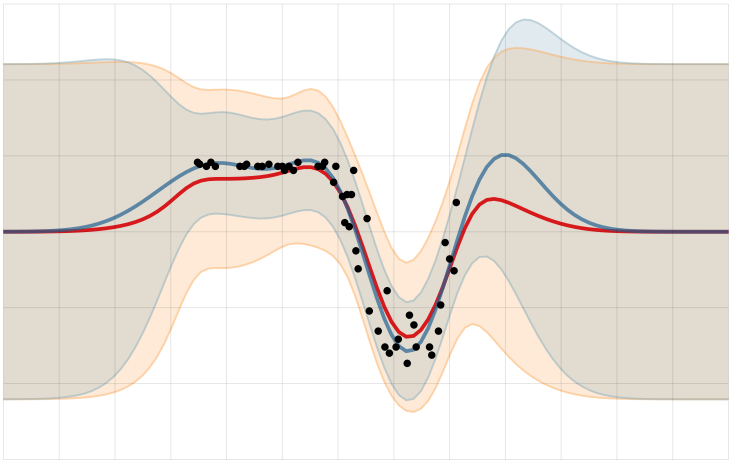
6.5 Related Work

While DSMGPs are a process model in its own right, the primary motivation for this work is to utilise the *divide-and-conquer* approach applied by SPNs for efficient approximation of a full GP. In this sense, the most related approaches are expert-based approaches, which we review in this section.

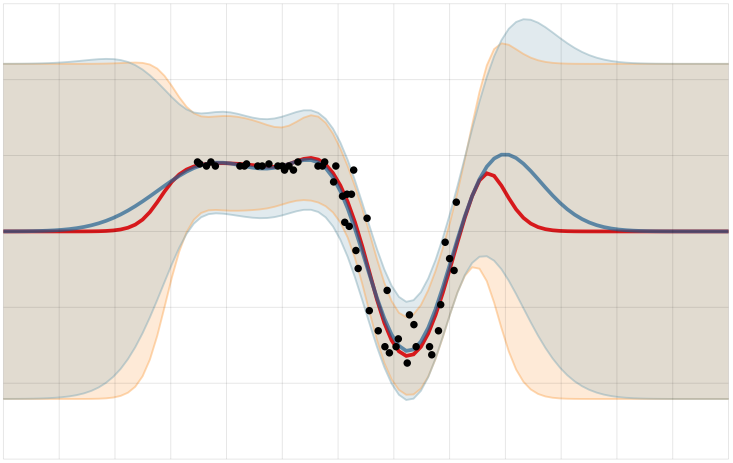
The probably simplest approach is based on so-called Naive-Local-Experts (NLE) [148], [149], [166]. NLEs use a pre-defined, sometimes nested, partition of the input space and model each subspace using an independent GP expert. Due to the independence assumptions, NLEs introduce hard discontinuities. Recent approaches, such as [167], try to ameliorate this effect by imposing continuity constraints onto the local experts using patched GPs. However, this approach suffers from inconsistent variances and does not scale well with the number of boundaries and, consequently, the dimensionality of the input space. In contrast to NLEs and patched GPs, DSMGPs do not rely on a single partition, but instead, perform posterior inference over a large set of partitions, and thus effectively selects the partitions that are well supported by the data.

PoE approaches, such as gPoE, BCM and rBCM, distribute subsets of the data to local experts and aggregate their predictive distributions using a product operation – weighted by some adaptive or non-adaptive scale factors. The key motivation in these approaches is that a product of Gaussians is still Gaussian. However, predictions using PoE base approaches typically do not correspond to inference in some well-defined statistical model. BCMs, on the other hand, justify their formulation as an approximation to posterior inference in GPs, but the introduced approximation error is hard to analyse. Moreover, the product aggregation of expert predictions is Kolmogoroff inconsistent [154], and PoEs are known to have sub-optimal rates of the posterior contraction, and, therefore, can result in uncalibrated predictive uncertainties [153]. In contrast to PoE approaches, our model is a well-defined stochastic process and adequately captures predictive uncertainties.

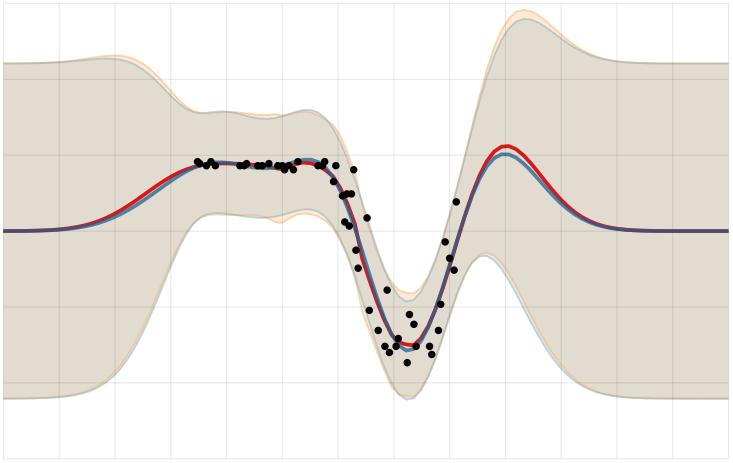
The MoE model [168] is a sound probabilistic model, defined as a mixture of GP experts and a so-called gating network, which dynamically assigns data to GPs. One of the most prominent variants is the infinite MoE model [152], which removes the i.i.d. assumption of the MoE and uses a Dirichlet process as gating network. Alternative formulations and improvements of the infinite MoE model can be found in [169], [170]. However, while MoE models are designed to capture multi-modality and non-stationarity, they usually lack tractable inference. Consequently, they inherently rely on approximate posterior inference, which hampers their application to large data domains. In contrast to MoE models, our approach does not use a gating network but performs inference over a large set of pre-determined partitions of the input space. Crucially, and unlike in MoE models, posterior inference in our model can be performed exactly and efficiently. Note that the approach by Zhang and Williamson [171], which was published around the time of this thesis, is similar in spirit to DSMGPs, but does not utilise exact posterior inference.



(a) generalized PoE



(b) robust BCM



(c) DSMGP

Figure 6.6: Comparison of generalized PoE, robust BCM and DSMGP (orange) against an exact GP (blue).

7

Discussion & Future Work

In this thesis, we have developed various novel techniques for learning Sum-Product Networks (SPNs) in complex modelling scenarios. At the core of this thesis are principled approaches, often leveraging the Bayesian approach, to i) learn SPNs in case of semi-supervision, ii) obtain SPNs structures in a principled way, and iii) extend SPNs for non-linear nonparametric regression analysis. We will first summarise the main findings of this thesis and then discuss future directions and open questions.

In many domains, it is cheap to acquire unlabelled data while obtaining class labels for supervised learning is expensive. Often these domains not only lack in labelled training data but also come with strong requirements on the learner. We have shown that the Contrastive Pessimistic Likelihood (CPL), which has originally been proposed for linear discriminant analysis [35], is a viable objective for safe semi-supervised learning in SPNs [95]. In Chapter 4, we have extended the CPL objective for generative and discriminative learning of semi-supervised SPNs. By maximising the CPL as a semi-supervised objective, we can guarantee that the semi-supervised SPN cannot degenerate with an increase in unlabelled data, making semi-supervised SPNs an attractive approach in safety-critical domains. In various experiments, we have shown that semi-supervised SPNs perform favourably compared to other semi-supervised methods and are sometimes on par with an SPN trained on the fully labelled dataset. In addition to semi-supervised parameter learning, we have examined the inherent acceleration effects of gradient-based optimisation in SPNs [72]. In particular, we have shown that overparameterised SPNs exhibit similar dynamics as observed in linear neural networks [90] and that gradient-based optimisation corresponds to optimisation with an adaptive and time-varying learning rate and momentum term. Therefore, gradient-based optimisation of overparameterised SPNs results in implicit acceleration effects, resulting in faster convergence. We have shown that this effect can indeed be observed across different datasets.

