,

Bachelor's Thesis

L_1 Metric for Probabilistic Circuits

Author: Maximilian Neumann

Matriculation Number: 6034716

23.01.2025

First Examiner: Prof.Michael Beetz PhD

Second Examiner: Dr.-Ing. Joachim Clemens

Advisor: Tom Schierenbeck

Universität Bremen

Nachname	Neumann	Matrikelnr.	6034716
Vorname	Maximilian		

A) Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet, dazu zählen auch KI-basierte Anwendungen oder Werkzeuge. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht. Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

☒ Ich habe KI-basierte Anwendungen und/oder Werkzeuge genutzt und diese im Anhang "Nutzung KI basierte Anwendungen" dokumentiert.

B) Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

☒ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

C) Einverständniserklärung zur Überprüfung der elektronischen Fassung der Bachelorarbeit / Masterarbeit durch Plagiatssoftware

Eingereichte Arbeiten können nach § 18 des Allgemeinen Teil der Bachelor- bzw. der Masterprüfungsordnungen der Universität Bremen mit qualifizierter Software auf Plagiatsvorwürfe untersucht werden.
Zum Zweck der Überprüfung auf Plagiate erfolgt das Hochladen auf den Server der von der Universität Bremen aktuell genutzten Plagiatssoftware.

☐ Ich bin damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum oben genannten Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutionseigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.

☒ Ich bin nicht damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum o.g. Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutionseigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.

Datum Unterschrift

Contents

1 Abstract

Modern artificial intelligence, machine learning, and robotic planning can depend on probabilistic models to effectively navigate the complexities and uncertainties inherent to real-world scenarios. One problem is that the generating of models from identical data is not guaranteed to produce identical results. Quantifying the disparity between these models and establishing a comparative framework is essential for understanding model behavior and performance. This thesis delves into the issue of quickly and reliably measuring the distance between two probabilistic models within the Probabilistic Circuits (PC) framework. The L_1 metric is employed to generate quantifiable values representing model (dis)similarity. While the L_1 provides a robust measure, its computation can be demanding. The shallow circuit transformation is needed to calculate the L_1 which is computationally costly to perform. Alternatively, a Monte Carlo approach to L_1 calculation is investigated, bypassing the need for the shallow transformation and tractable modelling altogether. This approach introduces a trade-off between computational efficiency and precision. The results of the L_1 and its Monte Carlo estimator across different models in robotics, especially Joint Probability Trees, are compared to see the effectiveness of the metric.

2 Introduction

Artificial intelligence (AI) models have become increasingly prevalent in many parts of life, offering significant benefits. A common example of how AI models could help in day-to-day living would be a navigation app that predicts traffic to achieve a better travel time [Sha+22]. Another example is pattern recognition in the field of medicine to find otherwise hardly detectable anomalies [Wan+24]. These and more tasks are reaching a level of complexity that renders them difficult to solve with classic means. Another reason for the accelerating adoption of AI is its ability to process huge amounts of data, providing solutions to problems that could never be realized on a smaller scale.

One last example is in the field of robotics, where robots are faced with the challenges of everyday activities. For instance, a robot may needs to know the exact position of a gripper arm for a mobile pick and place scenario [Dec]. However, real world nature is often difficult to understand and interpret. Even a seemingly simple task can involve a multitude of variables, including an object's properties, an agents capabilities, and the environment's characteristics. To navigate such complexity, probabilistic models provide a valuable framework. Probabilistic models can be constructed by learning from data, which serve as examples of the desired task. However, it's important to note that models trained on the same baseline data may not be entirely identical. This discrepancy can lead to different models for the same input data, making it challenging to assess the extent of their divergence. This thesis shows that an approach to quantify the dissimilarity between models is possible for Probabilistic Circuits. The method aims to show that a human-intuitive distance measure can also be applied to Probabilistic Circuits with specific constraints. In this case leveraging the L_1-metric, it can effectively evaluate the differences between these models. However, this approach can be computationally intensive. Therefore, it also compares with a Monte Carlo method estimator, which offers an alternative approach to calculating the L_1-distance in this context.

3 Probabilistic Circuits

For machine learning and robotic planning, probabilistic models offer an approach with which to address the inherent complexities and uncertainties encountered in real-world environments. These models provide a framework for dealing with incomplete or noisy data, enabling more robust and adaptable decision-making processes. While the trend towards larger models is a common topic, these kinds of models are not without problems.

> "These models are often trending favoring ever larger models trained on lager datasets. However this trend can lead to models that excel at pattern recognition within the training data but failing archiving solving the problem through probabilistic reasoning."
>
> — [CVB20] by Choi, Y and Vergari, Antonio and Van den Broeck, Guy

Tractable probabilistic models offer an alternative approach. [CVB20] introduces PCs as a unified framework for representing these kinds of models. PCs leverage computational graphs to define joint probability distributions. These graphs consist of nodes representing two key operations.

Sums: These nodes perform a mixture of the probability distributions computed by their subgraphs.

Products: These nodes perform a factorization of the probability distributions computed by their subgraphs.

Leaves: The leaves of these computational graphs represent simple probabilistic distributions.

The key advantage of PCs is highlighted by the authors:

> "PCs allow for exact probabilistic inference in time linear in the size of the circuits and the cost of performing it can be theoretically certified when the circuit has certain structural properties"
>
> — [CVB20] by Choi, Y and Vergari, Antonio and Van den Broeck, Guy

This allows for efficient reasoning in the modeled uncertain world, making PCs well-suited for various applications in machine learning and robotics. Within a PC, the uncertain world is represented by a collection of random variables (RVs). These variables act as the inputs or attributes of the model. Each RV possesses a supported domain, which defines the set of possible values it can take. The distribution describing the frequency in the supported domain of the RV can be either discrete where each element has an assigned probability or continuous where a probability density function exists, requiring integration to determine its probabilities. By combining these random variables within the PC, a joint probability distribution is constructed. This joint distribution encompasses all possible combinations of values for the model's random variables, effectively representing every conceivable state (or "world state") within the modeled world.

A fundamental functionality of probabilistic models lies in their ability to respond to queries. These queries act as questions posed to the model, prompting it to calculate an answer based on its internal representation of the world. Some queries require inclusion of information about the question. One element of this information is the evidence about the current state of the world. Evidence specifies a particular configuration of the model's random variables, setting parts of the world as certain. In probabilistic models, a query is formulated based on a specific event. This event represents a particular aspect of the world that the user is interested in. By focusing on this event, the model can direct its reasoning towards a relevant subset of possible outcomes. The exact nature of the query depends on the query class used. Different query classes can be defined to address various types of information needs.

