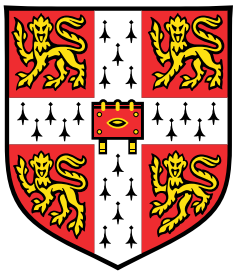
Sum-Product Copulas



Ramona-Gabriela Comănescu

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

St. Catharine's College August 2019

Declaration

I, Ramona-Gabriela Comănescu of St. Catharine's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Word count: 11,481.

Existing software:
The software is implemented in Python and PyTorch. Apart from standard libraries, the following components are reused:

- RegionGraph.py follows exactly the implementation of a random graph as in Peharz et al. [1], provided by Robert Peharz.
- SumNodes.py and ProductNodes.py are adapted versions based on the implementation provided by Wolfgang Roth, used for implementing a regular SPN.

Ramona-Gabriela Comănescu
August 2019

Acknowledgements

I would like to acknowledge my supervisor, Dr. Robert Peharz for proposing this stimulating project, sharing his knowledge on Sum-Product Networks, answering my questions with great enthusiasm, guiding me along the way and providing constructive feedback.

I would also like to acknowledge my second supervisor, Dr. José Miguel Hernández-Lobato for sharing his knowledge on density estimation and copulas.

Thank you to Wolfgang Roth for providing an initial SPN implementation. Many thanks to Francisco Vargas for extensive talk on optimization methods and knowledge on copulas. Thank you to Francisco Vargas, Andreea Cucu and Kamen Brestnichki for careful proofreading and helpful suggestions.

Finally, I would like to thank my parents for their continuous support.

Abstract

Sum-Product Networks are a promising deep probabilistic model with tractable inference. Copulas are a powerful and flexible framework widely used in finance, from which the machine learning community can benefit. Most known parametric copulas are hard to extend apart from the bivariate case. We introduce a sum-product network that satisfies the copula constraints as a way to construct flexible and high-dimensional copulas, which we call Sum-Product Copulas. Sum-Product Copulas achieve competitive results on a range of general-purpose real-valued density estimation tasks.

Contents

Nomenclature

CDF Cumulative distribution function

GCS Synthetic Dataset generated using Gaussian Copulas

GM Graphical Model

GMM Gaussian Mixture Model

KDE Kernel Density Estimation

PDF Probability density function

RV Random variable

SP-CL Sum-Product Network with Copula Leaves

SPC Sum-Product Copula

SPN Sum-Product Network

Chapter 1

Introduction

1.1 Motivation

Multivariate real-valued distributions occur in a variety of fields, such as computational biology and neuroscience, finance and climatology. This ubiquity of real-valued data makes the task of density estimation a central goal of machine learning. More generally, being able to build a model for the joint density $p(\mathbf{x})$ of a probability distribution enables a wide range of tasks such as inference, prediction, data generation and data completion. For instance, density estimation can be used for learning suitable priors from unlabelled datasets, for use in Bayesian inference (Zoran and Weiss [2]), or for learning effective proposals for importance sampling (Papamakarios and Murray [3]).

A challenge in modelling probability distributions is constructing models that are flexible enough to represent high-dimensional distributions (which can be skewed, heavy tailed or multi-modal), while also having tractable density functions and learning algorithms. Probabilistic generative models (generative models are models that specify a full probability density over all variables [4]) have a range of applications (image inpainting, synthesis, compression, unsupervised feature learning, etc.). However, evaluating the likelihood of many of the models can be computationally intractable.

In this thesis, we propose a novel approach of combining two techniques, namely copulas and sum-product networks (SPN [5]). In this way, we address the challenges exposed above: copulas give us flexibility and sum-product networks allow constructing more powerful copulas, while still maintaining tractable inference. We refer to our model as Sum-Product Copula (SPC).

A copula function is a joint distribution defined on the unit hypercube, with uniform marginals. It is a general framework that allows decomposing the problem of joint multivariate distribution estimation into estimating marginal distributions (which is sufficiently

easy) and estimating the dependencies between random variables (RVs). This allows us to separate the choice of marginals from that of the dependence structure expressed by the copula. Despite their generality and flexibility, constructing high-dimensional copulas is difficult and most of the times only the bivariate case is employed. We would like to take advantage of their desirable properties by combining them with another model. For this reason, we propose leveraging SPNs, in order to construct powerful copulas.

SPNs are a promising probabilistic model, offering the advantage of tractable and exact inference, as well as the ability to quickly perform marginalisation and answer inference queries. The potentially deep structure of SPNs allows them to capture complex interactions between variables, which makes them a great candidate for representing a copula distribution.

We modify the density of an SPN such that it learns to represent a distribution that satisfies the constraints of a copula (uniform marginals). The tractable inference and fast marginalisation properties will allow us to easily obtain the terms needed to construct a copula. In this manner, we obtain a new way of building high-dimensional copulas, giving us a powerful framework for density estimation.

A regular SPN is implemented. To simplify learning, we implement Random SPNs (Peharz et al. [1]), which allow training SPNs in a classical deep learning manner. During learning, we enforce the constraint of uniform marginals, using various constrained and unconstrained optimization methods. We then train SPCs with different structures and compare them with their corresponding SPN, as well as other density estimation models from literature. We evaluate our models on a variety of real-valued density estimation tasks, including highly-dimensional images.

1.2 Outline

- In Chapter 2, we introduce essential background material on SPNs (Section 2.1) and Copulas (Section 2.2), where we explain the main concepts, notation and necessary building blocks for our approach. We include related work at the intersection of traditional statistics and machine learning, explaining how a traditional financial modelling tool can benefit the field. We mention several research avenues aimed at building high dimensional copulas.

- Chapter 3 introduces and proves a novel result, proving that under a set of circumstances, an SPN with copula input distributions builds a valid highly dimensional copula. This is a useful exercise for better understanding of the two concepts (SPNs and copulas). We find that this model offers a promising research direction and that small extensions on it could yield competitive results.

- Chapter 4 explains the main method of this thesis, based on modifying an SPN with any type of input distributions over continuous data and introduces a framework for satisfying the necessary copula constraints. We include a review of constrained optimization and root-finding methods and list our final algorithm for building an SPC. This section also shows how to compute the derivatives of implicit functions that our method introduces, since automatic differentiation can not be used.
- Chapter 5 presents presents the practical implementation details of our approach, as well as choice of evaluation tasks and methodology. We follow by showing the experimental results of our approach, contrasting our method with others. Moreover, we present insights into the behaviour of the methods during training.
- Chapter 6 summarizes our findings and formulates conclusions.

Chapter 2

Background and related work

In this chapter we introduce essential background material on Sum-Product Networks (Section 2.1) and Copulas (Section 2.2). We explain how to learn the structure and parameters of an SPN, as well as how to perform inference. We introduce key theorems regarding copulas and present related work at the intersection of machine learning models and copulas. The formal notation related to SPNs and copulas will be introduced here and used accordingly for the rest of the thesis.

2.1 Sum-Product Networks (SPNs)

Sum-Product Networks are a promising deep probabilistic architecture introduced by Poon and Domingos [5]. Their advantages include: **tractable inference** while still being expressive, and **deep** architecture while maintaining full and clear **probabilistic** semantics. In this chapter we introduce the concept of SPNs and outline the main algorithms that will be used to build towards our final copula approach.

2.1.1 Origin of SPNs

Poon and Domingos [5] introduce deep sum-product networks as a method to compute partition functions of tractable graphical models. The concept can be traced back to Darwiche [6], under the name of Arithmetic Circuits. According to this approach, a Bayesian Network (a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph) can be compiled into a multivariate polynomial. The Bayesian Network is expressed as a polynomial of **indicator variables**, leading to the concept of a **network polynomial** (a multilinear function of indicator variables). For a binary

RV X_i and its negation $\bar{X}_i$, we define the indicator function $[.]$, which takes value 1 when its argument is true and 0 otherwise.

For an unnormalized probability distribution $\Phi(x) \geq 0$, the network polynomial is $\sum_x \Phi(x)\Pi(x)$, where $\Pi(x)$ is the product of indicators that have value 1 in state x. For example, the network polynomial of a Bernoulli distribution with parameter p is $p[X_i] + (1-p)[\bar{X}_i]$. To compute the unnormalized probability of evidence e, where e is a partial instantiation of X, we compute e as the network polynomial where all indicators compatible with 1 are set to 1 and the rest are 0. This allows us to also compute the partition function, where all indicators are set to 1. Thus, network polynomials can easily be used for computing $P(e)$ and performing inference tasks. However, the cost of computing $P(e)$ is linear in the size of the network polynomial, which is exponential in the number of variables.

At this is point, Poon and Domingos [5] propose using **sum-product networks** as a way to represent and evaluate network polynomials in polynomial space and time. The Boolean variable case can be extended to multivalued discrete variables (by replacing Boolean indicators with indicators for the variable's possible values), as well as continuous variables (by viewing them as multinomial variables with an infinite number of values).

Consequently, Poon and Domingos [5] show that under certain conditions, an SPN computes the probability of evidence in time linear in its size. For this to be true, the SPN needs to be valid. We will explore what this entails in the following subsections.

2.1.2 Formal Definitions and Notation

We will formally define an SPN and its properties. From now on, we will be considering generalized SPNs, defined over arbitrary distributions (rather than indicator variables), as in Peharz et al. [7]. We base our presentation on the notation and definitions introduced in Peharz et al. [7] and Peharz et al. [8].

RVs are denoted by uppercase letters, X, Y. Sets of RVs are denoted by boldface uppercase letters $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$. The set of values a RV X can take is val(X). For a set of RVs $\mathbf{X} = \{X_1, \ldots, X_N\}$ we define the Cartesian product $\text{val}(\mathbf{X}) = \times_{n=1}^{N} \text{val}(X_n)$ and we use $\mathbf{x}$ for elements of $\text{val}(\mathbf{X})$. When $\mathbf{Y} \subseteq \mathbf{X}$ and $X \in \mathbf{X}$, we use $\mathbf{x}[\mathbf{Y}]$ and $\mathbf{x}[X]$ for the projections of $\mathbf{x}$ onto $\mathbf{Y}$ and X, respectively.

The elements of $\text{val}(\mathbf{X})$ can be interpreted as *complete evidence*, assigning a fixed value for each RV in $\mathbf{X}$.

Partial evidence is represented as a subset $\mathcal{X} \subseteq \text{val}(X)$, which is an element of the sigma-algebra $\mathcal{A}_X$ induced by RV X. We use $\mathcal{A}_X = \{\mathcal{X} \subseteq \mathcal{B} | \mathcal{X} \in \text{val}(X)\}$, where $\mathcal{B}$ represents the Borel-set over $\mathbb{R}$. For discrete RV, this choice gives the power-set $\mathcal{A}_X = 2^{\text{val}(X)}$.

For set of continuous RVs, we use $\mathcal{H}_{\mathbf{X}} := \left\{ \times_{n=1}^{N} \mathcal{X}_n | \mathcal{X}_n \in \mathcal{A}_{X_n} \right\}$ to represent partial evidence about $\mathbf{X}$. We denote the elements of $\mathcal{H}_{\mathbf{X}}$ as $\boldsymbol{\mathcal{X}}$. For $\mathbf{Y} \subseteq \mathbf{X}$, $X \in \mathbf{X}$ and $\boldsymbol{\mathcal{X}} \in \mathcal{H}_{\mathbf{X}}$, we define $\boldsymbol{\mathcal{X}}[\mathbf{Y}] := \{\mathbf{x}[\mathbf{Y}] | \mathbf{x} \in \boldsymbol{\mathcal{X}}\}$ and $\boldsymbol{\mathcal{X}}[Y] = \{\mathbf{x}[Y] | \mathbf{x} \in \boldsymbol{\mathcal{X}}\}$. The definition of partial evidence will become important when expressing **marginalisation** for a given RV. Furthermore, we use e to symbolise any combination of complete and partial evidence.

For a given node N in some directed graph $\mathcal{G}$, let $\mathrm{Ch}(N)$ be the set of children of N and $\mathrm{Pa}(N)$ the set of parents of N.

For generalized SPNs, we follow the definition formulated in Peharz et al. [9].

Definition 2.1.1. (Sum-Product Network). A sum-product network $\mathcal{S}$ defined over a set of RVs $\mathbf{X}$ is a tuple $(\mathcal{G}, w)$, where $\mathcal{G}$ is a directed acyclic graph which is both rooted and connected (directed tree) and $\mathbf{w}$ is a set of non-negative parameters. There are three types of nodes in the graph: distributions, sums and products. All leaves are distributions, while all internal nodes are either sums or products. A distribution node (input distribution) $D_{\mathbf{Y}} : \mathrm{val}(\mathbf{Y}) \mapsto [0, \infty]$ is a distribution function defined over a subset of RVs $\mathbf{Y} \subseteq \mathbf{X}$. The value of a sum node S is the weighted sum of the values of its children, $S = \sum_{C \in \mathrm{Ch}(S)} w_{S,C}$, where $w_{S,C}$ is a non-negative weight associated with edge $S \to C$. The value of a product node is the product of the values of its children, $P = \prod_{C \in \mathrm{ch}(P)} C$.

Definition 2.1.2. (Scope). For any node N in an SPN $\mathcal{S}$, the **scope** $\mathrm{sc}(N)$ is the set of variables in N. For a leaf node N where N is a distribution $D_{\mathbf{Y}}$, the scope is defined as the variables that appear in N, $\mathrm{sc}(N) := \mathbf{Y}$. For an internal (sum or product) node, the scope is defined recursively as

$$\mathrm{sc}(N) = \bigcup_{N' \in \mathrm{Ch}(N)} \mathrm{sc}\left(N'\right).$$

The value of an SPN $\mathcal{S}$ is the value computed by its root and it is denoted as $S(\mathbf{x})$. Without loss of generality, the scope of the root is $\mathbf{X}$.

An example SPN is shown in Figure 2.1. Performing inference in an unconstrained SPN is usually intractable. However, we can guarantee efficient inference for *complete* and *decomposable* SPNs.