Besides our contributions on parameter learning, we have introduced the first principled structure learning approach for SPNs [157]. For this, we have developed a novel representation of SPNs, by decomposing the problem of finding a suitable SPN structure into i) laying out a computational graph, and ii) learning the so-called scope-function over the graph. The first is rather unproblematic and akin to neural network architecture validation. The second part, i.e. learning a scope-function, is arguably the more difficult task, as the scope-function has to respect the usual structural constraints in SPNs, i.e. completeness and decomposability. Even though representing and learning the scope-function is somewhat

involved in general, we have shown that we can find a natural parametrisation for an important and widely used case of SPNs, i.e. those defined over a tree-structured region graph, using discrete latent variables. Further, we have shown how to incorporate these structural parameters into a Bayesian model, allowing simultaneous structure and parameter learning using joint Bayesian posterior inference. For this, we have introduced a Gibbs sampling scheme that exploits conjugacy in the model formalism and leverages efficient ancestral sampling in SPNs. We observed in various experiments that Bayesian SPNs often improve test likelihoods over greedy SPN learners. Further, since the Bayesian framework protects against overfitting, we can evaluate hyperparameters directly on the Bayesian model evidence, waiving the need for a separate validation set, which is especially beneficial in low data regimes. Moreover, we have shown that Bayesian SPNs can be directly applied to heterogeneous domains and can learn structures robustly in the context of missing data. In addition, we introduced two formulations for nonparametric SPNs, enabling SPN structure and parameter learning in domains where the model complexity is expected to increase with growing amounts of data.

In the last part of this thesis, we have developed SPNs as a stochastic process model called Deep Structured Mixtures of Gaussian Processes (DSMGPs). For this, we first discussed the probability measure induced by an SPN, and subsequently introduced DSMGPs by equipping SPNs with Gaussian measures at the leaves. Because of the one-to-one relationship between Gaussian measures and Gaussian processes for the space of continuously differentiable functions on any real interval, our approach directly extends SPNs to infinitely many random variables and establishes SPNs as a well-defined stochastic process model. The resulting model inherits tractable inference from SPNs and exact posterior inference in Gaussian processes and has an interesting interpretation as exact Bayesian model averaging over a rich set of naive-local expert models. We have shown that DSMGPs not only allow exact and efficient posterior inference but also outperform existing expert-based approaches and obtain low approximation errors when used as an approximation to a Gaussian process. Furthermore, DSMGPs have attractive computation costs for hyperparameter optimisation and allow fine-tuning of local hyperparameters to account for nonstationarities in the dataset. Last but not least, we have shown that the structure of the DSMGP can be exploited to speed up computations by sharing solutions of the Cholesky decompositions of the experts.

Even though this thesis introduces many important advances in SPNs, various questions remain to be addressed in future work. We will highlight the most important open questions and provide pointers to possible future directions of research.

- Optimising the CPL objective has certain shortcomings. Most importantly, the pessimistic strategy of the CPL often results in a very conservative learner and prevents the learner from improving upon the supervised SPN in certain scenarios. An important direction of research is, therefore, to examine strategies that trade-off between optimism and pessimism in the CPL objective.

- Another shortcoming of the current semi-supervised approach for SPNs is that the CPL does not account for uncertainties about the

model parameters. However, in safety-critical data domains, it is particularly relevant to incorporate these uncertainties instead of relying on a single point estimate for the model parameters.

- An important future direction for structure learning in SPNs is to derive fast approximation schemes for Bayesian SPNs. Even though our Markov chain Monte Carlo approach scales well with moderately sized datasets, it does not allow efficient inference in large scale applications. Moreover, it is often more desirable to obtain a variational approximation to the true posterior, thus allowing for a more compact description of the posterior distribution. However, obtaining a variational inference scheme for Bayesian SPNs is non-trivial. We believe that exploiting recent advances in Bayesian neural networks, such as continuous relaxations using the Gumble-softmax trick, could be a promising future direction.

- Even though we have introduced techniques to obtain nonparametric formulations for SPN, efficient inference in more flexible formulations is still an open question.

- Last but not least, we have shown that SPNs can result in an effective stochastic process model. However, the interpretability of the posterior distribution of DSMGPs and the capability to infer kernel-functions has yet to be explored more extensively.

The results of this thesis have shown that SPNs are an incredibly flexible and versatile model formalism. Therefore, we want to conclude this thesis with some futuristic ideas on how to advance probabilistic machine learning.

- We have shown that SPNs can be an effective approach for nonparametric regression analysis. However, can we also utilise SPNs as a surrogate model for other nonparametric models, e.g. to perform efficient inference in models with a nonparametric prior on partitions?

- Can we exploit SPNs for agent-based systems and reinforcement learning? In particular, can we utilise exact integration in nested settings?

- In SPNs we utilise decomposability to guarantee that global integration simplifies to local integration. Can we utilise a similar strategy to automatically accelerate probabilistic inference in general-purpose probabilistic programming systems?

A

Appendix: Sum-Product Networks

A.1 Compiling Region Graphs to Sum-Product Networks

The following pseudocode outlines how to obtain an SPN from a region graph. Note that the algorithm assumes all parameters θ and $\boldsymbol{w}$ to exists, but can also be written in form of a generative process if used in the context of Bayesian learning. Further note that we assume each partition node to have exactly two children. This assumption can of course be relaxed if necessary.

Algorithm 1: Convert Region-Graph To SPN

Input: $I > 0$, $J > 0$, θ, $\boldsymbol{w}$
Output: $\mathcal{S}$
Function *getNodes(R: Atomic region)*
  $\mathbf{L} \leftarrow \emptyset$;
  for $i = 1, \ldots, I$ **do**
    $\mathbf{L} \leftarrow \mathbf{L} \cup p(\boldsymbol{x} \,|\, \theta_i)$
  return $\mathbf{L}$
Function *getNodes(P: Partition)*
  $\mathbf{P} \leftarrow \emptyset$;
  $R_1, R_2 \leftarrow \mathrm{ch}(P)$;
  $\mathbf{N}_1 \leftarrow$ *getNodes*(R_1) ;
  $\mathbf{N}_2 \leftarrow$ *getNodes*(R_2) ;
  foreach $\mathsf{N} \in \mathbf{N}_1$ **do**
    foreach $\mathsf{N}' \in \mathbf{N}_2$ **do**
      $\mathbf{P} \leftarrow \mathbf{P} \cup \mathsf{N} \times \mathsf{N}'$
  return $\mathbf{P}$
Function *getNodes(R: Region)*
  $\mathbf{S} \leftarrow \emptyset$;
  for $j = 1, \ldots, J$ **do**
    $\mathbf{N} \leftarrow \emptyset$ **foreach** $P \in \mathrm{ch}(R)$ **do**
      $\mathbf{N} \leftarrow \mathbf{N} \cup$ *getNodes*(P)
    $\mathbf{S} \leftarrow \mathbf{S} \cup \sum^{\mathsf{N} \in \mathbf{N}} w_{\mathsf{N}} \mathsf{N}$
  return $\mathbf{S}$

B

Appendix: Safe Semi-Supervised Learning

The following pseudocode illustrates an implementation of the learnMCP-SPN algorithm for safe-semi-supervised learning of SPNs.