In PCs, the fundamental building block is the input unit, these are the leaves of the graph. The simplest PCs imaginable consists solely of a single input unit. These units serve a purpose to encode the probability distribution associated with a specific random variable. The core functionality of an input unit revolves around taking the value of an RV and "casting" it through the defined probability distribution. This translates the observed state of the RV into a probability value based on the chosen distribution. Importantly, for efficient inference in PCs, this chosen distribution needs to be tractable. Tractable in this context means assuring two guarantees.

> "The first guarantee is that the model is able to perform exact inference: the answers to queries are faithful to the model's distribution, and no approximations are involved in obtaining them. The second guarantee is that the query computation can be carried out efficiently, that is, in time polynomial in the size of the probabilistic model."
>
> — [CVB20] by Choi, Y and Vergari, Antonio and Van den Broeck, Guy

For input units, simple distributions are ideal. The complexity of the unit is achieved by using the inner nodes of the PC instead of complex joint distributions.

At the leaf nodes of probabilistic models, simple continuous or symbolic distributions can be used to represent the probabilities of different outcomes. A continuous distribution assigns probabilities to an interval range. For example, an uniform distribution would assign equal probability to all values within a specific interval. A symbolic distribution is a discrete distribution that assigns probabilities to a specific set of elements within its domain. This could be a Bernoulli distribution for binary outcomes or a multinomial distribution for categorical outcomes.

Definition 3.1 (Factorized models [CVB20]): Consider the probabilistic model $\mathbf{m}$ encoding a joint probability distribution over a collection of RVs $\boldsymbol{X} = \cup_{i=1}^{k} \boldsymbol{X}_i$ partitioned into disjoint set $\boldsymbol{X}_i \cap \boldsymbol{X}_j = \emptyset$ for any $i \neq j$ in $1, ..., k$ where $k > 1$. Model $\mathbf{m}$ is a factorized model *iff*

$$p_{m(\boldsymbol{X})} = \prod_{i=1}^{k} p_{m_i}(\boldsymbol{X}_i)$$

where each p_{m_i} is a distribution over the subset of RVs $\boldsymbol{X}_i$

While input units provide a foundation for representing basic probabilistic relationships, describing even slightly complex real-world scenarios with a model requires the ability to express joint distributions. As seen in the in Definition 3.1, a joint distribution can be decomposed into a collection of marginal distributions over smaller sets of random variables. PCs achieve this capability through the introduction of product units. These units act as combiners, taking a collection of distributions as their inputs. The product unit then performs a computation, resulting in a new distribution that represents the joint distribution over the combined set of random variables. The key advantage of PCs lies in their ability to create complexity within their graph structure rather than relying solely on intricate input distributions. This approach allows for efficient representation and manipulation of complex relationships between variables using product units as the building blocks for joint distributions.

Definition 3.2 (Mixture models [CVB20]): Let $\left\{p_{m_i}\right\}_{i=1}^{k}$ a finite collection of probabilistic models, each defined over the same collection of RVs **X**. A mixture model is the probabilistic model defined as the convex combination

$$p_{m(\boldsymbol{X})} = \sum_{i=1}^{k} \theta_{ip_{m_i}}(\boldsymbol{X})$$

for a set of positive weights (called the mixture parameters)

$$\theta_i > 0, i = 1, ..., k \text{ and} \sum_{i=1}^{k} \theta_i = 1.$$

[...]

Simple probability density functions often lack the expressiveness needed to capture complex real-world data distributions. In PCs the role of sum units is to combine simpler distributions. As highlighted in the definition of mixture models, these simpler densities can be weighted and summed to create a mixture density. This mixture density possesses properties that cannot be described by its individual components alone. This characteristic allows PCs equipped with sum units to represent complex dependencies within a density function, while maintaining a well-defined and functional form.
The full definition of a PC is show in Definition 3.3, Definition 3.4, Definition 3.5 and Definition 3.6.

Definition 3.3 (Probabilistic Circuits (PCs) [CVB20]): A probabilistic circuit (PC) C over RVs **X**, is a pair (G, θ) where G is a computational graph, also called the **circuit structure** that is parameterized by θ, also called the **circuit parameters**, as defined next. The PC C computes a function that characterizes a (possibly unnormalized) distribution p(**X**).“

Definition 3.4 (PC structure [CVB20]): Let $C = (G, \theta)$ be a PC over RVs $\mathbf{X}$. G is a computational graph in the form of rooted DAG, comprising computational units, also called nodes. The standard evaluation ordering of G, also called feedforward order, is defined as follows. If there is an edge $n \to o$ from unit $n \in G$ to unit $o \in G$, we say n the input of o and o its output. Let $\text{in}(n)$ denote the set of all input units for unit $n \in G$ and equivalently, $\text{out}(n)$ denotes the set of its outputs. The input units of C are all units $n \in G$ for which $\text{in}(n) = \emptyset$. Analogously, the output unit of C, also called its root, is the unit $n \in G$ for which $\text{out}(n) = \emptyset$. The structure G comprises three kinds of computational units: input distribution units, product units and sum units, to which a scope is associated as formalized in the following definitions.

Definition 3.5 (PC structure: scope [CVB20]): Let $C = (G, \theta)$ be a PC over RVs $\mathbf{X}$. The computational graph G is equipped with a scope function φ which associates to each unit $n \in G$ a subset of $\mathbf{X}$, i.e., $\varphi(n) \subseteq \boldsymbol{X}$. For each non-input unit $n \in G, \varphi(n) = \cup_{c \in \text{ in}(n)} \varphi(c)$. The scope of the root of C is $\mathbf{X}$.

Definition 3.6 (PC structure: computational units):

Let $C = (G, \theta)$ be a PC over RVs $\mathbf{X}$. Each unit $n \in G$ encodes a non-negative function C_n over its scope: $C_n : \text{val}(\varphi(n)) \to \mathbb{R}_+$. A n input unit n in C encodes a non-negative function that has a support $\text{supp}(C_n)$ and is parametrerized by θ_n. A product unit n defines the product $C_{n(\boldsymbol{X})} = \sum_{c \in \text{ in}(n)} C_{c(\boldsymbol{X})}$. A sum unit n defines the weighted sum $C_{n(\boldsymbol{X})} = \sum_{c \in \text{ in}(n)} \theta_{n,c} C_{c(\boldsymbol{X})}$, parameterized by weights $\theta_{n,c} \geq 0$.

Summarized, a PC represents a probability distribution by composing simpler distributions at its input nodes. These base distributions are combined through product and sum operations to construct a more complex output distribution. An example is presented in Figure 1.