Definition 2.1.3. (Completeness [7]). A sum-product network is **complete** if and only if all children of the same sum node have the same scope. That is, for all sum nodes it holds that:

$$\mathrm{sc}\left(C'\right) = \mathrm{sc}\left(C''\right), \forall C', C'' \in \mathrm{Ch}(S).$$

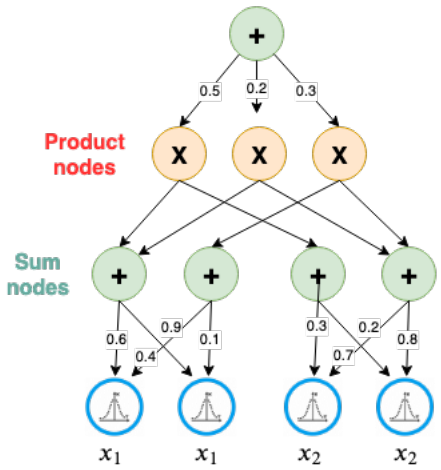


Figure 2.1 Example sum-product network with Gaussian input distributions

Definition 2.1.4. (Decomposability [5]). A sum-product network is **decomposable** if and only if no variable appears in more than one child of a product node. That is:

$$\forall C', C'' \in \mathrm{Ch}(P), C' \neq C'' \rightarrow sc\left(C'\right) \cap sc\left(C''\right) = \varnothing.$$

Certain operations are only efficient when applied to *selective* (Peharz et al. [9]) SPNs.

Definition 2.1.5. (Selective SPN). A sum node S is *selective* if for all choices of sum-weights $\mathbf{w}$ and all possible inputs $\mathbf{x}$ it holds that at most one child of S is non-zero.

2.1.3 SPN Operations

[8] showed that under certain conditions, SPNs can perform tractable exact inference. In fact, many of the inference scenarios can be expressed as a single forward or backward passes. The value of an SPN corresponds to the value of its root, evaluated after a forward (bottom-up) pass. In the rest of this subsection, we will work with normalized SPNs, where the weights of sum nodes sum up to 1. Therefore, evaluating the SPN will give normalised distributions (Peharz et al. [8]). For the rest of this thesis, we only consider complete and decomposable SPNs.

For generalized SPNs that are complete and decomposable, efficient algorithms (Peharz et al. [8]) exist for evaluating:

- Probability of a sample

- Marginalisation
- Probability of partial observations
- Maximum a Posteriori state (for selective SPNs)

A key idea for continuous variables is that during inference, the value of an integral node over a marginalised out variable is 1. Moreover, we can pull integrals over all sums and products down to input distributions. Thus, operations that require integrating out variables can be performed easily, by evaluating integrals and the usual sums and products. We briefly outline algorithms that can be performed by SPNs, as they will be useful in deriving our proposed model.

Probability of a sample

The probability of a sample (complete observation) $\mathcal{S}(\mathbf{x})$ for $\mathbf{x} \in \text{val}(\mathbf{X})$ can easily be computed by evaluating the SPN in a upwards pass, by first evaluating the values at the given inputs and propagating to the root. Since we are working with valid SPNs, we are guaranteed that this evaluation returns the true probability.

Marginalisation

In marginalisation, we compute:

$$\mathcal{S}(\boldsymbol{\mathcal{X}}) = \int_{x_1 \in \boldsymbol{\mathcal{X}}_1} \cdots \int_{x_N \in \boldsymbol{\mathcal{X}}_N} \mathcal{S}(x_1, \ldots, x_N)\, dx_1 \ldots dx_N.$$

To marginalise out variables, we simply perform marginalisation on the input distribution leaves and propagate up in the usual way (Figure 2.2). As stated in Peharz et al. [8], summing out marginalised variables translates to computing integrals over input distributions in the continuous case. Due to completeness, integrals can be exchanged with sums at sum nodes. Similarly, decomposability allows us to interchange integrals with products at product nodes. This means that on the upwards pass, we simply perform marginalisation over the input distributions over their scopes and proceed to evaluate SPNs in the normal way.

Probability of Modified Evidence

We aim to evaluate the probability of some modified evidence, that is $\mathcal{S}(X, \boldsymbol{\mathcal{X}}[\mathbf{X} \backslash X])$. for $X \in \mathbf{X}$.

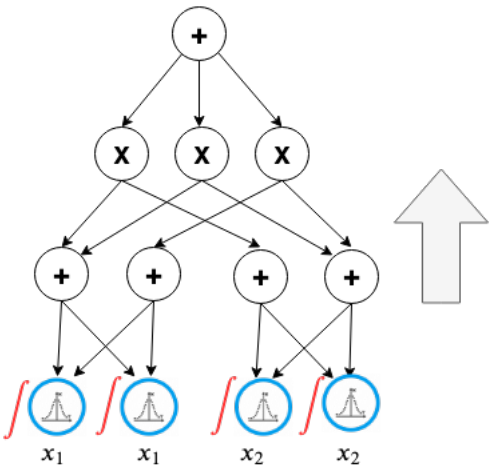


Figure 2.2 Marginalisation in an SPN

The most straightforward approach is to compute the marginals of variables by differentiation (Darwiche [6]). If node i is a product node and Pa_i are its parents, and assuming alternating sum and product layers, we have:

$$\frac{\partial S(\mathbf{x})}{\partial S_i(\mathbf{x})} = \sum_{k \in Pa_i} \left[\frac{\partial S(\mathbf{x})}{\partial S_k(\mathbf{x})} w_{ki} \right].$$

For a sum node, we have:

$$\frac{\partial S(\mathbf{x})}{\partial S_i(\mathbf{x})} = \sum_{k \in Pa_i} \left[\frac{\partial S(\mathbf{x})}{\partial S_k(\mathbf{x})} \prod_{l \in Ch_{-i}(k)} S_l(\mathbf{x}) \right],$$

where $Ch_{-i}(k)$ are the children of node k excluding node i.

We will be more interested in an extension to the differential approach as given in Peharz et al. [8], since it allows evaluating modified evidence simultaneously for all $X \in \mathbf{X}$. Let $\mathbf{D}^X$ be the set of distributions containing X in their scope, with an arbitrary fixed ordering $\mathbf{D}^X = \left\{ D^{X,1}, \ldots, D^{X,|\mathbf{D}^X|} \right\}$. Algorithm 1 summarizes the steps used for infering marginal posteriors. We draw the reader's attention to this algorithm. For our purposes, this method will allow us to compute univariate marginals fast. We view marginalisation as a special case of modified evidence: specifically, we set $\mathbf{X} = \emptyset$. Thus, the result of the algorithm will be all single dimensional marginals, computed simultaneously. Informally, we simply need to collect the derivatives with respect to all input distributions which have a given RV in

Algorithm 1: Inferring marginal posteriors

1. Evaluate input distributions for evidence $\boldsymbol{\mathcal{X}}$.
2. Evaluate all sums and products in an upwards pass.
3. For all nodes N, compute $\frac{\partial S}{\partial N}$, using backpropagation.
4. $\forall X, k$, let $f_k(X) = D^{X,k}(X, \boldsymbol{\mathcal{X}}[\mathrm{sc}(D^{X,k}) \backslash X])$
5. $\forall X : \mathcal{S}(X, \boldsymbol{\mathcal{X}}[\mathbf{X} \backslash X]) \leftarrow \sum_{k=1}^{|\mathbf{D}^X|} \frac{\partial S}{\partial D^{X,k}} f^k(X)$.

their scope (as described in step 5). This approach gives a nice interpretation, as we can now perform backpropagation in an SPN, while maintaining probabilistic semantics.

Most Probable Explanation (MPE)

Another advantage of SPNs is that they can perform (approximate, but exact for certain SPNs [7]). Most Probable Explanation $\arg\max_{\mathbf{X},\mathbf{Y}} P(\mathbf{X}, \mathbf{Y}|\mathbf{e})$ fast (Poon and Domingos [5]). [7] show that the MPE inference algorithm is actually only correct for selective SPNs. In general, MPE inference is NP-hard.

Sum nodes are replaced by max nodes. Input distributions are replaced by maximizing distributions. The SPN is evaluated in a bottom up pass. On the top down pass, the child of a max node with the highest value is picked. For product nodes, all children are picked. Eventually, a leaf node is reached, returning the most likely value for the query variables.

Being able to compute MPE allows SPNs to perform tasks such as image completion, where they particularly excel. Moreover, MPE inference can be used at learning time as an alternative to marginal inference.

2.1.4 Learning SPNs

Parameter Learning

One approach to learning SPNs is to start with a dense valid arithmetic circuit and learn the weights using Expectation Maximization (EM) or gradient descent, both approaches being detailed in Poon and Domingos [5]. In practice, Poon and Domingos [5] find that replacing marginal inference with MPE inference (called Hard EM) makes it possible to learn deeper SPNs, by avoiding the problem of gradient diffusion. Note that Peharz et al. [8] derive the correct EM updates, as the one proposed in Poon and Domingos [5] is not correct.

A discriminative approach can also be employed, as shown in Gens and Domingos [10], where the SPN is trained to predict a label variable. Both Marginal Inference and MPE inference approaches exist, as in the case of the generative approach.

Structure Learning

In Poon and Domingos [5], structure is learned implicitly by starting with a dense network and pruning it. However, more advanced methods have been researched. When building SPNs, the concepts of *regions* and *region graphs* ([11], [5], [12], [1]) will be useful.

A region $\mathbf{R}$ is defined as any non-empty subset of the set of RVs $\mathbf{X}$. A $K-$*partitions* $\mathcal{P}$ of $\mathbf{R}$ is a collection of K non-empty and non-overlapping subsets $\mathcal{P} = \{\mathbf{R}_1, \ldots, \mathbf{R}_k\}$ of $\mathbf{R}$, whose union is also $\mathbf{R}$. In this thesis, we only consider two-partitions, thus all product nodes have exactly two children. Partition nodes are restricted to being the children of region nodes and vice versa.

Definition 2.1.6. (Region Graph). A region graph $\mathcal{R}$ over $\mathbf{R}$ is a rooted directed acyclic graph consisting of region nodes and partition nodes such that:

(i) X is the *root* region in $\mathcal{R}$ and has no parents

(ii) All other regions have at least one parent

(iii) All children of regions are partitions and all children of partitions are regions

(iv) If $\mathcal{P}$ is a child of $\mathbf{R}$, then $\cup_{\mathbf{R}' \in \mathcal{P}} \mathbf{R}' = \mathbf{R}$

(v) If $\mathbf{R}$ is a child of $\mathcal{P}$, then $\mathbf{R} \in \mathcal{P}$.

The above definition shows how every region graph dictates a hierarchical partition of scope $\mathbf{X}$. Region and partition nodes have scopes just like nodes in an SPN. We denote regions which have no child partitions as *leaf partitions*.

For any given region graph $\mathbf{R}$, a corresponding SPN with S sum nodes (assuming one node for the root region) and I input distributions can be constructed, as suggested in Peharz et al. [1] and shown in Algorithm 2. Informally, sum nodes are introduced for regions and product nodes are introduced for all possible child partitions.

Poon Architecture

Poon and Domingos [5] propose an architecture suited for images, usually referred to as the Poon architecture. This architecture selects all rectangular regions, where the smallest regions correspond to pixels. Each rectangular region is decomposed vertically and horizontally into

Algorithm 2: Constructing an SPN from Region Graph

input : $\mathcal{R}, S, I$
Create empty SPN.
for $\mathbf{R} \in \mathcal{R}$ **do**
  if $\mathbf{R}$ *is a leaf region* **then**
    Introduce I distribution nodes
  else if $\mathbf{R}$ *is the root region* **then**
    Introduce one sum node as root
  else
    Introduce S distribution nodes
  end
end
for $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\} \in \mathcal{R}$ **do**
  Let $\mathbf{N}_\mathbf{R}$ be the nodes for region $\mathbf{R}$ **for** $N_1 \in \mathbf{N}_{\mathbf{R}_1}, N_2 \in \mathbf{N}_{\mathbf{R}_2}$ **do**
    Introduce product node $P = N_1 \times N_2$ Let P be a child for each $N \in \mathbf{N}_{\mathbf{R} \cup \mathbf{R}_2}$
  end
end

two rectangular subregions, in all possible ways (an example is shown in in Figure 2.3). For larger regions, coarse decomposition can also be used, for a less granular resolution.

Clustering on Variables

While intuitive, the Poon architecture includes many long and skinny rectangular regions, which might mean pixels from the end of the regions are grouped together, despite only having weak interactions. Another drawback is that non-rectangular regions are not explained.

Dennis and Ventura [11] introduce clustering on variables as a way to learn the architecture of SPNs. Variable subsets that strongly interact with each other are learnt from data. Several other approaches exist for learning the structure of SPNs. One of the most well known is *LearnSPN* (Gens and Domingos [13]), based on dividing current variables into approximately independent subsets for producing product nodes and clustering for producing sum nodes (usually using the EM algorithm for clustering).

Random and Tensorized SPN (RAT-SPN)

Peharz et al. [1] introduce RAT-SPNs, a way to treat SPNs as classical deep learning models, making it possible to employ tools such as automatic differentiation and Stochastic Gradient Descent (SGD) for learning, while also automatically introducing structure. The advantage

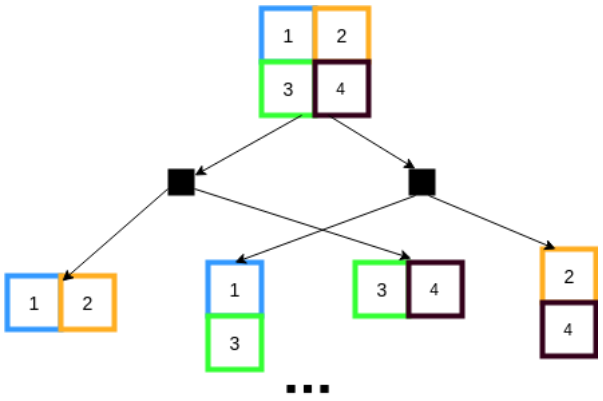


Figure 2.3 Example Poon architecture based on region splitting

of this view is that is makes it trivial to learn SPNs, while still maintaining their generative model interpretation and maintaining well-calibrated uncertainties.

Random Region Graphs are obtained on recursively and randomly dividing the root region into two sub-regions of the same size, down to a given depth **D** and repeating the mechanism **R** times. The procedure for constructing random region graphs, as proposed in Peharz et al. [1], is given in Algorithm 3.

This method of building SPNs maintains the decomposability and completeness properties and also allows building deep SPNs than can be organized in a layer-wise architecture (alternating sums and products).