Algorithm 2: Learn MCP-SPN Parameters

Input: SPN structure $\mathcal{S}$, labelled data $\mathcal{D}$, unlabelled data $\mathcal{U}$
Output: Learned parameters and soft labels
`// learn supervised SPN`
if *generative* **then**
  $\theta_{\text{sup}} \leftarrow \arg\max_{\theta\in\Theta} \mathcal{L}(\theta \mid \mathcal{D})$
else
  $\theta_{\text{sup}} \leftarrow \arg\max_{\theta\in\Theta} \mathcal{C}(\theta \mid \mathcal{D})$
`// initialize soft labels`
if *optimistic* **then**
  $q_{k,m} \leftarrow \frac{\mathcal{S}_k(\boldsymbol{u}_m)}{\mathcal{S}(\boldsymbol{u}_m)} \;\; \forall k \forall m$
else
  $\boldsymbol{q}_m \sim \text{Dir}(\frac{1}{K}, \ldots, \frac{1}{K}) \;\; \forall m$
`// learn safe semi-supervised SPN`
repeat
  `// optimistic parameter learning`
  if *generative* **then**
    $\theta_{\text{CPL}} \leftarrow \arg\max_{\theta\in\Theta} \mathcal{L}(\theta \mid \mathcal{D}, \mathcal{U}, \boldsymbol{q})$
  else
    $\theta_{\text{CPL}} \leftarrow \arg\max_{\theta\in\Theta} \mathcal{C}(\theta \mid \mathcal{D}, \mathcal{U}, \boldsymbol{q})$
  `// pessimistic soft label adjustment`
  $\boldsymbol{q} \leftarrow \boldsymbol{q} - \alpha\nabla_{\boldsymbol{q}}$;
  Project each $\boldsymbol{q}_m$ back onto the Δ_{K-1} Simplex
until *convergence or early stopping*;
return $\theta_{CPL}, \boldsymbol{q}$

C

Appendix: Bayesian Learning of Sum-Product Networks

C.1 Heterogeneous Experiments

To conduct the heterogeneous data experiments, we introduce mixtures over likelihood model for each leaf node used the following likelihood and prior constructions in the experiment. Note that all of the used priors distributions are conjugate to their respective observation model and we can therefore obtain their posterior in closed-form.

Continuous: Gaussian likelihood $\mathcal{N}(\mu, \sigma^2)$

- Prior on $\sigma^2 \sim \Gamma^{-1}(\alpha_0, \beta_0)$ with $\alpha_0 = 2, \beta_0 = 3$ and with posterior parameters $\alpha_N = \alpha + N/2$ and $\beta_N = \beta_0 + \frac{1}{2}(\mu_0^2 {\nu_0}^{-1} + \sum_{i=1}^N x_i^2 - \mu_N^2 {\nu_N}^{-1})$.
- Prior on $\mu \sim \mathcal{N}(\mu_0, \sigma^2 \nu_0)$ with posterior parameters $\mu_N = \nu_N({\mu_0 \nu_0}^{-1} + N\bar{x})$ and ${\nu_N}^{-1} = {\nu_0}^{-1} + N$ where $\bar{x}$ is the sample mean.

Continuous (positive): Exponential likelihood $\mathrm{Exp}\,(\lambda)$

- Prior on $\lambda \sim \Gamma(\alpha_0, \beta_0)$ with $\alpha_0 = \beta_0 = 1$ and with posterior parameters $\alpha_N = \alpha_0 + N$ and $\beta_N = \beta_0 + \sum_{i=1}^N x_i$.

Discrete: Poisson likelihood $\mathrm{Poisson}\,(\lambda)$

- Prior on $\lambda \sim \Gamma(\alpha_0, \beta_0)$ with $\alpha_0 = \beta_0 = 1$ and with posterior parameters $\alpha_N = \alpha_0 + \sum_{i=1}^N x_i$ and $\beta_N = \beta_0 + N$.

Discrete: Categorical likelihood $\mathrm{Cat}(\boldsymbol{w})$

- Prior on $\boldsymbol{w} \sim \mathrm{Dir}(\alpha_1/K, \ldots, \alpha_K/K)$ with $\alpha_k = 0.1$ and with posterior parameters $\tilde{\alpha_k}/K = \alpha_k/K + \sum_{i=1}^N \mathbb{1}_{\{z_i=k\}}$.

Discrete: Bernoulli likelihood $\mathrm{B}\,(p)$

- Prior on $p \sim \mathrm{Beta}\,(\alpha_0,\ \beta_0)$ with $\alpha_0 = \beta_0 = 0.5$ and with posterior $\alpha_N = \alpha_0 + \sum_{i=1}^N x_i$ and $\beta_N = \beta_0 + N - \sum_{i=1}^N x_i$.

C.2 Statistical Significance Tests

To assess the statistical significance of the reported results we computed the p-value of the Mann-Whitney-U-Test [141]. The Mann-Whitney-U-Test is a nonparametric equivalent of the two sample t-test which does not require the assumption of normal distributions. The respective p-values obtained from the Mann-Whitney-U-Test for Bayesian SPNs and infinite mixtures of Bayesian SPNs are listed in Table C.1.

Dataset	LearnSPN	RAT-SPN	ID-SPN	LearnSPN	RAT-SPN	ID-SPN
NLTCS	0.726	0.573	0.291	0.887	0.950	0.123
MSNBC	0.634	0.420	0.474	0.911	0.173	0.842
KDD	0.792	0.044	0.505	0.755	0.050	0.472
Plants	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$
Audio	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$
Jester	$\underline{< 0.001}$	$\underline{< 0.001}$	0.908	$\underline{0.004}$	0.885	$\underline{0.001}$
Netflix	0.100	0.924	0.455	0.107	0.944	0.442
Accidents	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$
Retail	$\underline{< 0.001}$	$\underline{0.002}$	0.023	$\underline{< 0.001}$	$\underline{0.008}$	0.020
Pumsb-star	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	0.025	$\underline{< 0.001}$
DNA	$\underline{< 0.001}$	$\underline{0.001}$	0.084	$\underline{< 0.001}$	$\underline{< 0.001}$	0.023
Kosarak	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$
MSWeb	0.721	$\underline{0.005}$	0.030	0.354	0.047	$\underline{0.002}$
Book	$\underline{< 0.001}$	0.035	0.124	0.034	0.845	0.442
EachMovie	0.270	0.228	0.390	0.275	0.242	0.411
WebKB	$\underline{< 0.001}$	$\underline{0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	$\underline{< 0.001}$	0.645
Reuters-52	0.089	0.703	0.998	0.079	0.638	0.904
20 Newsgrp	0.846	0.508	0.969	0.326	0.636	0.214
BBC	$\underline{0.002}$	0.288	0.866	$\underline{0.002}$	0.335	0.795
AD	$\underline{0.004}$	0.635	0.774	$\underline{0.004}$	0.097	0.769

Table C.1: Mann-Whitney-U test p-values of Bayesian SPNs (left) and infinite mixtures of Bayesian SPNs (right), compared to LearnSPN, RAT-SPNs and ID-SPN. Values below 0.01 *are underlined*

C.3 Reported Configurations and Respective Runtime

We computed the average runtime for a single MCMC iteration measures for an i7-6900k CPU @ 3.2 GHz. The respective runtime for each dataset, measures for the computational graph used to report the results in the paper, are listed in Table C.2. Note that these timings vary depending on the dataset size, the number of dimensions and the complexity of the computational graph.

Dataset	runtime (sec.)	I	J	M	L
NLTCS	4.03	5	10	8	2
MSNBC	33.87	5	5	4	4
KDD	43.05	5	10	8	2
Plants	24.39	5	10	8	4
Audio	3.50	5	5	4	4
Jester	5.12	5	10	4	4
Netflix	7.53	5	10	4	4
Accidents	27.55	10	10	8	4
Retail	3.46	10	10	4	2
Pumsb-star	4.15	10	10	8	2
DNA	7.92	5	10	8	4
Kosarak	10.43	10	10	8	2
MSWeb	4.91	5	5	8	2
Book	5.23	10	10	8	2
EachMovie	30.23	5	10	8	4
WebKB	3.61	10	10	8	2
Reuters-52	6.37	10	10	8	2
20 Newsgrp	11.02	5	10	8	2
BBC	3.56	5	10	8	2
AD	1.05	5	5	2	2

Table C.2: Number of sum nodes per region (I), number of leaves per atomic region (J), number of partitions (M), number of layers (L) and runtime (in seconds) for and iteration of MCMC sampling.