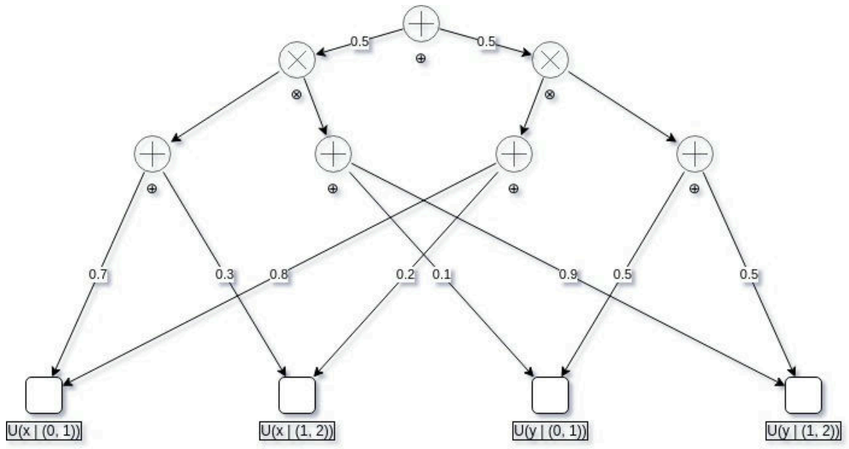

Figure 1: Example PC representing a joint distribution over variables X and Y.

The space of a PC offers a diverse range of models for representing probability distributions. However, not all PCs are equally suitable for the specific tasks addressed in this thesis. This thesis focuses on a subset of PCs with characteristics tailored for the needs of the L_1 metric. In the following section, definitions of the key attributes required for a "shallow PC" and L_1 metric are provided.

Definition 3.7 (Decomposability [CVB20]): A product node n is decomposable if the scopes of its input units do not share variables: $\varphi(c_i) \cap \varphi(c_j) = \emptyset, \forall c_i, c_j \in \text{in}(n), i \neq j$. A PC is decomposable if all of its product units are decomposable.

Definition 3.8 (Smoothness [CVB20]): A sum node n is **smooth** if its inputs all have identical scopes: $\varphi(c) = \varphi(n), \forall c \in \text{in}(n)$. A circuit is smooth if all of its sum units are smooth

Definition 3.9 (Determinism [CVB20]): A sum node is **deterministic** if, for any fully-instantiated input, the output of at most one of its children is nonzero. A circuit is deterministic if all of its sum nodes are deterministic.

Decomposability and smoothness are important properties in designing efficient and interpretable PC architectures. Decomposable PCs allow for modular reasoning of the product operations, which simplify the calculation. Without the decomposability the product would need to use integration by parts on every product. This would raise the complexity and calculation time for most systems. Furthermore, smooth PCs facilitate efficient computation and analysis of the sum operation. Both properties play a significant role in ensuring effective utilization of PCs for various probabilistic modeling tasks. In all cases, smoothness and decomposability are needed to create the shallow circuit. Finally, determinism is needed to ensure that the density calculation of a subset of the circuits domain is only subject to a subgraph of the PC. This enables not only the exact calculation of the mode of the entire distribution but is also helpful for calculating the L_1 metric.

This framework of PCs offers a powerful and tractable approach for reasoning under uncertainty in various machine learning and robotics tasks. Its ability to efficiently capture complex joint probability distributions from simple building blocks provides a valuable tool for navigating the inherently uncertain nature of the real world.

4 Learning Methods

One fundamental aspect in the field of machine learning and statistical modeling is the induction of knowledge from data. This process involves constructing statistical models or probability distributions that accurately represent the underlying patterns and relationships within a dataset. This dataset, a collection of observations or measurements, typically comprises instances characterized by a set of attributes relevant to the problem's domain. The process of transforming raw data into a structured representation, such as a model or distribution, is commonly referred to as learning. It entails the extraction of underlying regularities and the generalization of these patterns with respect to unseen data. In essence, learning is an approximation process where a simplified representation is derived to capture the essential characteristics of the observed data.

4.1 Nyga Distribution

The Nyga distribution is a non-parametric learning distribution using data to create a one dimensional distribution. It adopts a divide-and-conquer approach, progressively refining the model by identifying the most effective data partitions and approximating the distribution within each partition using a uniform model. This iterative process culminates in a PC that captures the complexity of the original data distribution through a combination of simpler uniform distributions. The fitness function based on the weighted log-likelihood is employed to evaluate the quality of the potential splits.

“$p(x) = \sum_{i=1}^{N} \log(w_i) - N \log(b - a)$”

— [Scha] by Tom Schierenbeck

For each split, the two resulting subsets are modeled using uniform distributions. The overall likelihood of the split is then calculated as the sum of the likelihoods of the individual subsets. This process identifies the split that maximizes the overall likelihood of the distribution. Only splits that lead to a likelihood improvement exceeding a set threshold are considered. This process of splitting and likelihood evaluation is repeated on the newly formed subsets until the improvement falls below the minimum value, indicating convergence.

4.1.1 Nyga Distribution Example

A visualization of the induction process of the Nyga Distribution is provided in Figure 2 to Figure 5 [Scha]. Initially, a uniform distribution is cast over the base data (Figure 2).

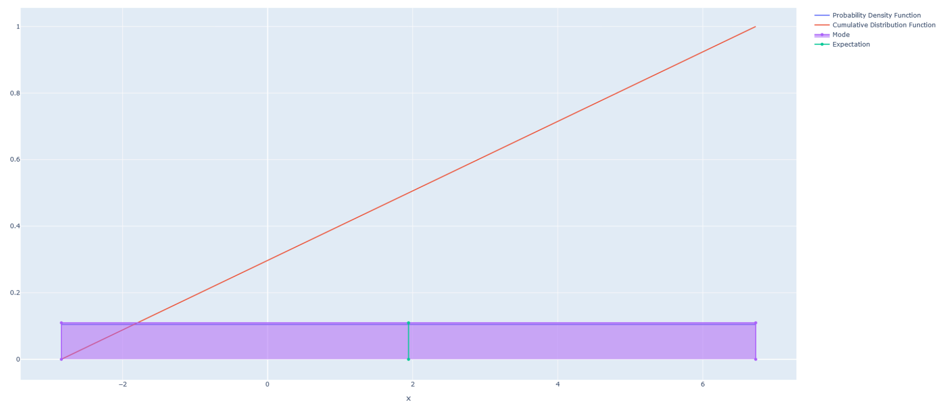

Figure 2: Example of a Nyga Distribution without any splits.

The first best split is then calculated following the previously explained method. The result is shown in Figure 3.