Training RAT-SPNs

The resulting layers can be tensorized and trained in a manner similar to training a classical Multilayer Perceptron (MLP). The training objective can be set to achieve a trade-off between generative and discriminative, balancing cross-entropy (discriminiative) and negative log-likelihood likelihood (generative).

The input distributions are usually set to be Gaussian or Categorical distributions, with I distributions per leaf node. Additionally, each region has **S** sum nodes. The nodes in a region can be viewed in matrix form, where rows correspond to samples and columns correspond to the number of distributions.

Algorithm 3: Obtaining Random Region Graphs

```
1  procedure RandomRegionGraph(X, D, R)
2      Initialize empty region graph R
3      Insert X in R
4      for r = 1 ... R do
5          SPLIT(R, X, D)
6      end
1  procedure SPLIT(R, R, D)
2      Randomly draw a balanced partition P = {R_1, R_2} of R.
3      Insert R_1 and R_2 in R
4      Insert P in R if D > 1 then
5          if |R_1| > 1 then
6              SPLIT(R, R_1, D − 1)
7          end
8          if |R_2| > 1 then
9              SPLIT(R, R_2, D − 1)
10         end
11     end
```

2.1.5 Applications

SPNs can easily perform inference, representing a joint distribution over a set of RVs. SPNs have achieved competitive results on numerous tasks.

A good test for deep architectures is that of **image completion**, where it is essential to detect deep structure. For this task, the Poon architecture for images can be used. More recently, Deep Convolutional SPNs (Butz et al. [14]) achieved impressive results on this task. There is a class of Convolutional Neural Networks that define valid SPNs. The reconstruction is done by performing MPE to find the most probable values for the missing features.

Additionally, variants of SPNs can successfully be used in domains such as speech processing (as in Peharz et al. [15]), language (Cheng et al. [16] use discriminatively trained SPNs to predict the next word), activity recognition (Amer and Todorovic [17]). Moreover, mixed SPNs can be used in hybrid domains, with Molina et al. [18] proposing ways to simplify identifying the parametric form of leaf variables. Many other applications also exist and research is on-going, proving the versatility of SPNs.

2.1.6 SPNs and Other Models

Comparison to Graphical Models (GMs)

Graphical models are limited in some aspects. Many distributions cannot be represented in a compact manner as a graphical model, a famous example being the uniform distribution over binary vectors with even number of 1s (See the examples in [19]).

Any distribution can be encoded using a shallow exponentially large SPN. Moreover, some distributions can be encoded using a compact deep SPN. Delalleau and Bengio [20] show that the number of units in the shallow network has to grow exponentially, compared to a linear growth in the deep network. To obtain a deep SPN, multiple layers of sum and product nodes alternate.

What is more, exact inference in a GM is exponential in the worst case, with approximate techniques being often used. This is where the advantage of SPNs is clear, as they provide **fast, exact** and **tractable** inference, that is linear in the size of the model for a larger class of problems Hernández-Lobato [19].

SPNs as Mixture Models

The main drawback of Graphical Models is that inference cost can be exponential. More generally, an SPN can be seen as a **Deep Mixture Model** (Peharz et al. [7]). In fact, a mixture model is just a "shallow" SPN, consisting of leaf distributions and a single sum node. By hierarchically composing an SPN from sum and product layers, we can obtain the deep version of a mixture model, while still maintaining the universal approximator properties (Peharz et al. [1]).

2.2 Copulas

Copula modeling is widely used in finance, for predicting the price of financial instruments (University of Oxford [21]). Recently, copulas are becoming more prominent in machine learning, due to their flexibility in modelling continuous data and the fact that they can be successfully used for learning joint distributions. Their key advantage is that they can model the dependence structure between variables while separating this from modelling the marginal distributions of variables.

2.2.1 Formal Definitions and Notation

For the rest of the thesis, we will use $C(\cdot)$ for a copula function and $c(\cdot)$ for a copula density, and $C_\Theta(\cdot)$ and $c_\Theta(\cdot)$ respectively for their parameterized versions. We use Θ to define sets of parameters. As before, we use uppercase X_i to refer to RVs and lowercase x_i for an instantiation of X_i. Boldface letters are used for sets of RVs.

We use D to denote d dimensions. For a set of d real valued RVs $X_1, X_2, \ldots, X_d$, the *joint cumulative distribution* function (CDF) $F : \mathbf{R^d} \to [0,1]$. is defined as:

$$F(x_1, x_2, \ldots, x_d; \theta) = Pr(X_1 \leq x_1, \ldots X_d \leq x_d).$$

We will always use uppercase letters for CDFs. If the *probability density function* (PDF) of F exists, we denote $f : \mathbb{R}^d \to [0, \infty]$.

For a given joint distribution F, a *marginal distribution* can be obtained by marginalising out other dimensions. For a dimension d, we define its marginal as $F_d(x_d)$. The corresponding marginal PDF will be denoted as $f_d(x_d)$. By applying the CDF function to a variable, we obtain uniform variables. We always use the letter u to refer to instantiations of uniformly distributed random variables.

We use u_d to refer to a uniform marginal obtained by applying $F_d(x_d)$ to a RV x_d. We use $F_d^{-1}(u_d)$ for the inverse of the d^{th} dimension's marginal CDF (the inverse CDF can also be referred to as the *quantile* function). When we are presented with a dataset, we use n for its number of samples. The functions defined so far will be updated accordingly to take as parameters RVs $x_{n,d}$ (e.g. $F_d(x_d)$). We use x_d to reduce clutter, but this will be clear from context. As a shorthand, we introduce a new variable $y_d = F_d^{-1}(u_d)$ to refer to the result of applying the quantile function to u_d. We will call the y variables *pseudo-data*.

To differentiate between a general CDF F and some model specific CDFs, we will introduce Ψ and ψ as the joint CDF and PDF respectively that correspond to a particular model (e.g. the density as given by an SPN is ψ_{SPN}), where the model used will be specified. All the quantities define above apply: $\Psi(x_1, \ldots, x_d)$ for a CDF, $\psi(x_1, \ldots, x_d)$ for the corresponding PDF (if it exists), $\Psi_d(x_d)$ and $\psi_d(x_d)$ for single dimensional marginals, $\Psi_d^{-1}(x_d)$ for the quantile function of the d^{th} dimension's marginal CDF..

A *copula function* is a multivariate cumulative distribution function for which the marginal probability distribution of each variable is uniform. Formally:

Definition 2.2.1. (Copula function). Let $U_1, \ldots, U_d$ be real random variables marginally uniformly distributed on $[0,1]$. A copula function $C : [0,1]^d \to [0,1]$ is a joint distribution

$$C(u_1, \ldots, u_d) = Pr(U_1 \leq u_1, U_d \leq u_d).$$

As shown above, C is defined on the unit hypercube.

Sklar's Theorem (Sklar [22]) is key to copula-based density estimation:

Theorem 2.2.1. (Sklar's Theorem). A joint distribution $F(x_1, x_2, \ldots x_d)$ of random variables $X_1, X_2, \ldots, X_d$ can be expressed as a function of the marginal distributions $F_i(x_i)$:

$$F(x_1, \ldots, x_d) = C(F_1(x_1), \ldots, F_d(x_d)). \tag{2.1}$$

Moreover, if each F_i is continuous, then C is unique.

We can also rewrite 2.1 in terms of the univariate marginals:

$$C(u_1, u_2, \ldots, u_d) = F(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d)), \tag{2.2}$$

The most important result from Sklar's theorem tells us that we can write any joint density f as a product of univariate marginal densities f_d and a **copula density**, c, assuming that the copula function C has d^{th} order partial derivatives. We derive the joint density $f(\mathbf{x}) = \frac{\partial^d F(\mathbf{x})}{\partial x_1 \ldots \partial x_d}$ using the chain rule as:

$$f(\mathbf{x}) = \frac{\partial^d C(F_1(x_1), \ldots, F_d(x_d))}{\partial F_1(x_1) \ldots \partial F_d(x_d)} \prod_d f_d(x_d) = c(F_1(x_1), \ldots, F_d(x_d)) \prod_d f_d(x_d) \tag{2.3}$$

$$= c(u_1, \ldots, u_d) \prod_d f_d(x_d) \tag{2.4}$$

The copula density function $c(u_1, u_2, \ldots, u_d)$ will be of central use in our SPC model. Equation 2.3 mathematically illustrates the main result of copulas: they allow us to separate the choice of univariate marginals and that of the dependence structure expressed by the copula function.

2.2.2 Building Multivariate Copula Densities

Copularization

Equation 2.3 can be rearranged as:

$$c(u_1, \ldots, u_d) = \frac{f(x_1, \ldots, x_d)}{\prod_d f_d(x_d)}. \tag{2.5}$$

The distinction to be made here is that while Equation 2.3 uses a copula to estimate joint densities when the marginal and copula densities are known, we can use Equation 2.5 for deriving a copula density from a known multivariate joint density. We name this method

"copularization" and we will use it to derive a copula density corresponding to an SPC model and some estimated marginals.

Parametric Copula Models

There are several parametric 2-Dimensional copulas (Nelsen [23]), such as the Gaussian Copula, Archimedean Copulas (Clayton, Frank, Gumbel, Joe, etc.) and the t- Copula. One of the most commonly used copula is the Gaussian Copula, which can be constructed directly by inverting the formula in Sklar's Theorem:

$$C_{\Sigma}^{Gauss}\left(F(x_1),\ldots,F(x_N)\right)=\Phi_{\Sigma}\left(\Phi^{-1}\left(F\left(x_1\right)\right),\ldots,\Phi^{-1}\left(F\left(x_N\right)\right)\right), \tag{2.6}$$

where Φ is the CDF of standard normal distribution and Φ_{Σ} is the CDF of a zero mean normal distribution with correlation matrix Σ (where $\Sigma_{ij}\in[-1,1]\,\forall i\neq j$ and $\Sigma_{ii}=1$).

2.2.3 Copulas in Machine Learning

Until recently, the copula framework has not been very prominent in the field of machine learning. However, the complementing strength between machine learning and general probabilistic graphical models suggest that the two fields can mutually benefit from using copulas for multivariate modeling.

The expressiveness of parametric copulas is more limited in higher dimensions. The survey paper in Elidan [24] presents some recent copula based constructions in the field of machine learning that are useful for modeling high-dimensional data. The main motivation is based on combining copulas with graphical models, which are less powerful when it comes to modeling real-valued distributions. The introduction of the copula framework is meant do address this, since using graphical models means we diminish the difficulty of constructing high-dimensional copulas. Thus, we can see how the two frameworks will complement each other. All approaches start with first estimating the univariate marginals, which is an easy enough task, followed by the multivariate copula construction. We briefly outline two of the approaches below.

Tree-averaged Copulas

The first work combining ideas from graphical models and copulas belongs to Kirshner [25], based on an ensemble of trees framework, averaging over all possible tree distributions. Tree-averaged copulas build on the idea of tree-structured copulas. Let T be an undirected tree structured graph (no cycles) and $\mathcal{E}$ the set of edges in T that connect two vertices. The

authors show that

$$f_{\mathbf{X}}(\mathbf{x}) = \left[\prod_i f_i(x_i)\right] \prod_{(i,j)\in\mathcal{E}} \frac{f_{ij}(x_i,x_j)}{f_i(x_i) f_j(x_j)}.$$

Thus, the decomposition of joint copulas follows as:

$$c_T(\cdot) = \frac{f(\mathbf{x})}{\prod_i f_i(x_i)} = \prod_{(i,j)\in\mathcal{E}} \frac{f_{ij}(x_i,x_j)}{f_i(x_i) f_j(x_j)} = \prod_{(i,j)\in\mathcal{E}} c_{ij}\left(F_i(x_i), F_j(x_j)\right),$$

where c_T is used to denote the copula density corresponding to structure T and c_{ij} is used for the copula corresponding to the edge (i, j). In Kirshner [25] work, suggests constructing a mixture of all copulas trees model, instead of relying on only bivariate estimation, as in the case of tree-structured copulas.

Copula Bayesian Network

Elidan [26] relies on a copula based reparameterization of conditional densities and a directed acyclic graph that encodes independencies as building blocks. For a conditional density function $f(x|\mathbf{y}), \mathbf{y} = \{y_1, y_2, \ldots, y_k\}$, there exists a copula density $(c(F(x), F_1(y_1), \ldots, F_k(y_k))$, such that:

$$f(x|\mathbf{y}) = R_c\left(F(x), F_1(y_1), \ldots, F_K(y_K)\right) f_X(x),$$

where R_c is the copula ratio

$$R_c\left(F(x), F_1(y_1), \ldots, F_K(y_K)\right) \equiv \frac{c\left(F(x), F_1(y_1), \ldots, F_K(y_K)\right)}{\frac{\partial^K C(1, F_1(y_1), \ldots, F_K(y_K))}{\partial F_1(y_1) \ldots F_K(y_K)}}.$$

As it can be seen in the previous approaches, the copula factorization of a joint is a key starting-point to building new copulas, an idea which we will also exploit.

Copulas and Mixture Models

Perhaps the closest to our proposed approach are works on combining Mixture Models and copulas, since we will be working with the deep version of GMMs. We present two different approaches.

Mixture of Copulas

A mixture of copulas (Arakelian and Karlis [27], Nelsen [28]) is a straightforward approach of combining Gaussian copula distributions using a weighted sum. This approach has the advantage that it can capture heavy tails (as opposed to normal GMMs). We also empirically show how these can cluster a highly multimodal distributions with a much smaller number of parameters than GMMs.

$$c(u_1,..,u_d|\pi,\Theta) = \sum_{k=1}^{K} \pi_k c_k(u_1,..,u_d;\Theta_k), \tag{2.7}$$

where $c_k(u_1,..,u_d;\Theta_k)$ is a regular Gaussian or Archimedean copula. The key take-away from this method is that a mixture of copulas also defines a copula. Thus, this approach can easily be used in practice by implementing a weighted sum of copulas.