C.4 Extended Results Table

Table C.3: Average test log-likelihoods on discrete datasets using SOTA, Bayesian SPNs (ours) and infinite mixtures of SPNs (ours$^\infty$). Overall best result is in bold.

Dataset	LearnSPN	RAT-SPN	CCCP	ID-SPN	btd-PSDD	btd-CNet	btd-SPN	ours	ours$^\infty$
NLTCS	−6.11	−6.01	−6.03	−6.02	−5.99	**−5.97**	−6.01	−6.00	−6.02
MSNBC	−6.11	−6.04	−6.05	−6.04	−6.04	**−6.03**	−6.04	−6.06	**−6.03**
KDD	−2.18	−2.13	−2.13	−2.13	**−2.11**	−2.13	−2.12	−2.12	−2.13
Plants	−12.98	−13.44	−12.87	−12.54	−13.02	**−11.84**	−12.54	−12.68	−12.94
Audio	−40.50	−39.96	−40.02	−39.79	−39.94	**−39.39**	−39.79	−39.77	−39.79
Jester	−53.48	−52.97	−52.88	−52.86	**−51.29**	−52.21	−52.80	−52.42	−52.86
Netflix	−57.33	−56.85	−56.78	−56.36	**−55.71**	−55.93	−56.36	−56.31	−56.80
Accidents	−30.04	−35.49	−27.70	**−26.98**	−30.16	−29.27	−27.70	−34.10	−33.89
Retail	−11.04	−10.91	−10.92	−10.85	**−10.72**	−10.81	−10.85	−10.83	−10.83
Pumsb-star	−24.78	−32.53	−24.23	**−22.41**	−26.12	−23.44	**−22.41**	−31.34	−31.96
DNA	−82.52	−97.23	−84.92	−81.21	−88.01	**−81.07**	−81.21	−92.95	−92.84
Kosarak	−10.99	−10.89	−10.88	−10.60	**−10.52**	−10.60	−10.60	−10.74	−10.77
MSWeb	−10.25	−10.12	−9.97	−9.73	−9.89	**−9.62**	−9.73	−9.88	−9.89
Book	−35.89	−34.68	−35.01	−34.14	−34.97	−34.46	−34.14	**−34.13**	−34.34
EachMovie	−52.49	−53.63	−52.56	−51.51	−56.43	**−50.34**	−51.49	−51.66	−50.94
WebKB	−158.20	−157.53	−157.49	−151.84	−161.09	**−149.20**	−151.84	−156.02	−157.33
Reuters-52	−85.07	−87.37	−84.63	−83.35	−89.61	**−81.87**	−83.35	−84.31	−84.44
20 Newsgrp	−155.93	−152.06	−153.21	−151.47	−155.97	**−151.02**	−151.47	−151.99	−151.95
BBC	−250.69	−252.14	−248.60	−248.93	−253.19	**−229.21**	−248.50	−249.07	−254.69
AD	−19.73	−48.47	−27.20	−19.05	−30.49	**−14.00**	−19.05	−63.80	−63.80

D

Appendix: Deep Structured Mixture of Gaussian Processes

D.1 Datasets

If available we used the existing training set / testing set splits and otherwise randomly split the dataset into 70% for training and 30% for testing. We pre-processed each dataset to have zero mean and unit variance – in the inputs and outputs – and used a zero mean function for each approach.

Table D.1: Statistics of benchmark datasets. For each dataset we list the number of training examples N (train), the number of test examples N (test), the number of input dimensions (D) and the number of output dimensions (P).

Dataset	**N (train)**	**N (test)**	**D**	**P**
Airfoil	1,052	451	5	1
Parkin.	4,112	1,763	16	2
Kin40k	10,000	30,000	8	1
House	15,949	6,835	16	1
Protein	32,011	13,719	9	1
Year	360,742	154,603	90	1
Flight	500,000	200,000	8	2

D.2 Algorithms

In the following text, we will provide pseudocode implementations of the algorithms for structure construction and posterior inference.

D.2.1 Structure Construction

The following pseudocode illustrates the recursively construction of a deep structured mixture of Gaussian processes containing sum nodes with K_{S} many children and product nodes with K_{P} many children. The argument M controls the minimum number of observations per GP expert.

Algorithm 3: Construction of a Deep Structured Mixture of Gaussian Processes

Input: $K_{\mathsf{S}}, K_{\mathsf{P}}, M$
Output: $\mathcal{S}$
Function *buildGP($\mathcal{X}$)*
  Equip L with a Gaussian process expert on the domain $\mathcal{X}$
  return L
Function *buildSumNode($\mathcal{X}$)*
  $\boldsymbol{w}_{\mathsf{S}} \leftarrow \{\frac{1}{K_{\mathsf{S}}}\}_{k=1}^{K_{\mathsf{S}}}$;
  Let $u_1, \ldots, u_D$ represent the sample variance in $\mathcal{X}$ for each data dimension;
  for $k = 1, \ldots, K_{\mathsf{S}}$ **do**
    $d \sim$ with probability proportional to u_d;
    $\mathsf{S} \leftarrow \mathsf{S} + w_k$ *buildProductNode($\mathcal{X}, d$)*
  return S
Function *buildProductNode($\mathcal{X}, d$)*
  $m \leftarrow$ sample mean or median of observations in $\mathcal{X}$ for dimension d;
  $l \leftarrow$ lower bound of $\mathcal{X}$ for dimension d;
  $u \leftarrow$ upper bound of $\mathcal{X}$ for dimension d;
  $v \leftarrow u - l$;
  for $k = 1, \ldots K_{\mathsf{P}} - 1$ **do**
    $s_k \sim \lambda(v\text{Beta}\,(2,\ 2) + l) + (1 - \lambda)m$
  Sort $s_1, \ldots, s_{K_{\mathsf{P}}}$ such that $l \leq r_1 \leq r_2 \leq \ldots, \leq r_{K_{\mathsf{S}}} \leq u$;
  $\tilde{l} \leftarrow l$;
  for $k = 1, \ldots, K_{\mathsf{P}} - 1$ **do**
    $\tilde{u} \leftarrow r_k$;
    $\tilde{\mathcal{X}} \leftarrow$ sub-domain of $\mathcal{X}$ such that upper and lower bounds for d are equal to $\tilde{u}, \tilde{l}$, respectively ;
    if *Number of observations in* $\mathcal{X} > M$ **then**
      $\mathsf{P} \leftarrow \mathsf{P}\times$ *buildSumNode($\tilde{\mathcal{X}}$)*
    else
      $\mathsf{P} \leftarrow \mathsf{P}\times$ *buildGP($\tilde{\mathcal{X}}$)*
  return P

D.2.2 Exact Posterior Inference

The following pseudocode illustrates exact posterior inference in deep structured mixture of Gaussian processes. The procedure recursively performs exact posterior updates and starts at the root node of the network. Note that for reasons of numerical stability, an actual implementation of the algorithm will need to perform the operations in log-space.

Algorithm 4: Exact posterior inference

Input: $\mathcal{S}$
Output: posterior probability
Function *infer(*N*)*
  $z \leftarrow 0$;
  if N *is a sum node* **then**
    for $\mathsf{C} \in \mathrm{ch}(\mathsf{N})$ **do**
      $w_{\mathsf{N},\mathsf{C}} \leftarrow w_{\mathsf{N},\mathsf{C}}$ *infer(*C*)*;
      $z \leftarrow z + w_{\mathsf{N},\mathsf{C}}$
    $\boldsymbol{w}_{\mathsf{N}} \leftarrow \boldsymbol{w}_{\mathsf{N}} / z$
  else if N *is a product node* **then**
    $z \leftarrow \prod_{\mathsf{C} \in \mathrm{ch}(\mathsf{N})}$ *infer(*N*)*
  else
    $z \leftarrow p_{\mathsf{N}}(\boldsymbol{y}_{(\mathsf{N})} \,|\, \boldsymbol{X}_{(\mathsf{N})})$
  return z

Bibliography

[1] Z. Ghahramani, 'Probabilistic machine learning and artificial intelligence', *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, 'Generative adversarial nets', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 2672–2680.