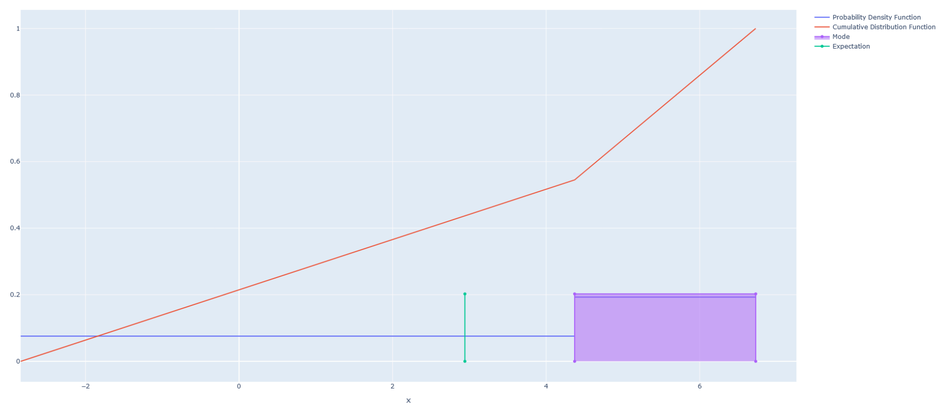

Figure 3: Example of a Nyga Distribution with one split.

Subsequent splits are generated for each subset, left and right of the current split, as long as the improvement exceeds the user-defined threshold, as shown in Figure 4.

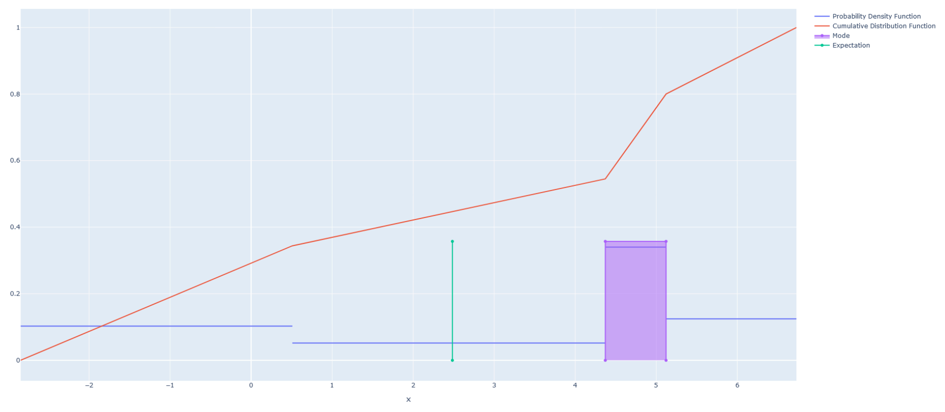

Figure 4: Example of a Nyga Distribution with three splits.

The splitting continues iteratively until the improvement falls below the threshold ($\varepsilon = 0.01$), as seen in Figure 5

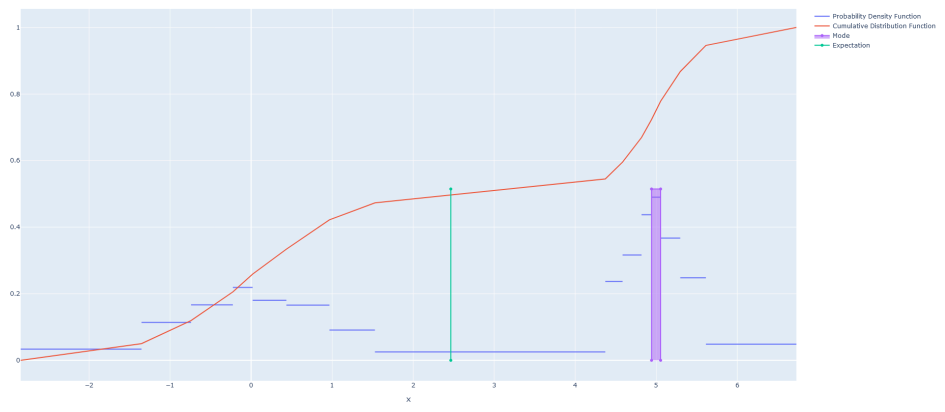

Figure 5: Example of a Nyga Distribution with many splits.

Figure 5 naturally marks a good stopping point for this example. However, if the minimum improvement threshold is set too low, the process can overfit the data, where the leaf distribution's usefulness diminishes, as in Figure 6.

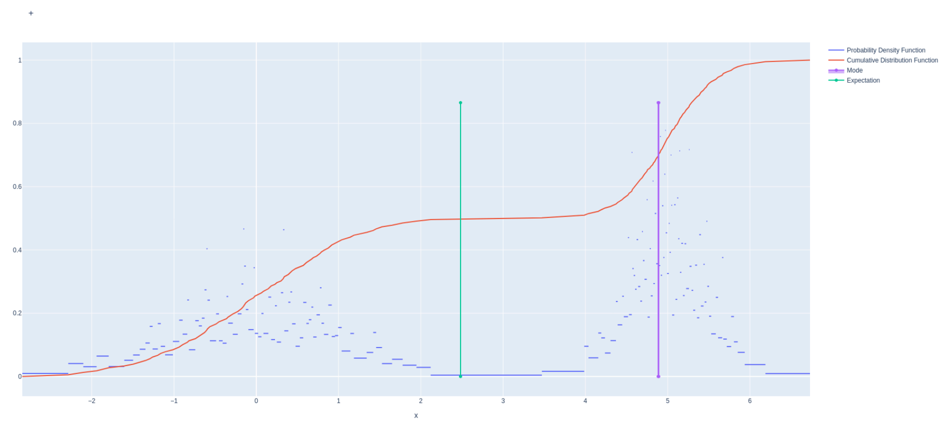

Figure 6: Example of an overfitted Nyga Distribution.

4.2 Joint Probability Tree

Joint Probability Trees (JPTs) are a class of non-parametric, multivariate, smooth, deterministic, and decomposable distributions. They are decision trees where each variable is associated with a Nyga distribution. The learning process of JPTs is based on the C4.5 algorithm, which optimizes mutual information gain by partitioning the dataset into subsets. Each potential split is evaluated using impurity metrics such as variance and Gini impurity [Nyg+23]. The split that results with the greatest improvement to the base dataset's impurity is selected. This process continues until the tree reaches its final state, where the leaves are one simple distribution over the split data. The structure of a JPT is in the scope of PCs. As will be demonstrated later, shallow circuits are structurally close to JPTs, suggesting their potential equivalence in certain scenarios. A visual example is shown in Figure 7.