Gaussian Mixture Copula Models

Despite the similar name, this method, proposed by Tewari et al. [29], takes a different approach. It is based on Equation 2.5, which we previously emphasised as a simple way of obtaining a copula distribution, where an expression for the joint distribution and for the marginals is known. A GMM is used for expressing joint distributions. Rewriting Equation 2.5 in terms of a GMM, we have:

$$c_{gmc}(u_1,u_2,\dots u_d;\Theta) = \frac{\psi(y_1,y_2,\dots y_d;\Theta)}{\prod_{j=1}^{d}\psi_j(y_j)},$$

where we have adhered to our previously defined notation i.e. $y = (y_1,y_2,\dots y_d)$ are the inverse distribution values with $y_j = \Psi^{-1}(u_j)$, ψ_j and Ψ_j^{-1} denote the marginal density and the inverse cumulative distribution of the GMM along the j^{th} dimension. A modified EM version is proposed, where the inverse distribution values y_j are computed at each step using numerical methods, making sure the copula constraints are satisfied.

This model is able to model distributions with multiple non-Gaussian modes, due to a higher flexibility than Gaussian Mixture Models, since copulas decouple the estimation of marginal distributions from the dependency structure. Synthetic experiments and image segmentation applications prove the effectiveness of this approach.

2.2.4 Transforming Data into Copula Space

Since a copula distribution is defined over uniformly distributed variables, we need to transform our inputs into copula space. Thus, given dimension d and a random variable x_d, we apply $u_d = F_d(x_d)$ to obtain u_d, where the marginal CDFs F_d can be learnt empirically. Additionally, the univariate marginal densities $f_d(x_d)$ are also required by Equation 4.1. These can be obtained by standard univariate density estimation approaches.

Empirical CDF

Let $x_1, x_2, \ldots, x_n$ be a univariate independent and identically distributed sample drawn from some distribution with an unknown density f and CDF F.

An empirical CDF is a non-parametric estimator of the underlying CDF of a random variable. It requires first ordering the data. The empirical CDF of z (element in the new ordering of x) is computed as:

$$\widehat{F}_n(z) = \frac{\text{number of elements in the sample } \leq z}{n} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{x_i \leq z},$$

where n is the total number of samples and $\mathbb{1}$ is the indicator function.

Kernel Density Estimation

Kernel Density Estimation (KDE) is a non-parametric approach to estimate the probability density function of a random variable. The kernel density estimator of f is defined as:

$$\widehat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$

where K is the kernel (a non-negative function) and $h > 0$ is a smoothing parameter called the bandwidth. Typical kernel functions are often used, such as uniform, triangular, biweight, triweight, normal, and others. We use the normal kernel, as in Tewari et al. [29]. The bandwidth selection method used was Scott's normal reference rule (Scott [30]), although other rules are also possible, including choosing the bandwidth based on cross-validation.

2.2.5 Generating Correlated Data

We will exemplify some of the steps in our pipeline aided by a 2-D dataset that was synthetically generated. A good way to sample dependent random variables is to use copulas, as in Nelsen [23]:

Algorithm 4: Simulating a Gaussian copula

1. Let $\mathbf{\Sigma}$ be a correlation matrix.
2. Compute the Cholesky decomposition $\mathbf{A}$ of $\mathbf{\Sigma}$, so that $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^\top$.
3. Generate a vector $\mathbf{Z}$ of independent standard normal variables.
4. Set $\mathbf{X} = \mathbf{A}^\top \mathbf{Z}$.
5. Return $\mathbf{U} = (\Psi(X_1), \ldots, \Psi(X_d))$, where Ψ is the standard univariate normal CDF .

We can use the above algorithm to generate any random vector $Y = (Y_1, \ldots, Y_d)$ with arbitrary marginals and Gaussian copula, provided we can apply an inverse transform $Y_i = F_i^{-1}(U_i)$.

Chapter 3

SPN with copula leaves (SP-CL)

Before explaining our modfied SPN Copula approach, we start by introducing a simpler alternative: an SPN of copulas, that is, an SPN where input distributions represent copula distributions. We call this model an SP-CL. We introduce a new theoretical result, proving how the hierarchical composition given by composing copula leaves in an SPN fashion results in a copula. This idea is an extension to the Mixture of Copula's Model presented in Arakelian and Karlis [27] (see Section 2.2.3). The advantage of our approach is that in comparison to a GMM, an SPN can also use product nodes, for a more flexible representation.

3.1 Formal Description

Definition 3.1.1. (SP-CL). A sum-product of copulas $\mathcal{C} = (\mathcal{G}, \mathbf{w})$ over continuous RVs $\mathbf{X} = \{X_1, \ldots, X_N\}$ is a complete and decomposable sum-product network $\mathcal{S} = (\mathcal{G}, \mathbf{w})$ whose leaf (distribution) nodes $\mathrm{D}_{\mathbf{Y}}$, $\mathbf{Y} \subseteq \mathbf{X}$ are copulas $\mathrm{C}(\mathbf{U}_{\mathbf{Y}})$.

Theorem 3.1.1. A **complete** and **decomposable** sum-product network $\mathcal{C} = (\mathcal{G}, \mathbf{w})$ with copulas for distribution nodes, is itself a **copula**.

Proof. In order to prove this, we will require Lemmas 3.1.1 and 3.1.2.

Lemma 3.1.1. For a **complete** SPN with copula leaves, any of its sum nodes S with copula children $Ch(S)$, S is itself a copula.

Proof. Since the SPN is complete, all children of the same sum node have the same scope. Thus, the above lemma can be restated as: Copulas are closed under convex combinations under the same scope. To prove this, consider the j^{th} marginal of S, S_{Y_j}, where $\mathbf{Y} = \mathrm{sc}(S)$

and $Y_j \in \mathbf{Y}$:

$$\begin{aligned}
\mathrm{S} &= \sum_{i=1}^{n} w_i C\left(\mathbf{U}_{\mathrm{sc}(S)}\right) \quad \text{s.t } w_i \geq 0, \sum_{i=1}^{n} w_i = 1 \\
\mathrm{S}_{Y_j} &= \sum_{i=1}^{n} w_i \int_{\mathbf{U}_{\mathbf{Y}\setminus Y_j}} C\left(\mathbf{U}_{\mathrm{sc}(S)}\right) d\mathbf{U}_{Y\setminus Y_j} \\
\mathrm{S}_{Y_j} &= \sum_{i=1}^{n} w_i \mathrm{Uniform}_{[0,1]}(u_j) \\
\mathrm{S}_{Y_j} &= \mathrm{Uniform}_{[0,1]}(u_j). \qquad (3.1)
\end{aligned}$$

We have shown that the marginals of S follow a Uniform$[0,1]$ distribution and thus S is itself a copula. □

Lemma 3.1.2. For a **decomposable** SPN with copula leaves and any of its product nodes P with copula children $Ch(P)$, P is itself a copula.

Proof. Since the SPN is decomposable, we know that no variable appears in more that one child of a product node. Thus, the above lemma can be restated as: the product of two copulas $P = C_i(\mathbf{U}_\mathbf{Y})C_k(\mathbf{U}_\mathbf{W})$ with non overlapping scopes (i.e $\mathbf{Y} \cap \mathbf{W} = \emptyset$) is itself a copula. W.l.o.g. we obtain the j^{th} marginal of P, P_{Y_j}, where $\mathbf{Y} = \mathrm{sc}(P)$ and $Y_j \in \mathbf{Y}$:

$$\begin{aligned}
\mathrm{P} &= C_i(\mathbf{U}_\mathbf{Y})C_k(\mathbf{U}_\mathbf{W}) \\
\mathrm{P}_{Y_j} &= \int_{\mathbf{U}_{\mathbf{Y}\setminus Y_j \cup \mathbf{W}}} C_i(\mathbf{U}_\mathbf{Y})C_k(\mathbf{U}_\mathbf{W}) d\mathbf{U}_{\mathbf{Y}\setminus Y_j \cup \mathbf{W}} \\
\mathrm{P}_{Y_j} &= \int_{\mathbf{U}_{\mathbf{Y}\setminus Y_j}} C_i(\mathbf{U}_\mathbf{Y}) d\mathbf{U}_{\mathbf{Y}\setminus Y_j} \int_{\mathbf{U}_\mathbf{W}} C_k(\mathbf{U}_\mathbf{W}) d\mathbf{U}_\mathbf{W} \\
\mathrm{P}_{Y_j} &= \int_{\mathbf{U}_{\mathbf{Y}\setminus Y_j}} C_i(\mathbf{U}_\mathbf{Y}) d\mathbf{U}_{\mathbf{Y}\setminus Y_j} \\
\mathrm{P}_{Y_j} &= \mathrm{Uniform}_{[0,1]}(u_j). \qquad (3.2)
\end{aligned}$$

As in the sum nodes case, we have shown that the marginals of P follow a Uniform$[0,1]$ distribution and thus P is itself a copula. □

By Lemmas 3.1.1 and 3.1.2 we have shown that any node $N \in \mathcal{C}$ will itself be a copula if all distribution nodes are copulas, which follows by induction over the topological order of the SPN nodes. □

3.2 Implementation

An SPN with copula leaves can be implemented as a regular SPN. All we need to determine is the type of copulas to be used as leaves. Any of the known parametric copulas listed in Section 2.2.2 can be used. In our experiments, we have used Gaussian copula leaves with a uniform correlation matrix, due to its simplicity in implementation. A uniform correlation matrix is a correlation matrix where all the off diagonals have the same value $\Sigma_{ij} = \rho$. Having a uniform correlation matrix comes with limited representational power and using a full correlation matrix can improve on this, especially for high dimensional domains.

We define $\mathbf{z}(\mathbf{u}) = (\Phi^{-1}(u_1), ..., \Phi^{-1}(u_d))^\top$, Φ as the CDF of a standard normal, $\mathbb{I}_d$ the d dimensional identiy matrix and $\mathbf{1}_d$ as a d dimensional vector of ones. The uniform Gaussian copula leaves are implemented using the following PDF:

$$c(u_1, ..., u_d; \rho) = (1-\rho)^{\frac{1-d}{2}} (1+(d-1)\rho)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathbf{z}(\mathbf{u})^\top \left(\left(\rho \mathbf{1}_d \mathbf{1}_d^\top + (1-\rho)\mathbb{I}_d\right)^{-1} - \mathbb{I}_d\right) \mathbf{z}(\mathbf{u})\right),$$

which can be simplified using the Sherman-Morrison formula (this makes evaluating the leaves $\mathcal{O}(d)$).

$$c(u_1, ..., u_d; \rho) = C(\rho) \exp\left(-\frac{1}{2}\mathbf{z}(\mathbf{u})^\top \left(\frac{\rho}{1-\rho}\mathbb{I} - \frac{\rho}{(1-\rho)(1+(d-1)\rho)}\mathbf{1}_d \mathbf{1}_d^\top\right) \mathbf{z}(\mathbf{u})\right) \tag{3.3}$$

We implement SPN leaves using Equation 3.3, however we parameterize the input to the SPN as $\mathbf{z}$ rather than $\mathbf{u}$. In order to do so, we first transform the already uniform transformed ($u_i = F(x_i)$) data to $\mathbf{z}$ by applying Φ^{-1} to each dimension of each data-point, obtaining $\mathbf{z}(\mathbf{u})$ as defined above.

The uniform Gaussian copula has the constraint $\rho \in [-1, 1]$. In order to enforce this when optimising ρ we perform the change of variable $\rho = \tanh(x)$ and optimise with respect to the unconstrained variable x.

Chapter 4

Sum-Product Copula (SPC)

In this chapter, we introduce the main result of this thesis: **modifying a regular SPN density such that it satisfies the copula constraints of having uniform marginals**. We present the key concepts of our proposed approach for learning a copula density using an SPN. We call this model a Sum-Product Copula (SPC). We follow the copula notation introduced in Section 2.2. First, a regular SPN is implemented in PyTorch and automatic differentiation is used for its optimization. We will detail how the distribution of the SPN is modified to satisfy the constraints of a copula.

4.1 Obtaining a Sum-Product Copula Density

As we have seen in Section 2.2, copula functions allow the factorization of joint densities as a product of marginal densities and a copula density, derived from Sklar's Theorem. Given random variables $\{X_1, X_2, \ldots X_d\}$ and applying $u_d = F_d(x_d)$, we can factorize any joint CDF in terms of a copula distribution c and the univariate marginal PDFs $f_d(x_d)$:

$$\begin{aligned} f(x_1, x_2, \ldots, x_d) &= f_1(x_1) \times f_2(x_2) \cdots \times f_d(x_d) \times c(F_d(x_1), F_d(x_2), \ldots, F_d(x_d)) \\ &= f_1(x_1) \times f_2(x_2) \cdots \times f_d(x_d) \times c(u_1, u_2, \ldots, u_d). \end{aligned} \tag{4.1}$$

4.1.1 Estimating Uniform Marginals

The first step is estimating $F_d(x_d)$, which can be done empirically, using empirical CDF, as explained in Section 2.2.4. We obtain the new copula transformed variables $u_d = F_d(x_d)$. The PDF marginals $f_d(x_d)$ are also estimated independently of the copula, using KDE.

4.1.2 Copularization: Building New Copulas

The next step is estimating the unknown copula density c. This can be done using known parametric families, such as the Gaussian copula introduced in Section 2.2.2. However, known copula families are usually less powerful in high dimensional spaces, especially for multimodal data.

Equation 2.1 can also be viewed as a way of deriving a copula density function, given a known multivariate joint density. We have previously described this as a "copularization" step, allowing us to come with c:

$$\begin{aligned} c(u_1, u_2, \ldots, u_d) &= \frac{f(x_1, \ldots, x_d)}{\prod_d f_d(x_d)}. \\ &= \frac{f\left(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d)\right)}{f_1\left(F_1^{-1}(u_1)\right) \times f_2\left(F_2^{-1}(u_1)\right), \ldots, f_d\left(F_d^{-1}(u_d)\right)}. \end{aligned} \tag{4.2}$$

4.1.3 SPN Copula Density

We can write 4.2 as long as we can come up with a sufficiently good approximation for f. We will use an SPN to represent the dependence structure, and we estimate f by ψ, which will be given by the SPN evaluated at the root. We also require the marginal densities ψ_d and the marginal quantiles Ψ_d^{-1}. Here we can readily exploit the property of SPN that PDF and CDF joints and marginals are efficiently computed in at most one forward and one backward pass of the SPN, with the corresponding algorithms explained as operations that SPNs can perform in Section 2.1.3. The SPN Copula density is:

$$\begin{aligned} c_{SPN}(u_1, u_2, \ldots, u_d; \theta) &= \frac{\psi\left(\Psi_1^{-1}(u_d;\theta), \Psi_2^{-1}(u_d;\theta), \ldots, \Psi_d^{-1}(u_d;\theta); \theta\right)}{\prod_{d=1}^{D} \psi_d\left(\Psi_d^{-1}(u_d); \theta\right)} \\ &= \frac{\psi(y_1, y_2, \ldots, y_d; \theta)}{\prod_{d=1}^{D} \psi_d(y_d; \theta)} \end{aligned} \tag{4.3}$$

where the SPN Copula is written in terms of the SPN joint PDF ψ and CDF Ψ and their corresponding univariate marginal and θ represents the parameters of the SPN.