[3] D. P. Kingma and M. Welling, 'Auto-encoding variational bayes', in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[4] D. J. Rezende and S. Mohamed, 'Variational inference with normalizing flows', in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 37, 2015, pp. 1530–1538.

[5] E. G. Tabak, E. Vanden-Eijnden *et al.*, 'Density estimation by dual ascent of the log-likelihood', *Communications in Mathematical Sciences*, vol. 8, no. 1, pp. 217–233, 2010.

[6] B. Uria, M. Côté, K. Gregor, I. Murray and H. Larochelle, 'Neural autoregressive distribution estimation', *Journal of Machine Learning Research (JMLR)*, vol. 17, 205:1–205:37, 2016.

[7] A. Vergari, Y. Choi, R. Peharz and G. V. den Broeck, 'Probabilistic circuits: Representations, inference, learning and applications', Tutorial at the Conference on Artificial Intelligence, 2020.

[8] H. Poon and P. M. Domingos, 'Sum-product networks: A new deep architecture', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011, pp. 337–346.

[9] D. Kisa, G. V. den Broeck, A. Choi and A. Darwiche, 'Probabilistic sentential decision diagrams', in *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning KR*, 2014.

[10] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. **Trapp**, X. Shao, K. Kersting and Z. Ghahramani, 'Random sum-product networks: A simple and effective approach to probabilistic deep learning', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[11] Y. Liang, J. Bekker and G. V. den Broeck, 'Learning the structure of probabilistic sentential decision diagrams', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[12] R. Peharz. (14th Jun. 2019). Sum-product networks and deep learning: A love marriage, [Online]. Available: `https://slideslive.com/38917679/sumproduct-networks-and-deep-learning-a-love-marriage` (visited on 17/02/2020).

[13] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. **Trapp**, G. V. den Broeck, K. Kersting and Z. Ghahramani, 'Einsum networks: Fast and scalable learning of tractable probabilistic circuits', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[14] C. J. Butz, J. de S. Oliveira, A. E. dos Santos and A. L. Teixeira, 'Deep convolutional sum-product networks', in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 3248–3255.

[15] H. Zhao, P. Poupart and G. J. Gordon, 'A unified approach for learning the parameters of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 433–441.

[16] R. Peharz, R. Gens, F. Pernkopf and P. Domingos, 'On the latent variable interpretation in sum-product networks', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 10, pp. 2030–2044, 2017.

[17] A. Vergari, A. Molina, R. Peharz, Z. Ghahramani, K. Kersting and I. Valera, 'Automatic Bayesian density analysis', in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[18] A. Rashwan, H. Zhao and P. Poupart, 'Online and distributed Bayesian moment matching for parameter learning in sum-product networks', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016, pp. 1469–1477.

[19] H. Zhao, T. Adel, G. J. Gordon and B. Amos, 'Collapsed variational inference for sum-product networks', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1310–1318.

[20] R. Gens and P. M. Domingos, 'Discriminative learning of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 3248–3256.

[21] Z. Yang, W. Cohen and R. Salakhutdinov, 'Revisiting semi-supervised learning with graph embeddings', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 40–48.

[22] L. Maaloe, C. K. Sonderby, S. K. Sonderby and O. Winther, 'Auxiliary deep generative models', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1445–1453.

[23] A. Rasmus, M. Berglund, M. Honkala, H. Valpola and T. Raiko, 'Semi-supervised learning with ladder networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 3546–3554.

[24] L. Käll, J. D. Canterbury, J. Weston, W. S. Noble and M. J. MacCoss, 'Semi-supervised learning for peptide identification from shotgun proteomics datasets', *Nature methods*, vol. 4, no. 11, pp. 923–925, 2007.

[25] M.-A. Krogel and T. Scheffer, 'Multi-relational learning, text mining, and semi-supervised learning for functional genomics', *Machine Learning*, vol. 57, no. 1-2, pp. 61–81, 2004.

[26] X. Zhang, N. Guan, Z. Jia, X. Qiu and Z. Luo, 'Semi-supervised projective non-negative matrix factorization for cancer classification', *PloS one*, vol. 10, no. 9, e0138814, 2015.

[27] O. Chapelle, B. Schölkopf and A. Zien, 'Semi-supervised learning', *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[28] X. Zhu and A. Goldberg, 'Introduction to semi-supervised learning', *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[29] A. Tarvainen and H. Valpola, 'Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 1195–1204.

[30] D. P. Kingma, S. Mohamed, D. J. Rezende and M. Welling, 'Semi-supervised learning with deep generative models', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 3581–3589.

[31] Y. Li and Z. Zhou, 'Towards making unlabeled data never hurt', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 1, pp. 175–188, 2015.

[32] R. Collobert, F. Sinz, J. Weston and L. Bottou, 'Large scale transductive svms', *Journal of Machine Learning Research*, vol. 7, pp. 1687–1712, 2006.

[33] T. Miyato, S.-i. Maeda, M. Koyama and S. Ishii, 'Virtual adversarial training: A regularization method for supervised and semi-supervised learning', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1979–1993, 2018.

[34] Z. Yang, W. W. Cohen and R. Salakhutdinov, 'Revisiting semi-supervised learning with graph embeddings', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 40–48.

[35] M. Loog, 'Contrastive pessimistic likelihood estimation for semi-supervised classification', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 3, pp. 462–475, 2016.

[36] R. Gens and P. Domingos, 'Learning the structure of sum-product networks', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, pp. 873–880.

[37] S.-W. Lee, M.-O. Heo and B.-T. Zhang, 'Online incremental structure learning of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 220–227.

[38] A. Rooshenas and D. Lowd, 'Learning sum-product networks with direct and indirect variable interactions', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2014, pp. 710–718.

[39] A. Vergari, N. Di Mauro and F. Esposito, 'Simplifying, regularizing and strengthening sum-product network structure learning', in *Proceedings of the European Conference on Machine Learning (ECML)*, 2015, pp. 343–358.

[40] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito and K. Kersting, 'Mixed sum-product networks: A deep architecture for hybrid domains', in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[41] T. Adel, D. Balduzzi and A. Ghodsi, 'Learning the structure of sum-product networks via an SVD-based algorithm', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

[42] A. W. Dennis and D. Ventura, 'Learning the architecture of sum-product networks using clustering on variables', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 2042–2050.

[43] R. Peharz, B. C. Geiger and F. Pernkopf, 'Greedy part-wise learning of sum-product networks', in *Proceedings of the European Conference on Machine Learning (ECML)*, vol. 8189, 2013, pp. 612–627.

[44] A. W. Dennis and D. Ventura, 'Greedy structure search for sum-product networks', in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 932–938.

[45] A. Kalra, A. Rashwan, W.-S. Hsu, P. Poupart, P. Doshi and G. Trimponias, 'Online structure learning for feed-forward and recurrent sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 6944–6954.

[46] P. L. Tan and R. Peharz, 'Hierarchical decompositional mixtures of variational autoencoders', in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 6115–6124.

[47] D. D. Mauá, F. G. Cozman, D. Conaty and C. P. de Campos, 'Credal sum-product networks', in *Proceedings of the International Symposium on Imprecise Probability: Theories and Applications*, vol. 62, 2017, pp. 205–216.