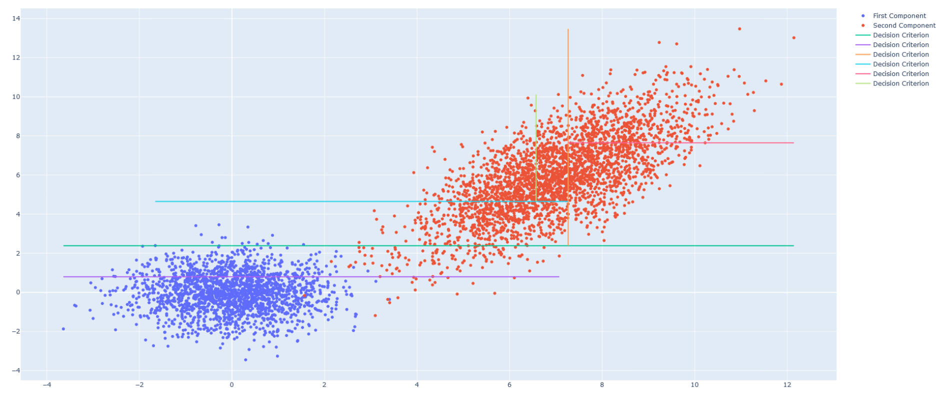

Figure 7: Example of a JPT over a Gaussian mixture with two components [Scha]. The decision criteria are the learned splits from the tree.

5 Queries

Definition 5.1 (Tractable probabilistic inference [CVB20]): A class of queries $\mathbf{Q}$ is **tractable** on a family of probabilistic models M *iff* any query $q \in \mathbf{Q}$ on a model $m \in M$ can be computed in time $\mathcal{O}(\text{poly}(|m|))$. We also say that M is a Tractable model for $\mathbf{Q}$.

As formally defined in Definition 5.1 a class of queries Q is considered tractable on a family of probabilistic models M if any query $q \in Q$ on a model $m \in M$ can be computed in polynomial time with respect to the model size. This notion of tractability offers two key guarantees.

Firstly, queries within a tractable class are guaranteed to produce exact results. This eliminates the need for approximation methods that introduce potentially critical errors into the reasoning process. The leaf distributions used in the model must not violate this guarantee, to ensure this for the complete PCs.

The second guarantee of tractability lies in the computational efficiency of answering queries. For any given tractable query class, the model computes the answer within a time bound proportionally to the size of the model, expressed in polynomial terms. This is given for all queries inside the query class.

5.1 Likelihood

Within the context of a probabilistic model, the likelihood refers to the concept of calculating the probability, denoted as $p(x)$, of a specific complete world state x occurring. The likelihood function estimates the probability of this specific world state materializing.

5.2 Marginals

Definition 5.2.1 (Marginal Query class [CVB20]): Let $p(X)$ be a joint distribution over random variables X. The class of marginal queries over $\mathbf{p}$ is the set of functions that compute

$$p(E = e, Z \in I) = \int_I p(z, e) dZ.$$

where $e \in \text{dom}(E)$ is a partial state for any subset of random variables $E \subseteq X$, and $Z = X \setminus E$ is the set of k random variables to be integrated over intervals $I = I_1 \times \ldots \times I_k$ each of which is defined over the domain of its corresponding random variables in $Z : I_i \subseteq \text{dom}(Z_i)$ for $i = 1, \ldots, k$.

The concept of marginal queries represents a general framework for reasoning about probabilistic models. It leverages the notion of marginalization, a technique for integrating out RVs that are not directly relevant to the query at hand. By marginalizing over unwanted RVs, a marginal distribution that focuses solely on the variables of interest is obtained. This makes it possible to extract knowledge specific to these chosen RVs. One key application of marginal queries is to calculate the likelihood of a specific characteristic emerging within the overall model. By marginalizing over all other RVs, the calculation can be concentrated on the specific

RV representing a chosen characteristic and compute its probability distribution. This provides valuable insights into the model's behavior concerning that particular characteristic.

5.3 Monte Carlo estimator

Computing marginal distributions, which involve integrating over random variables can be computationally expensive for non-tractable models. The integration process itself can be mathematically challenging and time-consuming. An alternative is approximation which exchanges precision for speed. One such approximate inference technique is the Monte Carlo (MC) estimate.

Definition 5.3.1 (Monte Carlo estimate [Scha]): Consider k independent samples $x_1, ..., x_k$ from a multidimensional random variable with a certain pdf $p(x)$. Then a Monte Carlo estimate would be a way of approximating multidimensional integrals of the form

$$\int f(x)p(x)dx$$

by using the empirical expectation of the function f evaluated at the samples

$$\int f(x)p(x)dx \approx \frac{1}{k}\sum_{i=1}^{k} f(x_i).$$

The Monte Carlo estimate is sometimes referred to as the expectation of the function f under the distribution p.

The MC estimate leverages sampling techniques to approximate the marginal distribution of an RV. This works by drawing samples from the joint distribution of all RVs in the model, then averaging the feature of interest (query) over the samples to obtain an estimate of the integral. This approach is computationally more tractable than exact integration, especially for high-dimensional models.

In essence, MC estimation provides an alternative to exact integration when dealing with intricate models. While it may not always yield the perfect answer, it offers a practical solution for obtaining approximate marginal distributions in computationally demanding scenarios.

The variance of an MC estimator is given by Definition 5.3.2. Hence, the expected deviation from the true value of an integral approximated by an MC estimator is dependent on the variance of the feature function f and the square root of the number of samples drawn.

Definition 5.3.2 (Variance of the Monte Carlo estimator [Hen20].): The variance of a Monte Carlo estimator is given by

$$\operatorname{Var}\left(\frac{1}{k}\sum_{i=1}^{k} f(x_i)\right) = \frac{\operatorname{Var}(f(x))}{\sqrt{k}}.$$

5.4 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence is a well-established method for comparing two probability distributions. It quantifies the unlikeness between two distributions p and q.

Definition 5.4.1 (Kullback-Leibler divergence [HO07]): The KL-divergence, also known as the relative entropy, between two probability density functions $f(x)$ and $g(x)$,

$$D(f\|g) = \int f(x) \ln\left(\frac{f(x)}{g(x)}\right) dx$$

One unfortunate attribute of KL divergence is its asymmetry. This means the KL divergence of p from q $(D_{KL}(p\|q))$ is not necessarily the same as the KL divergence of q from p $(D_{KL}(q\|p))$. A critical interpretation of the KL divergence value is obtained at $D_{KL}(p\|q) = 0$. This value indicates that the two distributions are identical, meaning $p(x) = q(x)$ for all possible events x. Conversely, nonzero KL divergence values signify a degree of difference between the distributions. However, it's crucial to remember that KL divergence outputs for different pairs of distributions cannot be directly compared due to the inherent asymmetry of the measure. Additionally, the logarithmic nature of KL divergence presents challenges when dealing with probabilities of zero. If either $p(x)$ or $q(x)$ is zero, the result becomes undefined. Furthermore, if one probability approaches zero, it can lead to inaccurate results. This is because the logarithm of a very small number can be significantly negative, potentially dominating the overall KL divergence calculation.