Here we have introduced *pseudo-data* $y_d = \Psi_d^{-1}(u_d; \theta)$, which becomes the input data to the SPN. We can now write an optimization objective in terms of the SPN paramaters. However, note that we will also need to satisfy $\psi_d^{-1}(u_d; \theta)$.

4.1.4 SPN Objective

Assuming we are working with a set of N CDF transformed observations $\{\mathbf{u}_n\}_{i=1}^N$ which are d-dimensional, we maximize the following objective, using logspace for numerical stability:

$$\begin{aligned}
&\max_{\theta} \sum_{n=1}^{N} \log c_{SPN}\left(u_{n,1},\ldots,u_{n,d};\theta\right) \\
=&\max_{\theta} \sum_{n=1}^{N} \left(\log \psi\left(\Psi_d^{-1}(u_{n,1};\theta),\ldots,\Psi_d^{-1}(u_{n,d};\theta);\theta\right) - \sum_{d=1}^{D} \log\left(\psi_d(\Psi_d^{-1}(u_{n,d};\theta);\theta)\right)\right) \\
=&\max_{\theta,\mathbf{y}} \sum_{n=1}^{N} \left(\log \psi\left(y_{n,1},\ldots,y_{n,d};\theta\right) - \sum_{d=1}^{D} \log\left(\psi_d(y_{n,d});\theta\right)\right), \text{ s.t. } y_{n,d} = \Psi_d^{-1}(u_{n,d};\theta)\forall_{n,d}.
\end{aligned} \tag{4.4}$$

Our goal becomes satisfying Equation 4.4, while enforcing $\Psi_d^{-1}(u_{n,d};\theta)$. This function given above can be optimized by standard gradient methods (using the Adam optimizer), but we also need to ensure the constraints are satisfied. The main focus of this section will describe the approaches that were employed for this and best practices for a theoretical motivated solution.

4.2 Satisfying the Copula Constraints

Equation 4.4 is a non-linear program. We can either turn it into a constrained optimization problem, or explicitly estimate $\Psi_d^{-1}(u_{n,d};\theta)$ at every step of the optimization process. The two techniques we propose are:

1. **Constrained** optimization approach
   Applying Ψ_d on both sides of the constraint in Equation 4.4, we obtain:

$$\begin{aligned}
\max_{\theta,\mathbf{y}} \sum_{n=1}^{N} \left(\log \psi\left(y_{n,1},\ldots,y_{n,d};\theta\right) - \sum_{d=1}^{D} \log\left(\psi_d(y_{n,d};\theta)\right)\right), \\
\text{s.t. } \Psi_d(y_{n,d};\theta) = u_{n,d}\forall_{n,d}.
\end{aligned} \tag{4.5}$$

   This parameterisation of the objective allows fitting the copula by maximum likelihood via a constrained optimisation approach problem that does not require explicit computation of the inverse CDFs Ψ_d^{-1}. This is convenient, since we do not have the form of the quantile function.

2. **Unconstrained** optimization, approach
   We directly optimize Equation 4.4 by explicitly inverting all $\Psi_d^{-1}(u_{n,d}), \forall n,d$ at each optimization step in order to evaluate the copula function and its gradient:

   - Numerically compute $y_{n,d} = \Psi_d^{-1}(u_{n,d};\theta)\forall_{n,d}$
   - Optimize Equation 4.4 for given $y_{n,d}$:

$$\max_{\theta} \sum_{n=1}^{N} \left(\log \psi \left(y_{n,1},\ldots,y_{n,d};\theta\right) - \sum_{d=1}^{D} \log \left(\psi_d(y_{n,d});\theta\right) \right) \tag{4.6}$$

   - Repeat until convergence

We outline the distinction between the two techniques. The first method has the advantage that we do not need to perform the inversion step of the CDF after every update. However, non-linear constrained optimization requires carefully choosing penalty coefficients, which make it difficult to use.

Note that the second class of methods are not to be confused with the projected gradient method. For a given constrained optimisation problem over a function $f(\mathbf{x})$ and a set of constraints $I = \{c_1(x),\ldots,c_{|I|}(x)\}$:

$$\min_{\mathbf{x}} f(\mathbf{x}),\ \text{s.t. } c_i(\mathbf{x}) = 0, \forall i \in I.$$

The projected gradient method uses the following updates at step t:

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \mathbf{x}_{t-1} - \lambda \nabla f(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_{t-1}} \\ \mathbf{x}_t &= \arg\min_{\mathbf{x}} ||\mathbf{x} - \tilde{\mathbf{x}}_{t-1}||^2,\ \text{s.t. } c_i(\mathbf{x}) = 0, \forall i \in I \end{aligned} \tag{4.7}$$

where Equation 4.7 is known as the projection step. This step consists in projecting back to the feasible set after carrying out an unconstrained step of gradient descent. In method 2 (Unconstrained optimisation) we have a step that could be confused with projection, however this is to compute the inverse CDF numerically in order to evaluate the copula function, which is not the same as the projection step in the projected gradients method. Applying the projected gradients method to our problem does not simplify the problem as the projection operator is itself an optimisation problem with non linear constraints ($\Psi(y_{n,d}) - u_{n,d} = 0$).

We proceed to describe common approaches to Constrained optimization and Unconstrained optimization methods.

1. Constrained optimization methods

Constrained optimization methods are used to solve problems over a function $f(\mathbf{x})$ and a set of constraints $I = \{c_1(x), \ldots . c_{|I|}(x)\}$:

$$\min_{\mathbf{x}} f(\mathbf{x}), \text{ s.t. } c_i(\mathbf{x}) = 0, \forall i \in I. \tag{4.8}$$

We present three methods for constrained optimization: the Penalty Method, Augmented Lagrangian Method and Newton-Lagrange Method.

Penalty Method

The main idea of penalty methods (Smith and Coit [31]) is to convert the constrained problem of Equation 4.8 into a sequence of unconstrained problems using a penalty function to achieve approximate feasibility.

We introduce the penalty coefficient μ_k and optimize the following now unconstrained problem:

$$\min \rho_k(\mathbf{x}) = f(\mathbf{x}) + \mu_k \sum_{i \in I} c_i(\mathbf{x})^2.$$

At each iteration k, we increase the penalty coefficient μ_k by a constant factor and solve the resulting unconstrained problem.
By applying the penalty method heuristic to our problem, the unconstrained objective becomes:

$$\max_{\theta, \mathbf{y}} \sum_{n=1}^{N} \left(\log \psi \left(y_{n,1}, \ldots, y_{n,d}; \theta \right) - \sum_{d=1}^{D} \log \left(\psi_d (y_{n,d}; \theta) \right) \right) + \mu \sum_{n=1}^{N} \sum_{d=1}^{D} \left(\Psi_d (y_{n,d}; \theta) - u_{n,d} \right)^2. \tag{4.9}$$

However, this method is known to suffer from ill conditioning. That is, to fully satisfy the original constrained problem (Equation 4.5) the multiplier μ becomes undefined ($\mu \to \infty$). This was confirmed in practice. Having only one penalty parameter for all dimensions and datapoints was also problematic.

Augmented Lagrangian

A way to circumvent the ill conditioning of the Penalty method is to use the Augmented Lagrangian method (Powell [32]). Given the same setup as the penalty method, the Augmented

Lagrangian introduces $\{\lambda_i\}_{i=1,I}$ for each constraint c_i, obtaining the following unconstrained objective:

$$\min \Lambda_k(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu_k}{2} \sum_{i \in I} c_i(\mathbf{x})^2 - \sum_{i \in I} \lambda_i c_i(\mathbf{x}).$$

In addition to updating μ_k after each iteration (which is also present in the Penalty method), we update each λ_i according to:

$$\lambda_i \leftarrow \lambda_i - \mu_k c_i(\mathbf{x}_k), \text{ where } \mathbf{x}_k \text{ is the solution found at the } k^{\text{th}} \text{ step.}$$

For our SPC objective, we require a separate λ_{nd} for each n datapoints and d dimensions.

The variable λ is an estimate of the Lagrange multiplier. This method has the advantage that it does not require $\mu \to \infty$ in order to satisfy the original constraint, avoiding ill-conditioning.

Newton-Lagrange Method

One of the most common approaches to solving non linear programs is the Sequential Quadratic Programming (SQP) method (Nocedal and Wright [33]). When presented with only equality constraints, as in our case, SQP reduces to applying Newton's method for optimisation directly to the Lagrangian.

The Lagrangian is given by:

$$\mathcal{L}(\mathbf{x}) = f(\mathbf{x}) - \sum_i \lambda_i c_i(\mathbf{x}),$$

and the Newton updates become:

$$\begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = -\eta \begin{bmatrix} \nabla^2_{\mathbf{x}} \mathcal{L} & \nabla_{\mathbf{x}\lambda} \mathcal{L} \\ \nabla^\top_{\mathbf{x}\lambda} \mathcal{L} & \nabla^2_{\lambda} \mathcal{L} \end{bmatrix}^{-1} \begin{bmatrix} \nabla_{\mathbf{x}} \mathcal{L} \\ \nabla_{\lambda} \mathcal{L} \end{bmatrix}. \tag{4.10}$$

This method searches for the Lagrangian's saddle points (Adolphs [34]) at which the constraints are satisfied and a local extrema of f is attained. However, the number of constraints scale with the size of the data (or batch size) and the number of variables $\mathbf{x}$ scale with both the size of the dataset and the number of parameters in the SPN. For most applications in machine learning, computing and inverting the Hessian in memory will be unattainable. In order to address this, we chose to approximate the Hessian-inverse and gradient product

iteratively using the L-BFGS method (Byrd et al. [35]). In practice this method fails to converge, we believe that this may happen for two potential reasons:

- The L-BFGS implementation provided by PyTorch is designed to minimise a loss and thus may have standard strategies to avoid saddle points.
- As mentioned in Adolphs [36], using Newton's method naively for constrained optimisation problems has the caveat that the method is attracted to any critical point and not just saddle points.

2. Unconstrained optimization methods

We explore direct approaches for obtaining the inverse of the univariate marginal CDFs, that is, finding y_d, such that $y = \Psi_d^{-1}(u_d)$. We invert this function numerically. First, we look at root finding approaches. We also explore linear interpolation as a simple way of obtaining the quantile function.

Root Finding

We can estimate the quantile function $\Psi_d^{-1}(u_d)$ by numerically finding the roots of:

$$g(x) = \Psi_d(x) - u_d = 0, \tag{4.11}$$

with respect to x. We can do this numerically by applying the Newton-Rhapson method to Equation 4.11, leveraging the fact that the derivative of the CDF $\Psi_d(x)$ with respect to x is just the PDF $\psi_d(x)$. This yields the following iterative updates:

$$\begin{aligned} x_t &= x_{t-1} - \frac{g(x_{t-1})}{g'(x_{t-1})}, \\ x_t &= x_{t-1} - \frac{\Psi_d(x_{t-1}) - u_d}{\psi_d(x_{t-1})}, \end{aligned} \tag{4.12}$$

which we run until convergence every time we wish to evaluate the copula PDF. Newton's method has the caveat of being very sensitive to initialisation and highly numerically unstable when initialised in areas where the derivatives of the function are close to 0. As a heuristic to mitigate this we cast the root finding problem as an optimisation problem:

$$\min_x ||\Psi_d(x) - u_d||^2, \tag{4.13}$$

which we solve with gradient based methods, using an Adam optimizer. In practice, we can first use the Newton-Rhapson method at train time where we have a good heuristic to initialise x_0, and minimise Equation 4.13 at test time where a lack of good initialisation is affecting Newton-Rhapson. We find that Newton-Rhapson converges much faster at train time which motivates the combination of both methods. Other root finding methods also exist, such as Halley's method and the Bisection Method, but we found that a combination of Newton-Rhapson for inversion at training time and a gradient based method at test time is the fastest and most accurate solution. To contrast approaches, we monitor the maximum difference $g(x) = \Psi_d(x) - u_d$.

Linear Interpolation

A straightforward approach to inverting the CDF is to simply use linear interpolation, which has also been used in Tewari et al. [29]. Linear interpolation is a curve fitting method that uses linear polynomials and it can be used for approximating the value of a function f. In our case, the function f that we aim to learn is each of the Ψ_d. In order for this method to work, we need to provide pairs of the form $(x, \psi(x))$, since we know $\psi^{-1}(\psi(x)) = x$. Then, the learnt function ψ^{-1} can be applied to the copula-space transformed data, u. We need to provide datapoints x such that $\psi(x)$ will provide a good coverage of the $[0,1]$ interval, or use extrapolation. This approach has proven to be very effective for most datasets. It is also common to use spline interpolation when the interpolation error is not low enough.

4.3 Derivatives of Implicit Functions

By finding a numerical estimation of the quantile function, we are introducing an implicit function for which the derivatives are not straightforward to obtain. We will now derive these derivatives, which Tewari et al. [29] have not accounted for.