[48] M. Melibari, P. Poupart, P. Doshi and G. Trimponias, 'Dynamic sum product networks for tractable inference on sequence data', in *Proceedings of the International Conference on Probabilistic Graphical Models (PGM)*, vol. 52, 2016, pp. 345–355.

[49] E. Spodarev, *Lecture notes in stochastics ii*, 2017.

[50] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, ser. Adaptive computation and machine learning. MIT Press, 2006, ISBN: 026218253X.

[51] S. Arora, N. Cohen and E. Hazan, 'On the optimization of deep networks: Implicit acceleration by overparameterization', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 244–253.

[52] H. Ge, Y. Chen, M. Wan and Z. Ghahramani, 'Distributed inference for Dirichlet process mixture models', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 2276–2284.

[53] R. L. Schilling, *Measures, integrals and martingales*, 2nd ed. Cambridge University Press, 2017.

[54] A. N. Kolmogoroff, *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer, 1933.

[55] G. Vitali, *Sul problema della misura dei Gruppi di punti di una retta: Nota*. Tip. Gamberini e Parmeggiani, 1905.

[56] F. Hausdrogg, *Grundzüge der Mengenlehre*. Veit & Comp., 1914.

[57] J. Oxtoby, *Measure and Category*. Springer, 1980.

[58] P. Billingsley, *Probability and Measure*, 3rd ed. John Wiley & Sons, 1995.

[59] J. S. Rosenthal, *A first look at rigorous probability theory*, 2nd ed. World Scientific Publishing Company, 2006.

[60] R. Diestel, *Graph Theory*, 5th ed. Springer, 2017.

[61] W. Cheng, S. Kok, H. V. Pham, H. L. Chieu and K. M. A. Chai, 'Language modeling with sum-product networks', in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2014, pp. 2098–2102.

[62] R. Peharz, G. Kapeller, P. Mowlaee and F. Pernkopf, 'Modeling speech with sum-product networks: Application to bandwidth extension', in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 3699–3703.

[63] F. Rathke, M. Desana and C. Schnörr, 'Locally adaptive probabilistic models for global segmentation of pathological OCT scans', in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2017, pp. 177–184.

[64] K. Zheng, A. Pronobis and R. P. N. Rao, 'Learning graph-structured sum-product networks for probabilistic semantic maps', in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018, pp. 4547–4555.

[65] C. J. Butz, A. E. dos Santos, J. de S. Oliveira and J. Stavrinides, 'Efficient examination of soil bacteria using probabilistic graphical models', in *Proceedings of the International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, 2018, pp. 315–326.

[66] K. Stelzner, R. Peharz and K. Kersting, 'Faster attend-infer-repeat with tractable probabilistic models', in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 5966–5975.

[67] A. Darwiche, 'A differential approach to inference in Bayesian networks', *Journal of the ACM (JACM)*, vol. 50, no. 3, pp. 280–305, 2003.

[68] R. Peharz, 'Foundations of sum-product networks for probabilistic modeling', PhD thesis, Graz University of Technology, 2015.

[69] D. Lowd and P. M. Domingos, 'Learning arithmetic circuits', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008, pp. 383–392.

[70] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[71] R. Peharz, S. Tschiatschek, F. Pernkopf and P. M. Domingos, 'On theoretical properties of sum-product networks', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 38, 2015.

[72] M. **Trapp**, R. Peharz and F. Pernkopf, 'Optimisation of overparametrized sum-product networks', in *proceedings of TPM workshop at ICML*, 2019.

[73] H. Zhao, M. Melibari and P. Poupart, 'On the relationship between sum-product networks and bayesian networks', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 116–124.

[74] A. Kimmig, G. V. den Broeck and L. D. Raedt, 'Algebraic model counting', *Journal of Applied Logic*, vol. 22, pp. 46–62, 2017.

[75] A. L. Friesen and P. M. Domingos, 'The sum-product theorem: A foundation for learning tractable models', in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 48, 2016, pp. 1909–1918.

[76] A. W. Dennis and D. Ventura, 'Learning the architecture of sum-product networks using clustering on variables', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 2042–2050.

[77] C. J. Butz, J. de S. Oliveira, A. E. dos Santos and A. L. Teixeira, 'Deep convolutional sum-product networks', in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 3248–3255.

[78] O. Sharir, R. Tamari, N. Cohen and A. Shashua, 'Tensorial mixture models', *CoRR*, vol. abs/1610.04167, 2016.

[79] H. Zhao and G. J. Gordon, 'Linear time computation of moments in sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6894–6903.

[80] A. W. V. d. Vaart, *Asymptotic Statistics*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[81] S. Ruder, 'An overview of gradient descent optimization algorithms', *CoRR*, vol. abs/1609.04747, 2016.

[82] A. G. Baydin, B. A. Pearlmutter, A. A. Radul and J. M. Siskind, 'Automatic differentiation in machine learning: A survey', *Journal of Machine Learning Research*, vol. 18, 153:1–153:43, 2017.

[83] J. Duchi, S. Shalev-Shwartz, Y. Singer and T. Chandra, 'Efficient projections onto the l1-ball for learning in high dimensions', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2008, pp. 272–279.

[84] B. O. Koopman, 'On distributions admitting a sufficient statistic', *Transactions of the American Mathematical Society*, vol. 39, no. 3, pp. 399–409, 1936.

[85] S. Kullback, *Information Theory and Statistics*. Dover Publications Inc., 1959.

[86] A. Rashwan, P. Poupart and Z. Chen, 'Discriminative training of sum-product networks by extended baum-welch', in *Proceedings of the International Conference on Probabilistic Graphical Models (PGM)*, 2018, pp. 356–367.

[87] P. Baldi and K. Hornik, 'Learning in linear neural networks: A survey', *IEEE Transactions in Neural Networks*, vol. 6, no. 4, pp. 837–858, 1995.

[88] S. Arora, N. Cohen, W. Hu and Y. Luo, 'Implicit regularization in deep matrix factorization', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 7411–7422.

[89] A. M. Saxe, J. L. McClelland and S. Ganguli, 'Exact solutions to the nonlinear dynamics of learning in deep linear neural networks', in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[90] S. Arora, N. Cohen and E. Hazan, 'On the optimization of deep networks: Implicit acceleration by overparameterization', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 244–253.

[91] Y. E. Nesterov, 'A method for solving the convex programming problem with convergence rate O(1/k2)', *Soviet Mathematics Doklady*, vol. 269, pp. 543–547, 1983.

[92] N. USDA, *The plantss database (http://plants.usda.gov, may 2011). national plant data team, greensboro*, 2015.

[93] Z. Sun, C. Liu, J. Niu and W. Zhang, 'Discriminative structure learning of sum-product networks for data stream classification', *Neural Networks*, vol. 123, pp. 163–175, 2020.

[94] A. Nicolson and K. K. Paliwal, 'Sum-product networks for robust automatic speaker recognition', *CoRR*, vol. abs/1910.11969, 2019.

[95] M. **Trapp**, T. Madl, R. Peharz, F. Pernkopf and R. Trappl, 'Safe semi-supervised learning of sum-product networks', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[96] S.-W. Lee, M.-O. Heo and B.-T. Zhang, 'Online incremental structure learning of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 220–227.

[97] I. Paris, R. Sanchez-Cauce and F. J. Diez, 'Sum-product networks: A survey', *CoRR*, vol. abs/2004.01167, 2020.

[98] S.-W. Lee, C. Watkins and B.-T. Zhang, 'Non-parametric bayesian sum-product networks', in *ICML Workshop on Learning Tractable Probabilistic Models*, vol. 32, 2014.

[99] M. **Trapp**, R. Peharz, M. Skowron, T. Madl, F. Pernkopf and R. Trappl, 'Structure inference in sum-product networks using infinite sum-product trees', in *Proceedings of BNP workshop at NeurIPS*, 2016.