6 Shallow Circuit

A shallow representation of a circuit is essentially a flattened version of the original structure. This simplification facilitates the computation of certain metrics, such as the L_1 metric as used in this thesis. [CVB20] formally describes a shallow circuit in Definition 6.1.

Definition 6.1 (Shallow Circuit [CVB20]):
A shallow circuit is a PC S of the form

$$S(x) = \sum_{i=1}^{K} \theta_i \prod_{j=1}^{M_i} L_{ij}(x),$$

i. e a PC that has a sum unit as root, followed by a layer of product units and a final layer of input distributions.

Structurally, a shallow circuit comprises a sum node as its root, followed by a set of weighted product nodes. The leaf nodes of the circuit are connected to the product nodes. This architectural design enables structural similarity for any pair of PCs. While [CVB20] provides a rigorous mathematical formulation of shallow circuits, this thesis presents a pseudocode interpretation of this formula for clarity and accessibility in the subsequent section and implements it in the "probabilistic_model" framework [Scha]. To address the specific challenge of calculating the L_1-Metric, a refinement to the shallow circuit structure is an option. A crucial addition is the differentiation between symbolic and integer leaf nodes. This distinction is essential because symbolic nodes represent variables or parameters, while integer nodes represent concrete values. By differentiating between these node types, it can replace each leaf node with a sum node. Each sum node incorporates a subcircuit consisting of leaf nodes representing every element within the corresponding leaf node's distribution space. This enhanced shallow circuit structure provides us with a reliable base for the calculation of the L_1-Metric.

6.1 Pseudocode Shallow Circuit

Firstly the circuit P needs to be smooth and decomposable. If the root node is not a sum unit, a sum unit should be added as a new root. This root is connected only to the old root with a weight of 1.

Definition 6.1.1 (Pseudocode of a shallow Circuit):

```
function SHALLOWING(pc, node, predecessors)
    ▷ "pc is the probabilistic which being shallowed"
    ▷ "Node is the circuit node in focus"
    ▷ "the predecessors is the node from which the call originated, NULL for root"
    if isInstance(node, leafUnit) then
        ▷ "This means it has to be Distribution meaning a Leaf"
        new_sumUnit ← SUMUNIT()
        new_productUnit ← PRODUCTUNIT()
        pc.add_nodes([new_sumUnit, new_productUnit])
        pc.add_edges([(new_productUnit, node), (new_sumUnit, node, weight=1)])
        if Predecessors $\neq \emptyset$ then
            pc.get_edge(predecessors, node).change_nodes(predecessors, new_sumUnit)
        return
    else if isInstance(node, sumUnit) then
        for successors $\in$ node.successors() do
            ▷ "calling all successors to guarantee the children are in shallow form"
            shallowing(pc, successors, node)
        for successors $\in$ node.successors() do
            for successors_of_successors $\in$ successors.successors() do
                weight ← pc.get_weight_between(node, successors_of_successors)
                pc.add_edge(node, successors_of_successors, weight= weight)
            pc.remove_edge(node, successors)
        return
    else
        ▷ "This can only be a productUnit"
        for suc $\in$ node.successors() do
            ▷ "calling all Successors to guarantee the Children are in shallow form"
            shallowing(pc, successors, node)
        node.change_to_sumUnit()
        combination_list ← {$\forall$ li = node_successors.successors_list}
        for combination $\in$ cartesian_product(combination_list) do
            new_ProductUnit ← PRODUCTUNIT()
            pc.add_node(nPU)
            base_weight ← 1
            for productUnit $\in$ combination do
                product_weight ← pc.get_weight_between(node, productUnit)
                base_weight ← base_weight $\times$ product_weight
                for successors $\in$ productUnit.successors() do
                    weight ← pc.get_weight(productUnit, successors)
                    pc.add_edge(new_productUnit, successors, weigth=weight)
            pc.add_edge(node, new_productUnit,weigth=base_weight)
```

Definition 6.1.1 is for an in-place transformation algorithm designed to convert a PC into a shallow form. This process may introduce isolated node structures that are irrelevant to the circuit's overall functionality and can thus be pruned. The transformation is a recursive function whose operations are contingent on the type of unit being processed. For leaf nodes representing distributions, the algorithm expands the incoming edge into a shallow structure. Specifically, the edge is replaced by a sum unit preceding a product unit with a weight of 1. The sole successor of this product unit is the focused leaf node. When the focused unit is a sum unit, the underlying product units of its successors become direct successor units. The edge weights of these units are multiplied to form the new edge weight. The most complex case arises when the focused unit is a product unit. A sum unit is initially placed at the position of the focused product unit. Subsequently, for each successor of the focused unit, one underlying product unit is selected. These grouped product units, organized under a sum unit, constitute the new successors of the focused unit. The weight of this new successor is computed as the product of the weights of the selected product nodes. This process iterates over all possible combinations where exactly one product unit is chosen from each sum unit under the focused unit, akin to a Cartesian product. Upon completion of the transformation at the root node, the circuit exhibits a shallow structure.

The complexity of the shallowing algorithm is predominantly influenced by the node types and their respective processing costs. While every node is visited exactly once, the computational effort varies significantly. Leaf nodes, being the simplest, have constant time complexity as they require only basic expansion. Sum nodes, on the other hand, exhibit linear complexity relative to the total number of their children's children. This linear relationship arises from the need to iterate over each child's children to calculate the sum. The most computationally demanding node type is the product node. This node necessitates the generation of all possible Cartesian product combinations of its children and their descendants. As the number of children and their descendants grows, the complexity of this operation escalates exponentially. Consequently, the overall complexity of the shallowing algorithm is primarily determined by the distribution of product nodes within the PCs and the size of their subcircuit.

6.2 Shallowing Example

This section explores an example to show the fundamental steps involved in the shallowing process. The shallowing algorithm operates in a manner akin to a depth-first search, traversing the nodes of the circuit systematically. To initiate the shallowing process, the algorithm starts with the base circuit which serves as the starting point for our transformation. This base circuit will undergo a series of modifications to achieve its shallower form.