A downside of the unconstrained optimization approaches (Root Finding and Linear Interpolation) is that given they estimate $\ln c_{SPN}(u_1, u_2, \dots u_d; \theta)$ directly and estimate $\Psi_d^{-1}(u_{n,d}; \theta)$ numerically, automatic differentiation is no longer able to estimate the full partial derivative, since $\Psi_d^{-1}(u_{n,d}; \theta)$ is a function of θ and autograd (PyTorch) cannot back-propagate through a numerical estimate of the quantile function. To illustrate this more succinctly we define $\ln c_{SPN}$ as $\tilde{c}$, obtaining the following:

$$\begin{aligned} &\tilde{c}\left(\Psi_1^{-1}(u_{n,1}; \theta), \Psi_2^{-1}(u_{n,2}; \theta), \dots \Psi_d^{-1}(u_{n,d}; \theta), \theta\right) \\ &= \tilde{c}\left(y_1(u_{n,1}, \theta), y_2(u_{n,2}, \theta), \dots y_d(u_{n,d}, \theta), \theta\right). \end{aligned} \tag{4.14}$$

Its full partial derivative (by the multivariate chain rule) is given by:

$$\frac{\partial}{\partial \theta_i}\tilde{c}\left(y_1(u_{n,1},\theta),y_2(u_{n,2},\theta),\dots y_d(u_{n,d},\theta),\theta\right) = \frac{\partial \tilde{c}}{\partial \theta_i} + \sum_{j=1}^{d}\frac{\partial \tilde{c}}{\partial y_j}\frac{\partial y_j}{\partial \theta_i} \tag{4.15}$$

Derivatives $\frac{\partial \tilde{c}}{\partial \theta_i}, \frac{\partial \tilde{c}}{\partial y_j}$ [1] can be computed by PyTorch, however $\frac{\partial y_j}{\partial \theta_i}$ is the derivative of the quantile function with respect to θ_j and we do not have an analytic expression for the quantile function. Using the method of implicit differentiation we provide a way of estimating the derivatives of the quantile function in terms of the quantile function itself.
First we re-express the quantile function as a root equation:

$$\Psi_j(y_j;\theta) - u_{n,j} = 0 \tag{4.16}$$

The y_j that satisfies this root equation corresponds to the quantile function. We can now take the total derivative of equation 4.16:

$$\begin{aligned}\frac{\partial}{\partial \theta_i}\left(\Psi_i(y_j;\theta) - u_{n,j}\right) &= 0\\ \frac{\partial \Psi_i(y_j;\theta)}{\partial y_j}\frac{\partial y_j}{\partial \theta_i} + \frac{\partial \Psi_i(y_j;\theta)}{\partial \theta_i} &= 0,\end{aligned}$$

rearranging for $\frac{\partial y_j}{\partial \theta_i}$:

$$\begin{aligned}\frac{\partial y_j}{\partial \theta_i} &= -\left(\frac{\partial \Psi_i(y_j;\theta)}{\partial y_j}\right)^{-1}\frac{\partial \Psi_i(y_j;\theta)}{\partial \theta_i}\\ \frac{\partial y_j(u_{n,j};\theta)}{\partial \theta_i} &= -\frac{1}{f_j(y_j;\theta)}\frac{\partial \Psi_i(y_j;\theta)}{\partial \theta_i}\end{aligned} \tag{4.17}$$

Substituting the numerical estimate for the quantile function y_i obtained by root finding or interpolation into Equation 4.17 gives us a numerical estimate for the gradient of the quantile function. Whilst we provided the derivation of the full partial derivative, we found that empirically using the term $\frac{\partial \tilde{c}}{\partial \theta_i}$ as the estimate of the gradient proved to be sufficient in fitting the copula for the experiments we performed.

[1] $\frac{\partial \tilde{c}}{\partial \theta_i}$ is a proper partial derivative that treats $y_j(u_{n,j},\theta)$ as constants with respect to θ_i. $\frac{\partial}{\partial \theta_i}\tilde{c}\left(y_1(u_{n,1},\theta),y_2(u_{n,2},\theta),\dots y_d(u_{n,d},\theta),\theta\right)$ does not treat $y_j(u_{n,j},\theta)$ as constants with respect to θ_i

4.4 Overview of Implementing an SPC

The main data structure of our approach is a normal SPN, but the objective and inputs will be different, so that the SPN can satisfy the copula constraints. Since our aim is to maximize the copula distribution function, we optimize the following quantity (Equation 4.5) or it's unconstrained equivalent (Equation 4.6):

$$\max_{\theta,\mathbf{y}} \sum_{n=1}^{N} \left(\log \psi \left(y_{n,1}, \ldots, y_{n,d}; \theta \right) - \sum_{d=1}^{D} \log \left(\psi_d(y_{n,d}; \theta) \right) \right), \quad \text{s.t. } \Psi_d(y_{n,d}; \theta) = u_{n,d}, \forall_{n,d}. \tag{4.18}$$

The first term, $\psi \left(y_{n,1}, \ldots, y_{n,d}; \theta \right)$ can be obtained by a bottom-up evaluation of the SPN, representing its value evaluated at the root node. The univariate marginal PDFs $\psi_d(y_{n,d}; \theta)$ can be computed using the differential approach, as shown in the Background section 2.1.3 on SPN marginalisation. The univariate marginal CDFs $\psi_d(y_{n,d}; \theta)$ are obtained in the same fashion, except that the input distributions are now CDFs instead of densities. Marginalisation only requires one bottom-up and one top-down pass. Moreover, the marginals of all variables can be computed at once.

Algorithm 5: Training an SPC

input : u, an NxD matrix of copula transformed data; SPN S
Initialize : θ_0 SPN parameters randomly
Set : $l_{min} = \infty$

repeat
  y = $\psi^{-1}(u; \theta_t)$;
  if *S.get_log_marginal_cdf(y;*θ_t*) - u > tolerance* **then**
    return best solution
  end
  cost = -(S(y, θ_t) - $\sum_n \sum_d$ S.get_log_marginal_pdf(y, θ_t)) ;
  if $cost < l_{min}$ **then**
    save_parameters(θ_t)
    $l_{min} = cost$
  end
  θ_{t+1} = θ_t - $\eta \nabla$ cost(θ_t)
until *stopping condition*;

In Algorithm 5, we summarize the main steps for building an SPC. Note that the SPN operations are implemented in log space, thus the algorithm is updated accordingly.

The SPC implementation is done in PyTorch, making use of automatic differentiation. Where feasible, derivatives are computed explicitly, as in the case of the differential approach to marginalisation, which can be achieved by accumulating corresponding weights. The SPN components are implemented as tensors, following RAT-SPN (Peharz et al. [1]). The input distributions are diagonal Gaussians and the parameters are shared for the same variable appearing in different input nodes, similar to the implementation in SPFlow [37]. All operations are carried out in log space for computational stability and the *logsumexp* trick is used. The weights of sum nodes are re-parameterized via log-softmax, so that they are normalized and non-negative. Root finding methods are implemented in NumPy. Correctness is ensured by unit testing SPN operations against their expected output.

4.4.1 Choosing the Best Optimization Approach

In Section 4.2, we have exposed several methods used to solve our optimization problem. We monitor the largest error $\textit{constraint\_error} = \Psi(y_{n,d}) - u_{n,d}$. We set the tolerance of the *constraint_error* at 0.05. We also monitor the mean difference and standard deviation. From the constrained and unconstrained optimization methods, we have found empirically by experimenting with all techniques that the unconstrained approach performs best in practice and it is the most stable. That is, they obtain the lowest *constraint_error* and the highest value of the $logc_{SPN}$ objective. When the quantile function is too difficult to obtain a *constraint_error* within the tolerance interval, we refine the solution by optimizing $\min_{y_d} ||\Psi_d(y_d) - u_d||^2$ using gradient descent.

4.4.2 SPC Evaluation

Once we have trained an SPC with parameters θ, we can use the expression for the copula density $c_{SPN}(u_1, u_2, \dots u_d; \theta) = \frac{\psi(y_1, y_2, \dots y_d; \theta)}{\prod_{d=1}^{D} \psi_d(y_d; \theta)}$ and the KDE estimated marginals to express the final joint distribution, according to Equation 4.1. We also need to find the inverse CDFs for the dataset we are evaluating, but this only needs to be done once at test time. The approach detailed in Section 4.4.1 is used during both training and test time.

Having obtained $c_{SPN}(u_1, \dots, u_n)$, we can use it together with the KDE estimated marginals $f(x_d)$ to factorize the required multivariate joint f:

$$f(x_1, \dots, x_d) = c_{SPN}(u_1, u_2, \dots u_d; \theta) \prod_d f_d(x_d). \tag{4.19}$$

Other downstream applications can also be derived in a similar manner, for example conditional distributions. Since we separate the learning of the marginals f_d and the copula

c_{SPN} this method is effectively doing Maximum Likelihood Estimation for fixed f_d learned a priori.

Chapter 5

Experiments

In this chapter, we lay out our methodology and experimental results, comparing the performance of SPNs, SPCs and SP-CLs with GMMs performing density estimation on different real-valued datasets.

5.1 Methodology

5.1.1 Experimental Details

We compare average test log-likelihoods on several datasets. As a baseline, we fit one full covariance Gaussian (not a mixture), as this is often done for density estimation tasks ([38], [39]). We fit a regular SPN, SPC (copula constrained SPN), SP-CL (SPN with copula leaves) and diagonal GMM (viewed as a shallow SPN), all having the same number of input distributions. We optimize their parameters using gradient descent, with the Adam optimizer. The learning rate was kept 0.1 for SPNs and 0.001 for SPC and SP-CL and the rest of the Adam parameters are the default ones. The batch size was kept at 500. We employ early stopping based on the validation set performance, with a patience interval of 30 epochs.

For all datasets, we use a random 70-15-15 percent train-validation-test splits, except where otherwise mentioned (for datasets with a readily available split).

When comparing SPN and SPC/SP-CL, we first pick an SPN structure based on its validation performance. We report results on the corresponding SPC/SP-CL (same structure, same number of nodes). We also report the performance of a GMM with diagonal covariances and the same number of components as the number of input distributions of the SPN.

The structure of the SPNs is determined by random region graphs. We experimented with their parameters *split depth* $\in [0, 1, 2]$, *number of recursive splits* $\in [1, 2, 10]$, where a

split depth of 0 corresponds to a GMM. Other parameters are *number of input distributions* $I \in [5, 10]$, *number of sum nodes per region* $S \in [5, 10]$.

As mentioned in Uria et al. [39], mixtures of Gaussians are strong baselines for density estimation. Currently, state-of-the-art approaches to density estimations are in the line of neural autoregressive density-estimators. The advantage of our modified SPN approach is that it retains the benefits of exact tractable inference that SPNs have, while making use of the flexibility of the copula framework.

5.1.2 Evaluating Generative Models

We will detail our choice of using average log-likelihood as a performance metric for density estimation.

In their work on evaluation of generative models, Theis et al. [40] note that while **average log-likelihood** is widely used for measuring the modelling performance of generative models, care needs to be taken to ensure that the numbers reported are meaningful. Especially when working with images, we are treating discrete pixel values (usually in the range $[0, 256]$) as continuous random variables, but discrete data has differential entropy of negative infinity. This could result in arbitrarily high log-likelihoods. To avoid this problem, real-valued data is added to the discrete pixel values to *dequantize* the data, similar to Van den Oord and Schrauwen [41] and Uria et al. [39].

Moreover, Theis et al. [40] prove that adding the right amount of *uniform noise* makes the log-likelihood of the continuous model on the dequantized data be closely related to that of a discrete model on the discrete data.

Often, obtaining the log-likelihood is computationally intractable for some models, for example when the normalization constant is hard to compute. For this reason, it is common to evaluate models on downstream applications. However, SPNs come with the advantage of fast, tractable inference, which our SPC model preserves. Moreover, average likelihood is used in many important works on density estimation (NADE [42], MAF [38], etc.), as well as in works on copulas (Copula Bayesian Networks [29], Tree-Averaged Copulas [25]). Therefore, average log-likelihood will be our evaluation method of choice. As a sanity check, we will also plot estimated densities for 2D datasets, making sure the results are sensible.

5.1.3 Choice of Datasets

We perform density estimation on a range of real-valued datasets. We have chosen continuous valued data, so that we can compute the copula function.

We get an intuition of relative performances by first performing density estimation on two 2D synthetic datasets: correlated data generated by Gaussian copulas and the Moons dataset. We then perform several standard density tasks on UCI (Asuncion and Newman [43]) datasets. To prove the high-dimensional capacities of our approach, we experiment on high dimensional image datasets: MNIST (LeCun and Cortes [44]) and CIFAR10 (Krizhevsky et al. [45]).

We included some datasets that have relatively low dimensionality, because they contain non-linear dependencies that might still make it difficult to fit GMMs.

The datasets we have considered are:

- Synthetic 2-dimensional datasets

    1. **Gaussian Copula Synthetic Dataset (GCS)**:
    We start with a 2D synthetic dataset, with data sampled with equal probability from two different Gaussian copulas. The process of generating correlated data using a copula was explained in Section 2.2.5. For each data point we uniformly sample a correlation ρ_i from a set of two values $\{\rho_1, \rho_2\} = \{0.9, -0.9\}$ then we use Algorithm 4 to sample from uniform Gaussian copula $c(u_1, u_2; \rho_i)$. This is equivalent to sampling from a mixture of copulas with correlations $\{0.9, -0.9\}$. We then transform the vector (u_1, u_2) by applying a quantile function of a GMM with 2 components centred at different points $(x_1, x_2) = (\Psi^{-1}_{\mu_1,\mu_2}(u_1), \Psi^{-1}_{\mu_1,\mu_2}(u_2))$. In this way, the resulting data is correlated and has multiple modes, thus making it more challenging to perform density estimation. We have drawn a total of 10000 samples.

    2. **Moons dataset**:
    The Moons Dataset [46], which will be more difficult for a GMM to fit. This dataset consists of two interleaving half circles which have been synthetically generated and random noise added. We have used a total of 10000 samples.

- Real-Valued UCI Datasets As a general preprocessing step for UCI datasets, we eliminated discrete-valued attributes. Additionally, we eliminated attributes in each pair with Pearson correlation coefficient greater than 0.98, following Tang et al. [47]. We normalize each dimension by subtracting its training sample mean and dividing by its standard deviation. All these steps were also performed in Papamakarios et al. [38].

    1. **Red Wine Dataset and White Wine Dataset:**
    The Wine dataset [48] contains two datasets regarding the quality of red wine and

white wine. Both datasets contain 11 features. The White Wine datasets contains 4898 total samples, and the Red Wine Dataset contains 1599.

2. **Crime Dataset**:
   The Crime Dataset [49] contains socio-economic data from the US, useful for predicting per capita crimes. Discrete features were discarded. There are 89 features and 1994 samples.

3. **Power Dataset**:
   The Power Dataset [50] consists of measurements of electric power consumption. We discarded discrete features and ignored the time series information and adding random noise. There are 6 features and $2,049,280$ samples.

4. **Gas Dataset**:
   The Gas Dataset consists of the readings of chemical sensors exposed to gas mixtures. Similarly, time series information is ignored. We discarded highly correlated attributes. There are 8 features and $1,052,065$ samples.