[100] T. Rahman, P. Kothalkar and V. Gogate, 'Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees', in *Proceedings of the European Conference on Machine Learning (ECML)*, 2014, pp. 630–645.

[101] R. Peharz, R. Gens and P. Domingos, 'Learning selective sum-product networks', in *Proceedings of the LTPM workshop at ICML*, 2014.

[102] N. D. Mauro, A. Vergari, T. M. A. Basile and F. Esposito, 'Fast and accurate density estimation with extremely randomized cutset networks', in *Proceedings of the European Conference on Machine Learning (ECML)*, 2017, pp. 203–219.

[103] T. Rahman, S. Jin and V. Gogate, 'Look ma, no latent variables: Accurate cutset networks via compilation', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019, pp. 5311–5320.

[104] X. Zhu, 'Semi-supervised learning with graphs', PhD thesis, Carnegie Mellon University, 2005.

[105] J. E. van Engelen and H. H. Hoos, 'A survey on semi-supervised learning', *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[106] S. Basu, A. Banerjee and R. J. Mooney, 'Semi-supervised clustering by seeding', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2002, pp. 27–34.

[107] F. G. Cozman, I. Cohen and M. Cirelo, 'Unlabeled data can degrade classification performance of generative classifiers.', in *Proceedings of the International Florida Artificial Intelligence Society Conference*, 2002, pp. 327–331.

[108] M. Betancourt, 'Cruising the simplex: Hamiltonian monte carlo and the dirichlet distribution', in *proceedings of the American Institute of Physics Conference*, vol. 1443, 2012, pp. 157–164.

[109] B. Kalantari, 'Approximating nash equilibrium via multilinear minimax', 2019.

[110] A. Gretton, K. Fukumizu, C. H. Teo, L. Song, B. Schölkopf and A. J. Smola, 'A kernel statistical test of independence', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2007, pp. 585–592.

[111] H. Akaike, 'A new look at the statistical model identification', *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[112] A. K. Jain and M. Law, 'Data clustering: A user's dilemma', in *Proceedings of the International Conference on Pattern Recognition and Machine Intelligence (PReMI)*, 2005, pp. 1–10.

[113] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`.

[114] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[115] Y. Grandvalet and Y. Bengio, 'Semi-supervised learning by entropy minimization', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2004, pp. 529–536.

[116] J. H. Krijthe and M. Loog, 'Implicitly constrained semi-supervised least squares classification', in *Proceedings of International Symposium on Intelligent Data Analysis*, 2015, pp. 158–169.

[117] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[118] J. Suzuki, 'A construction of Bayesian networks from databases based on an MDL principle', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 1993, pp. 266–273.

[119] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[120] D. Heckerman, D. Geiger and D. M. Chickering, 'Learning Bayesian networks: The combination of knowledge and statistical data', *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

[121] G. F. Cooper and E. Herskovits, 'A Bayesian method for the induction of probabilistic networks from data', *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[122] N. Friedman and D. Koller, 'Being Bayesian about network structure. a Bayesian approach to structure discovery in Bayesian networks', *Machine learning*, vol. 50, no. 1-2, pp. 95–125, 2003.

[123] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

[124] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[125] N. L. Hjort, C. Holmes, P. Müller and S. G. Walker, *Bayesian Nonparametrics*. Cambridge University Press, 2010.

[126] D. Blackwell and J. B. MacQueen, 'Ferguson distributions via polya urn schemes', *The Annals of Statistics*, vol. 1, no. 2, pp. 353–355, 1973.

[127] D. M. Blei, T. L. Griffiths and M. I. Jordan, 'The nested Chinese restaurant process and Bayesian nonparametric inference of topic hierarchies', *J. ACM*, vol. 57, no. 2, 7:1–7:30, 2010.

[128] C. E. Rasmussen, 'The infinite gaussian mixture model', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2000, pp. 554–560.

[129] D. W. Scott, 'Multivariate density estimation and visualization', in *Handbook of Computational Statistics*, Springer, 2012, pp. 549–569.

[130] G. M. Reaven and R. G. Miller, 'An attempt to define the nature of chemical diabetes using a multidimensional analysis', *Diabetologia*, vol. 16, no. 1, pp. 17–24, 1979.

[131] R. A. Fisher, 'The use of multiple measurements in taxonomic problems', *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[132] R. M. Neal, 'Markov chain sampling methods for dirichlet process mixture models', *Journal of computational and graphical statistics*, vol. 9, no. 2, pp. 249–265, 2000.

[133] M. D. Escobar and M. West, 'Bayesian density estimation and inference using mixtures', *Journal of the american statistical association*, vol. 90, no. 430, pp. 577–588, 1995.

[134] A. Vehtari and J. Lampinen, 'Bayesian model assessment and comparison using cross-validation predictive densities', *Neural Computation*, vol. 14, no. 10, pp. 2439–2468, 2002.

[135] J. Sethuraman, 'A constructive definition of Dirichlet priors', *Statistica sinica*, pp. 639–650, 1994.

[136] J. W. Paisley, C. Wang, D. M. Blei and M. I. Jordan, 'Nested hierarchical dirichlet processes', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 256–270, 2015.

[137] R. P. Adams, Z. Ghahramani and M. I. Jordan, 'Tree-structured stick breaking for hierarchical data', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2010, pp. 19–27.

[138] S. G. Walker, 'Sampling the dirichlet mixture model with slices', *Communications in Statistics—Simulation and Computation®*, vol. 36, no. 1, pp. 45–54, 2007.

[139] C. E. Rasmussen and Z. Ghahramani, 'Occam's Razor', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2001, pp. 294–300.

[140] P. Jaini, A. Ghose and P. Poupart, 'Prometheus: Directly learning acyclic directed graph structures for sum-product networks', in *Proceedings of the International Conference on Probabilistic Graphical Models (PGM)*, 2018, pp. 181–192.

[141] H. B. Mann and D. R. Whitney, 'On a test of whether one of two random variables is stochastically larger than the other', *The Annals of Mathematical Statistics*, pp. 50–60, 1947.

[142] B. Marlin, 'Missing data problems in machine learning', PhD thesis, University of Toronto, 2008.

[143] L. Beretta and A. Santaniello, 'Nearest neighbor imputation algorithms: A critical evaluation', *BMC medical informatics and decision making*, vol. 16, no. 3, p. 74, 2016.

[144] H. Liu, Y. Ong, X. Shen and J. Cai, 'When gaussian process meets big data: A review of scalable gps', *CoRR*, vol. abs/1807.01065, 2018.

[145] F. Wang, J. M. Decker, X. Wu, G. M. Essertel and T. Rompf, 'Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 10 201–10 212.

[146] M. K. Titsias, 'Variational learning of inducing variables in sparse gaussian processes', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009, pp. 567–574.

[147] D. R. Burt, C. E. Rasmussen and M. van der Wilk, 'Rates of convergence for sparse variational gaussian process regression', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019, pp. 862–871.

[148] S. Vasudevan, F. T. Ramos, E. Nettleton, H. F. Durrant-Whyte and A. Blair, 'Gaussian process modeling of large scale terrain', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 1047–1053.

[149] H.-M. Kim, B. K. Mallick and C. C. Holmes, 'Analyzing nonstationary spatial data using piecewise gaussian processes', *Journal of the American Statistical Association*, vol. 100, no. 470, pp. 653–668, 2005.

[150] Y. Cao and D. J. Fleet, 'Generalized product of experts for automatic and principled fusion of gaussian process predictions', *CoRR*, vol. abs/1410.7827, 2014.

[151] V. Tresp, 'A bayesian committee machine', *Neural Computation*, vol. 12, no. 11, pp. 2719–2741, 2000.

[152] C. E. Rasmussen and Z. Ghahramani, 'Infinite mixtures of gaussian process experts', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2001, pp. 881–888.