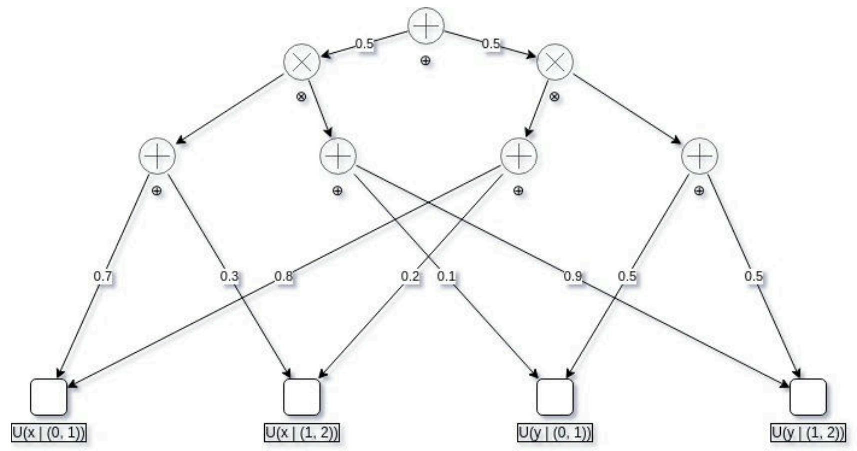

Figure 8: Example PC before applying the shallowing transformation.

Upon finding a leaf node the algorithm will transform it into a shallow circuit state. To facilitate this transformation, the edge leading to the leaf node is expanded. This expansion results in the creation of two new nodes: a sum unit and a product unit. The predecessor edge, which previously connected to the leaf node, is then redirected to the sum unit. This establishes a connection between the sum unit and the leaf node. Additionally, a new edge with a weight of 1 is introduced between the sum unit and the product unit. Finally, the product unit is connected to the leaf node. Through these modifications, the leaf node is transformed into a shallow circuit.

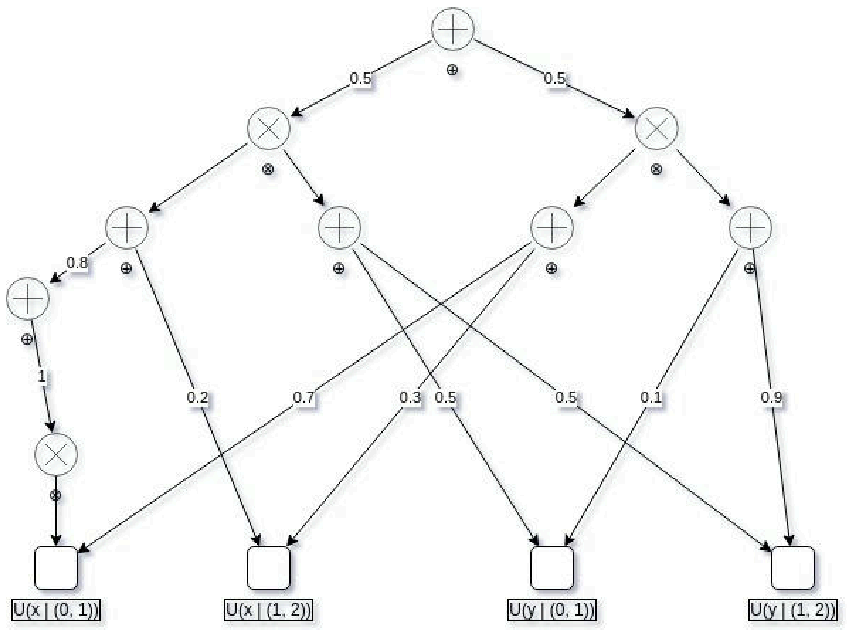

Figure 9: Expansion of the first leaf node in the shallowing transformation.

Following the transformation of the initial leaf node into a shallow circuit, the recursion process returns to the predecessor node. This still consistent with a depth-first search traversal. In this scenario, the discovered node is again a leaf distribution unit. Consequently the same transformation process applied to the initial leaf node is repeated for this newly identified leaf.

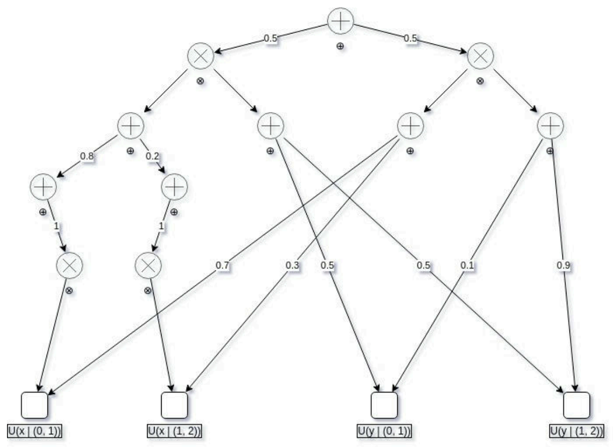

Figure 10: Expansion of the second leaf node in the shallowing transformation.

Once all child sum units are in the shallow form, the sum unit itself can now be transitioned into shallow form. This involves taking the product units of its children and directly attaching them as its own children. The edge weights of these new connections are calculated by multiplying the weights of the paths leading to the product units. In our example, this results in two edges with weights of $1 \cdot 0.8$ and $1 \cdot 0.2$. If any nodes lack predecessors, newly created nodes without predecessors are considered irrelevant to the circuit.

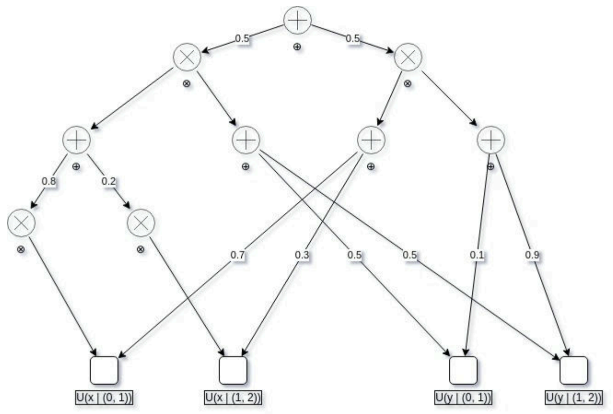

Figure 11: Expansion of the first sum node in the shallowing transformation.

Similarly, the second sum unit, identified through depth-first search logic, undergoes a transformation. Initially, the leaf edges are modified as previously described, followed by the transformation of the sum unit itself.

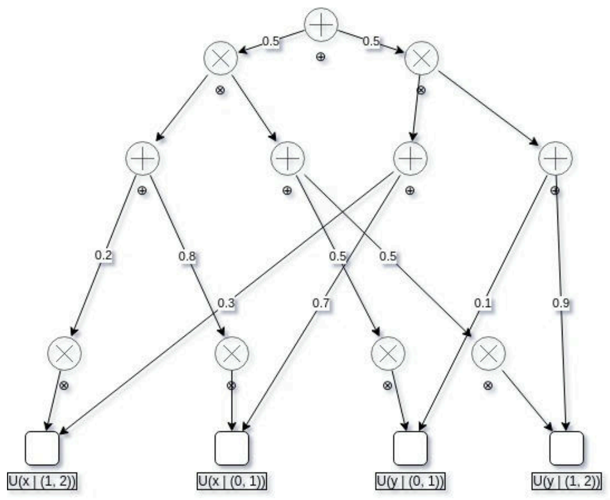

Figure 12: Expansion of the second sum node in the shallowing transformation.