- High-dimensional image datasets (784 and 1024 dimensions): As we already mentioned, it is important to dequantize the data by adding uniform random noise (Theis et al. [40]). We then rescale the pixel values into $[0,1]$. We follow the same preprocessing steps as Papamakarios et al. [38] and Dinh et al. [51]: In addition to the random noise, the data is also transformed to *logit* space: $x \to \text{logit}(\lambda + (1-2\lambda))$, where $\lambda = 10^{-6}$ for MNIST and $\lambda = 0.05$ for CIFAR10. The reason for this is that the original data has bounded magnitude, which we want to avoid.

  1. **MNIST**: Images of handwritten digits (0-9), containing 784 features (28×28 black and white images). The MNIST dataset contains a total of 70000 sample images, with 60000 training and 10000 test. We set aside 5000 samples of the training set as validation.

  2. **CIFAR10**: Images of 10 classes of natural images, containing 32×32 RGB images ($1024 \cdot 3$ total features). The original dataset contains a total of 60000 total images, with 10000 being reserved for testing. We set aside 5000 samples from the training sat as validatiion. Following Papamakarios et al. [38], we augment the training set with all possible horizontal flips of the training images, resulting in 9000 training images, 5000 validation and 10000 test. We convert the 3-channel RGB input images to grayscale, obtaining 1024 features, since square images are more convenient for using the Poon architecture.

5.2 Experimental results

5.2.1 2-D Synthetic Data

Gaussian Copula Synthetic Data (GCS)

Since there are only two dimensions, the SPN structure reduces to a GMM.

Transformed space: Figure 5.1a shows the KDE fit of the univariate marginal distributions of each dataset. Figure 5.1b shows the data in copula space, after estimating the Empirical CDF: this is the data our SPC will use.

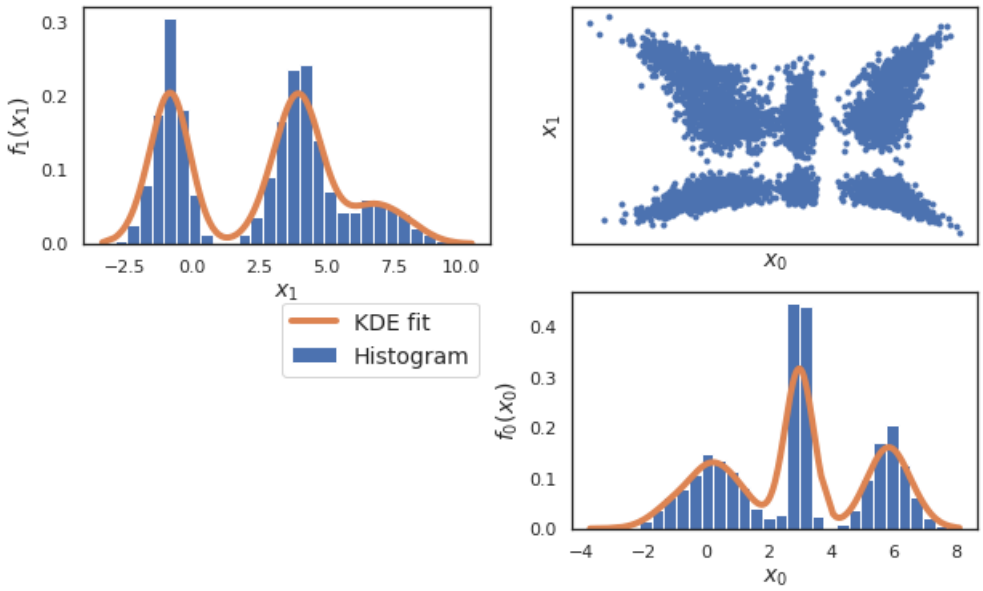
SPN/SPC/SP-CL fit: Figure 5.2a shows the contour plot fit of a normal SPN, while 5.2b shows the fit as estimated by SPC. We also show the fit of an SP-CL in 5.2c. Note that the contour has less levels for SP-CL here because when evaluating points on a grid, there are numerical issues when computing the inverse CDF of zero probability points. All methods manage to fit the data fairly well. However, the interesting aspect to note is that an SPN with 10 copula leaf distributions per input node only has 20 parameters (10 weights and 10 elements for correlations), since we used diagonal correlation matrices. Our other two models on the other hand, have 50 parameters (10 weights, 20 elements of the covariance matrices, 20 means), showing a copula can be a powerful modelling tool, while working with fewer parameters. Moreover, this synthetic experiment shows the potential of our proposed method, as its advantages might become more obvious in higher dimensions.

It is interesting to visualize the data after transforming it into copula space (through KDE estimation of marginal CDFs), as in Figure 5.3. Since a copula only needs to model the correlations in the data, we have separated the PDF marginal components. In this new transformed space, modelling might be easier than in the original space. We compare our SPC and SP-CL approaches, noting that both tend to allocate most mass towards the corners.

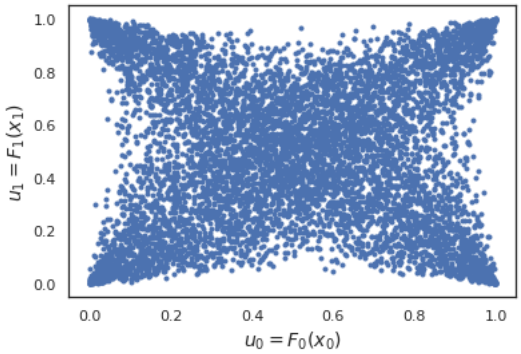
Copula models require less parameters: We further investigate the fact the copula version of SPNs can model data while using fewer parameters than a normal SPN. Figure 5.4 fits an SPN and an SPC on the synthetic datasets, using only 2 Gaussian input distributions per leaf. While the SPN does not manage to fit the data at all, the SPC and SP-CL fit is almost as good as before, when 10 distributions where used. This toy example confirms the power of copulas, even when fewer parameters are used.

Constrained and Unconstrained Optimization

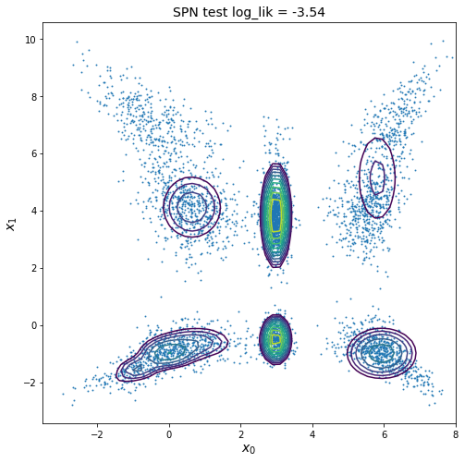
We will show experimentally why we have chosen to train SPCs as an unconstrained opti-



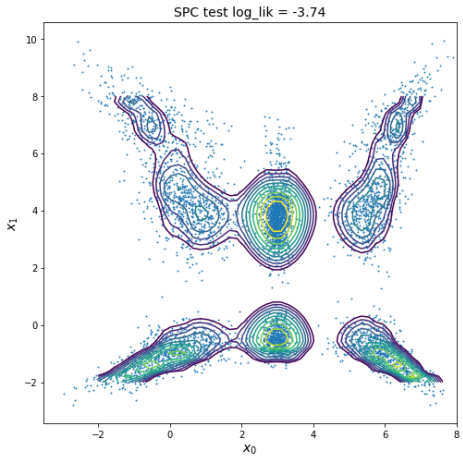
(a) Scatter plot of the GCS Dataset. The histograms on the side represent the univariate marginal distributions $f_d(x_d)$.



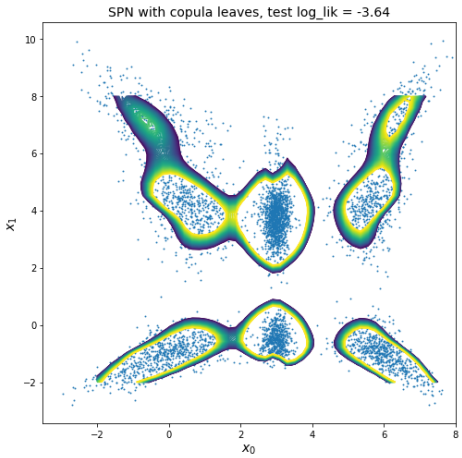
(b) GCS data in copula space, after estimating the Empirical CDFs.



(a) SPN

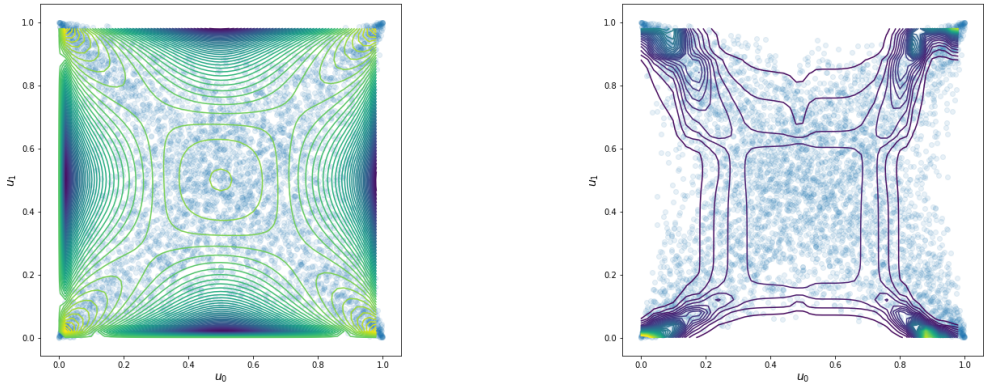


(b) SPC



(c) SP-CL

Figure 5.2 GCS Dataset fit of different models, using 10 input distributions per leaf



(a) SP-CL

(b) SPC

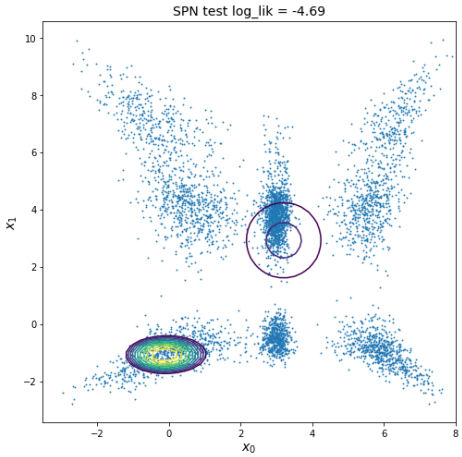
Figure 5.3 GCS Dataset fit in copula space.

mization problem. We analyse the learning process of each proposed method. Figure 5.5 compares the average log likelihood per epoch for each of the SPN variants: SPN, SPC and SP-CL, showing that they all achieve similar performances.
We have included an analysis of two ways of training an SPC as a constrained optimization approach using the Augmented Lagrangian method, or the unconstrained optimization approach using linear interpolation. As mentioned in Section 4.4.1, the unconstrained approach performs best in practice. We take a closer look at the differences between the two approaches by tracking both the average log likelihood during training, as well as statistics regarding the current difference between the target uniform marginals and the actual marginals (maximum difference, mean and standard deviation of the differences).

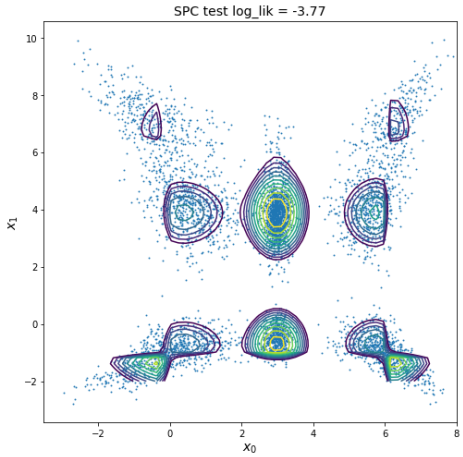
The unconstrained approach maintains a maximum difference of less than 0.005, which was expected since the constraint is enforced at every step. We note that the differences tend to increase during the optimization process, which could happen because the constraint becomes more difficult to enforce. The log likelihood is always increasing.

The constrained approach shows an oscillatory behaviour, which could happen because its penalty coefficients are not properly set. The log likelihood also becomes unstable, as the optimizer struggles to trade-off enforcing the constraints.

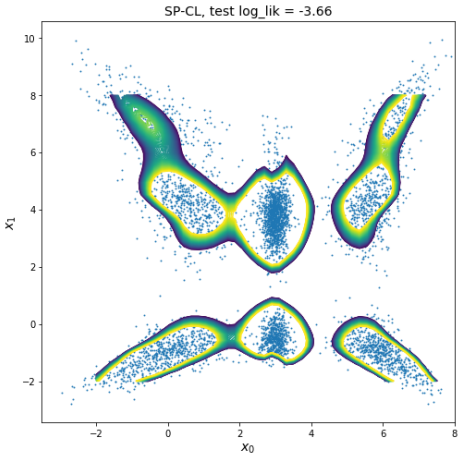
Following these observations, for the rest of our experiments, the copula constraints are enforced using the linear interpolation approach.



(a) SPN



(b) SPC



(c) SP-CL

Figure 5.4 Comparison between the SPN and SPC fit on GCS, when only using 2 input distributions per leaf.