[153] B. Szabó and H. van Zanten, 'An asymptotic analysis of distributed nonparametric methods', *Journal of Machine Learning Research*, vol. 20, 87:1–87:30, 2019.

[154] Y. K. Samo and S. J. Roberts, 'String and membrane gaussian processes', *Journal of Machine Learning Research (JMLR)*, vol. 17, 131:1–131:87, 2016.

[155] B. S. Rajput and S. Cambanis, 'Gaussian processes and gaussian measures', *The Annals of Mathematical Statistics*, pp. 1944–1952, 1972.

[156] W. H. Press, *Numerical recipes in C++: the art of scientific computing (second ed.)* Cambridge University Press, 2002.

[157] M. **Trapp**, R. Peharz, H. Ge, F. Pernkopf and Z. Ghahramani, 'Bayesian learning of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 6344–6355.

[158] V. Tolvanen, P. Jylänki and A. Vehtari, 'Expectation propagation for nonstationary heteroscedastic gaussian process regression', in *proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, 2014, pp. 1–6.

[159] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical recipes in c*, 1988.

[160] M. Seeger, 'Low rank updates for the cholesky decomposition', University of California at Berkeley, Tech. Rep., 2008.

[161] B. W. Silverman, 'Some aspects of the spline smoothing approach to non-parametric regression curve fitting', *Journal of the Royal Statistical Society*, vol. 47, no. 1, pp. 1–52, 1985.

[162] M. P. Deisenroth and J. W. Ng, 'Distributed gaussian processes', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 1481–1490.

[163] M. W. Seeger, C. K. I. Williams and N. D. Lawrence, 'Fast forward selection to speed up sparse gaussian process regression', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

[164] Y. Gal, M. van der Wilk and C. E. Rasmussen, 'Distributed variational inference in sparse gaussian process regression and latent variable models', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 3257–3265.

[165] A. G. Wilson and H. Nickisch, 'Kernel interpolation for scalable structured gaussian processes (KISS-GP)', in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 37, 2015, pp. 1775–1784.

[166] R. B. Gramacy and H. K. H. Lee, 'Bayesian treed gaussian process models with an application to computer modeling', *Journal of the American Statistical Association*, vol. 103, no. 483, pp. 1119–1130, 2008.

[167] C. Park and J. Z. Huang, 'Efficient computation of gaussian process regression for large spatial data sets by patching local gaussian processes', *Journal of Machine Learning Research (JMLR)*, vol. 17, 174:1–174:29, 2016.

[168] V. Tresp, 'Mixtures of gaussian processes', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2000, pp. 654–660.

[169] C. W. L. Gadd, S. Wade and A. Boukouvalas, 'Enriched mixtures of gaussian process experts', *CoRR*, vol. abs/1905.12969, 2019.

[170] E. Meeds and S. Osindero, 'An alternative infinite mixture of gaussian process experts', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2005, pp. 883–890.

[171] M. M. Zhang and S. A. Williamson, 'Embarrassingly parallel inference for gaussian processes', *Journal of Machine Learning Research*, vol. 20, M. Opper, Ed., pp. 1–26, 2019.

[172] M. Park, G. Horwitz and J. W. Pillow, 'Active learning of neural response functions with gaussian processes', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2011, pp. 2043–2051.

[173] J. W. Ng and M. P. Deisenroth, 'Hierarchical mixture-of-experts model for large-scale gaussian process regression', *CoRR*, vol. abs/1412.3078, 2014.

[174] B. Zoph and Q. V. Le, 'Neural architecture search with reinforcement learning', in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[175] Y. Bengio, N. Léonard and A. C. Courville, 'Estimating or propagating gradients through stochastic neurons for conditional computation', *CoRR*, vol. abs/1308.3432, 2013.

[176] J. A. Hoeting, D. Madigan, A. E. Raftery and C. T. Volinsky, 'Bayesian model averaging: A tutorial', *Statistical science*, pp. 382–401, 1999.

[177] T. Jaakkola, D. Sontag, A. Globerson and M. Meila, 'Learning Bayesian network structure using LP relaxations', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 358–365.

[178] H. Kang, C. D. Yoo and Y. Na, 'Maximum margin learning of t-SPNs for cell classification with filtered input', *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 1, pp. 130–139, 2016.

[179] M. Koivisto and K. Sood, 'Exact Bayesian structure discovery in Bayesian networks', *Journal of Machine Learning Research*, vol. 5, pp. 549–573, 2004.

[180] V. Mansinghka, P. Shafto, E. Jonas, C. Petschulat, M. Gasner and J. B. Tenenbaum, 'Crosscat: A fully Bayesian nonparametric method for analyzing heterogeneous, high dimensional data', *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 4760–4808, 2016.

[181] D. K. Duvenaud, J. R. Lloyd, R. B. Grosse, J. B. Tenenbaum and Z. Ghahramani, 'Structure discovery in nonparametric regression through compositional kernel search', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, pp. 1166–1174.

[182] N. Friedman, D. Geiger and M. Goldszmidt, 'Bayesian network classifiers', *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[183] D. Schabus, M. Skowron and M. **Trapp**, 'One million posts: A data set of german online discussions', in *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR*, 2017, pp. 1241–1244.

[184] M. **Trapp**, M. Skowron and D. Schabus, 'Retrieving compositional documents using position-sensitive word mover's distance', in *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval ICTIR*, 2017, pp. 233–236.

[185] M. Skowron, M. **Trapp**, S. Payr and R. Trappl, 'Automatic identification of character types from film dialogs', *Applied Artificial Intelligence*, vol. 30, no. 10, pp. 942–973, 2016.

[186] M. **Trapp**, R. Peharz, F. Pernkopf and C. E. Rasmussen, 'Deep structured mixtures of gaussian processes', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[187] M. **Trapp**, 'BNP.jl: Bayesian nonparametrics in Julia', in *proceedings of BNP workshop at NeurIPS*, 2015.

List of Publications

[10] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. **Trapp**, X. Shao, K. Kersting and Z. Ghahramani, 'Random sum-product networks: A simple and effective approach to probabilistic deep learning', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[13] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. **Trapp**, G. V. den Broeck, K. Kersting and Z. Ghahramani, 'Einsum networks: Fast and scalable learning of tractable probabilistic circuits', in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[72] M. **Trapp**, R. Peharz and F. Pernkopf, 'Optimisation of overparametrized sum-product networks', in *proceedings of TPM workshop at ICML*, 2019.

[95] M. **Trapp**, T. Madl, R. Peharz, F. Pernkopf and R. Trappl, 'Safe semi-supervised learning of sum-product networks', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[99] M. **Trapp**, R. Peharz, M. Skowron, T. Madl, F. Pernkopf and R. Trappl, 'Structure inference in sum-product networks using infinite sum-product trees', in *Proceedings of BNP workshop at NeurIPS*, 2016.

[157] M. **Trapp**, R. Peharz, H. Ge, F. Pernkopf and Z. Ghahramani, 'Bayesian learning of sum-product networks', in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 6344–6355.

[183] D. Schabus, M. Skowron and M. **Trapp**, 'One million posts: A data set of german online discussions', in *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR*, 2017, pp. 1241–1244.

[184] M. **Trapp**, M. Skowron and D. Schabus, 'Retrieving compositional documents using position-sensitive word mover's distance', in *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval ICTIR*, 2017, pp. 233–236.

[185] M. Skowron, M. **Trapp**, S. Payr and R. Trappl, 'Automatic identification of character types from film dialogs', *Applied Artificial Intelligence*, vol. 30, no. 10, pp. 942–973, 2016.

[186] M. **Trapp**, R. Peharz, F. Pernkopf and C. E. Rasmussen, 'Deep structured mixtures of gaussian processes', in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[187] M. **Trapp**, 'BNP.jl: Bayesian nonparametrics in Julia', in *proceedings of BNP workshop at NeurIPS*, 2015.

Index