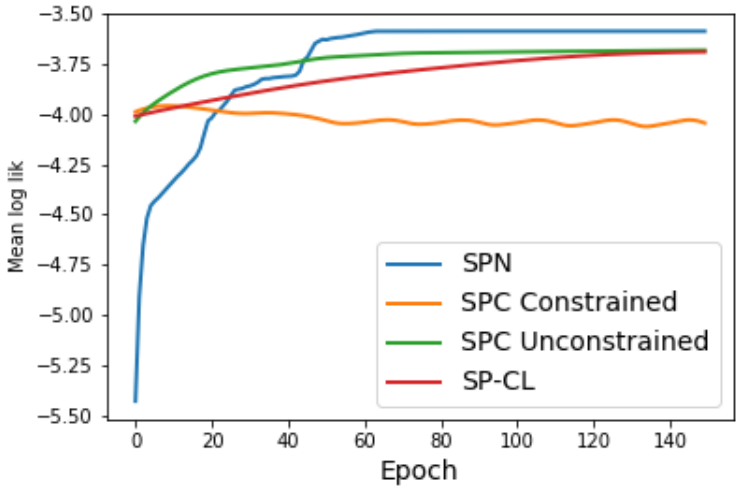
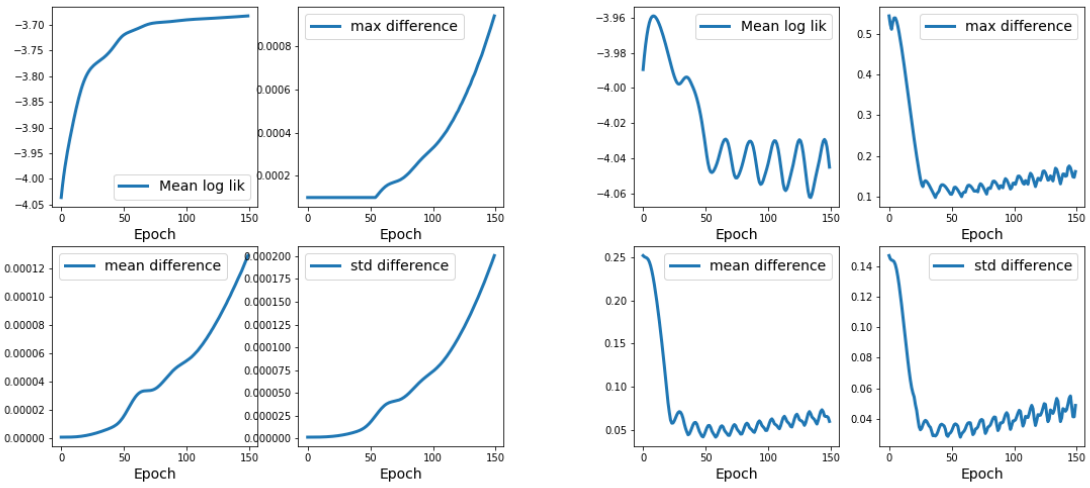


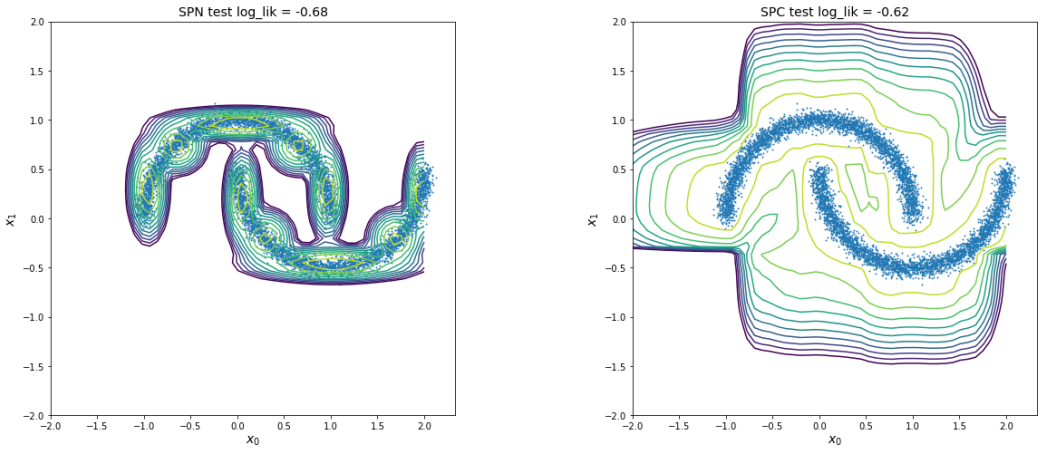
Figure 5.5 Average Log likelihood per epoch for each flavour of SPN considered: standard SPN, SPC SP-CL, evaluated on GCS.



(a) Unconstrained Optimization Approach.

(b) Constrained Optimization Approach.

Figure 5.6 Training time statistics and satisfying the copula constraint for SPC trained using Unconstrained Optimization (Linear interpolation) and Constrained Optimization Methods (Augmented Lagrangian)



(a) SPN fit (b) SPC fit.

Figure 5.7 Comparison between the SPN and SPC fit on the Moons synthetic dataset.

	N train	D	Full Gaussian	GMM	SPN	SPC	SP-CL
Red wine	1119	11	-12.85	-11.24	-10.98	**-10.65**	-11.75
White wine	3428	11	-12.95	-12.73	**-12.02**	-12.17	-13.33
Crime	1395	89	-48.59	-79.68	-68.92	-77.1	**-67.57**
Power	1,659,917	6	-7.74	-1.23	-0.14	**-0.09**	-1.04
Gas	852,174	8	-4.43	-1.13	-0.167	**0.05**	-3.53

Table 5.1 Average test log likelihood on UCI datasets. N train represents the number of training samples, D is the number of dimensions.

Moons Dataset

Figure 5.7 compares the SPN and SPC fits, showing that the SPC fits the data better visually, while also achieving a higher test mean log likelihood. However, this dataset is too complex to be modelled with an SP-CL model with uniform Gaussian copula leaves.

5.2.2 UCI Datasets

We evaluate our method in a systematic manner on several real-valued UCI datasets with different number of dimensions and sample sizes.

Table 5.1 summarizes the results for the five UCI datasets. We note that in the case of the Crime dataset, one full covariance Gaussian is by far superior to a mixture of diagonal Gaussians. Both our SPC and SP-CL approaches show promising results, being competitive with SPNs, especially considering the lack of hyperparameter tuning done for the copula variants, due to their slower training times.

	N train	D	Full Gaussian	GMM	SPN	SPC	SP-CL
MNIST	50000	784	-1366	-1516	-1358	**-1129**	-1244
CIFAR10	90000	1024	90.04	-1159	**-640**	-833	-909

Table 5.2 Average log-likelihood (in nats) on image datasets. Images have been transformed in logit space. N train represents the number of training samples, D is the number of dimensions.

5.2.3 Image Datasets

We assess our method on highly dimensional image datasets.

In terms of structure, the Poon architecture is used, with coarse decompositions (see Section 2.1.4) leading to leaves of size $\Delta \times \Delta$. We have used $\Delta = 8$ for MNIST and $\Delta = 7$ for CIFAR, as in Tan and Peharz [52]. We use $I = 20$ Gaussian input distributions with diagonal covariances and $S = 20$ sum nodes per region. We use ten percent of the training data as validation data.

Table 5.2 shows the average log-likelihoods obtained on the two image datasets, MNIST and CIFAR10. SPC and SP-L are superior to a standard SPN on MNIST, but perform worse on CIFAR10. Note that the best result on this dataset is actually one full covariance Gaussian, with a GMM performing far worse. This is also consistent with literature: as reported in [38], a MADE (Masked Autoencoder for Distribution Estimation, Germain et al. [53]) performed significantly worse than the Gaussian baseline.

5.3 Discussion

Our results show that it is possible to train SPNs that satisfy the copula constraints copula, even when varying the size of the datasets and the number of dimensions.

One thing to note is that training SPC could benefit from further tuning their learning rate instead of keeping it fixed as it was done in our experiments. Our SPC is also limited by the underlying SPN: all SPNs leaves are diagonal Gaussians, where full covariances could be helpful.

While the framework we have presented is principled, there are some approximate steps and more research in each of the steps involved could lead to further performances. For example, we are using KDE estimators with default Gaussian kernels, but the kernels could be cross-validated. The optimization process can be slow because it involves inverting CDFs at each step. Nonetheless, we have shown that an SPC is usually at least as good as an SPN and it some cases slightly better.

Moreover, it is important to put our method into perspective. First of all, copula applications are limited to small dimensions (< 10), yet we show that we can build copulas for hundreds of dimensions. Our approach also incorporates the benefits of an SPN as a probabilistic model. Especially for the image datasets, we report the first copulas successfully trained on datasets with several hundreds of dimensions.

Chapter 6

Conclusions

6.1 Conclusion

Our efforts contribute to the recent idea that the framework of copulas can be beneficial in the wider context of machine learning. Specifically, their power consists in separating modeling univariate marginal distributions from modeling correlations between variables. We combine copulas with SPNs, which are deep probabilistic models that offer exact and tractable inference. This marriage allows us to take advantage of the benefits of both ideas.

We have introduced two methods of building highly-dimensional copulas. Our first result is that a decomposable and complete sum-product network is in itself a copula (SP-CL). However, this approach is limited by the type of copulas used as input distributions. We then process to distort any SPN such that its univariate marginals come from a uniform distribution, by enforcing this during training (SPC). We investigate approaches for satisfying these constraints, with methods ranging from constrained optimization to approaches that directly compute the quantile function.

We obtain promising results on real-valued density estimation and go on to show we have obtained a version of copulas that are suitable for high-dimensional datasets. In contrast, known parametric copulas are usually more limited. Another interesting observation is that an SPC can often obtain similar performances to an SPN, while using less parameters, as it was shown by the synthetic experiments.

6.2 Summary of Contributions

The key contributions of this thesis are:

- PyTorch implementation of an SPN
- New result and proof that an SPN with copula leaves defines a copula
- Implementation of an SPN with Gaussian copula leaves with uniform correlation (SP-CL)
- End-to-end PyTorch framework for training an SPN Copula (SPC)
- An in depth analysis and comparison of constrained and unconstrained optimization approaches
- Computation of the derivatives of implicit functions
- Systematic experiments on different real-valued datasets

6.3 Further Work

There are several incremental improvements that could be done to our approaches to improve experimental results.

The SPN implementation could be changed to have full covariance Gaussian distributions as input leaves, instead of diagonal ones. When modeling images, variants of Deep Convolutional SPNs (as in Butz et al. [14]) could be employed.

Investigating practical advice on the Augmented Lagrangian method could accelerate the training process, without requiring inverting a function at every step, as it is currently required by the linear interpolation technique we are using.

For our SP-CL model, other types of copulas could be used as input (such as a Gaussian copula with full covariance), as we have only experimented with Gaussian copulas with a uniform correlation parameter, which is very limiting. Given the promising results this relatively straightforward method has shown, this change could have a significant impact on performance.

By introducing more powerful models (with full covariance distributions and possibly Convolutional structure) and further refining the optimization process, we could advance our method even further. However, we have already managed to provide a stand-alone framework for constructing high-dimensional copulas with the aid of SPNs.

Bibliography

[1] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[2] Daniel Zoran and Yair Weiss. From learning models of natural image patches to whole image restoration. In *2011 International Conference on Computer Vision*, pages 479–486. IEEE, 2011.

[3] George Papamakarios and Iain Murray. Distilling intractable generative models. In *Probabilistic Integration Workshop at Neural Information Processing Systems*, 2015.

[4] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[5] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[6] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

[7] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016.

[8] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On Theoretical Properties of Sum-Product Networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 744–752, San Diego, California, USA, 09–12 May 2015. PMLR.

[9] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, 2014.

[10] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247, 2012.

[11] Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pages 2033–2041, 2012.

[12] Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 612–627. Springer, 2013.

[13] Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880, 2013.

[14] Cory J. Butz, Jhonatan S. Oliveira, André E. dos Santos, and André L. Teixeira. Deep convolutional sum-product networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3248–3255, Jul. 2019.

[15] Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3699–3703. IEEE, 2014.

[16] Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming A Chai. Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[17] Mohamed R Amer and Sinisa Todorovic. Sum product networks for activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 38(4):800–813, 2015.

[18] Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[19] José Miguel Hernández-Lobato. An introduction to sum-product networks. accessed August 2019. URL https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf.

[20] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

[21] Mathematical institude University of Oxford. The gaussian copula and the financial crisis: A recipe for disaster or cooking the books? accessed August 2019. URL https://www.maths.ox.ac.uk/system/files/attachments/1000332.pdf.

[22] Abe Sklar. Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8:229–231, 1959.

[23] Roger B Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

[24] Gal Elidan. Copulas in machine learning. In *Copulae in mathematical and quantitative finance*, pages 39–60. Springer, 2013.

[25] Sergey Kirshner. Learning with tree-averaged densities and distributions. In *Advances in Neural Information Processing Systems*, pages 761–768, 2008.

[26] Gal Elidan. Copula bayesian networks. In *Advances in neural information processing systems*, pages 559–567, 2010.

[27] Veni Arakelian and Dimitris Karlis. Clustering dependencies via mixtures of copulas. *Communications in Statistics-Simulation and Computation*, 43(7):1644–1661, 2014.

[28] Roger B Nelsen. *An introduction to copulas*, pages 14, 72–74. Springer Science & Business Media, 2007.

[29] Ashutosh Tewari, Michael J Giering, and Arvind Raghunathan. Parametric characterization of multimodal distributions with non-gaussian modes. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 286–292. IEEE, 2011.

[30] David W Scott. Averaged shifted histograms: effective nonparametric density estimators in several dimensions. *The Annals of Statistics*, pages 1024–1040, 1985.

[31] Alice E Smith and David W Coit. Penalty functions. *Handbook of Evolutionary Computation*, 97(1):C5, 1995.

[32] Michael JD Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14(1):233–241, 1978.

[33] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[34] Leonard Adolphs. *Non Convex-Concave Saddle Point Optimization*, page 53. 2018.

[35] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[36] Leonard Adolphs. *Non Convex-Concave Saddle Point Optimization*, page 55. 2018.

[37] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. *CoRR*, abs/1901.03704, 2019. URL http://arxiv.org/abs/1901.03704.

[38] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[39] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013.

[40] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[41] Aaron Van den Oord and Benjamin Schrauwen. Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*, pages 3518–3526, 2014.

[42] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

[43] Arthur Asuncion and David Newman. UCI machine learning repository, 2007.

[44] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[45] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[46] Moons dataset, Accessed on 15 June 2019. URL https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html.

[47] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Deep mixtures of factor analysers. *arXiv preprint arXiv:1206.4635*, 2012.

[48] Wine quality dataset, Accessed on 15 June 2019. URL https://archive.ics.uci.edu/ml/datasets/wine.

[49] Communities and crime data set, Accessed on 15 June 2019. URL http://archive.ics.uci.edu/ml/datasets/communities+and+crime.

[50] Individual household electric power consumption data set, Accessed on 15 June 2019. URL http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

[51] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[52] Ping Liang Tan and Robert Peharz. Hierarchical decompositional mixtures of variational autoencoders. In *International Conference on Machine Learning*, pages 6115–6124, 2019.

[53] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.