Product units represent the most computationally demanding aspect of shallowing. To address this, a product unit is initially swapped with a sum unit. Subsequently, one product unit is selected from each child of the sum unit. The new product unit is positioned under the sum unit and inherits the children of the chosen product units. The edge weight connecting the sum unit to the new product unit is calculated by multiplying the weights of all paths originating from the utilized product units. This repeats for all possible combinations. In this example, this process results in four edges with weights of $0.1 \cdot 0.5$, $0.8 \cdot 0.5$, $0.1 \cdot 0.5$, and $0.8 \cdot 0.5$.

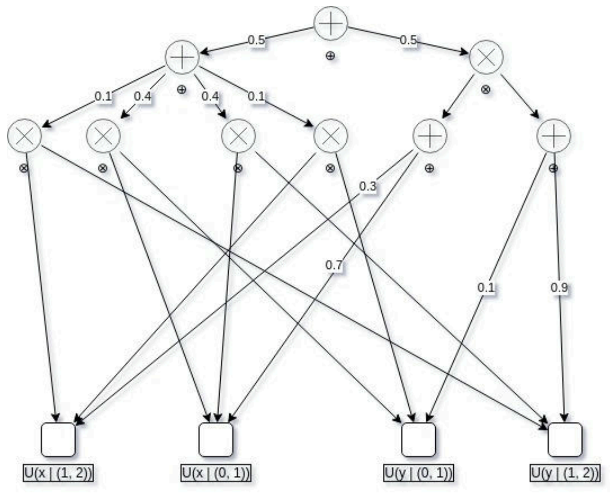

Figure 13: Expansion of the first product node in the shallowing transformation.

The same steps are repeated for the other part of the circuit. However, this explanation will skip the steps leading up to product unit shallowing for this section, focusing directly on the product unit transformation again. This means it will bypass the leaf and sum unit transformations that precede it.

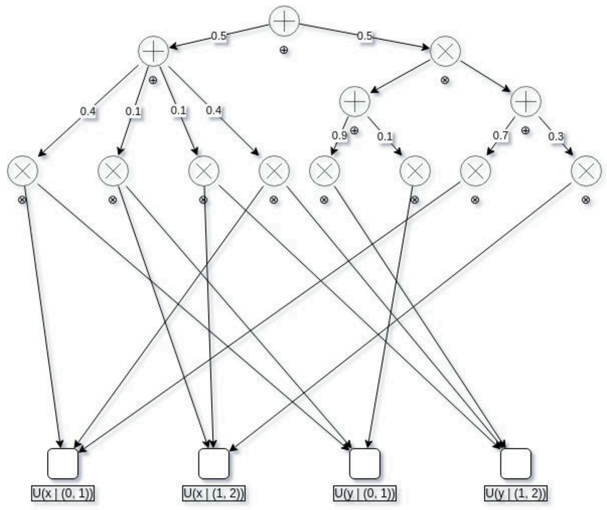

Figure 14: Expansion of the first product node with two sum nodes in the shallowing transformation.

Once more, for each child, one product unit is selected. This selection leads to the creation of a new product unit that replicates the children of the chosen product units. The weight of this new product unit is computed by multiplying the weights of the paths associated with all the chosen product units. This process will be repeated for every possible combination, ensuring a comprehensive exploration of shallowing possibilities.

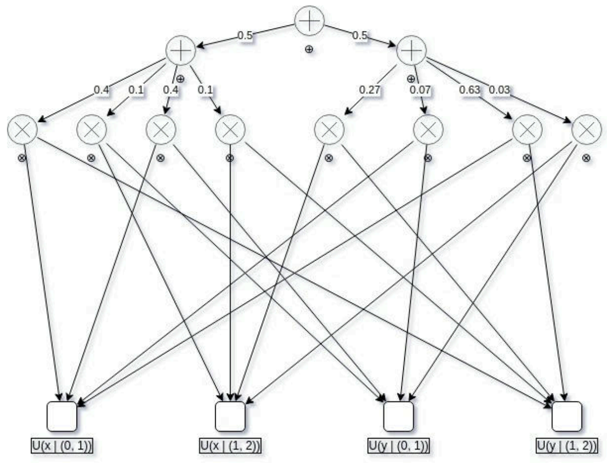

Figure 15: Expansion of the second product node in the shallowing transformation.

Finally, the root sum unit is the only remaining element to be shallowed. The process for shallowing a sum unit remains consistent. The root directly adopts all product units from its children. The edge weights between the root and these product units are calculated by multiplying the weights of the paths leading to the respective product units.

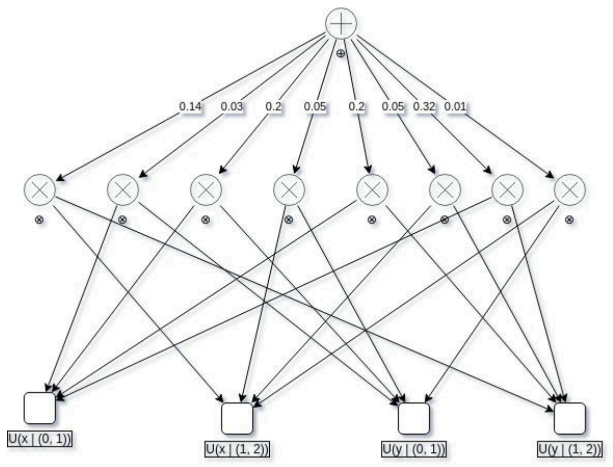

Figure 16: Shallow circuit representation of Figure 8.

With this, the circuit has been successfully shallowed. While this example may appear straightforward, it is important to note that the computational complexity of this algorithm can increase exponentially with larger circuits, deeper structures, or more complex relationships. In such cases, the process of shallowing can be significantly more intensive.

7 Events

Probabilistic models provide a framework for quantifying uncertainty in a system. At their core, these models are built upon the concept of a sample space, which encompasses all possible outcomes of a given experiment or observation. Each individual outcome is considered to be an element of this sample space.

"A sigma algebra (σ-algebra) is a set of sets that contains all set differences that can be constructed by combining arbitrary subsets of the said set. Furthermore, it contains all countable unions of sets and all infinite intersections of the set."

— [Schb] by Tom Schierenbeck

Definition 7.1 (sigma algebra [Schb]):
Let E be a space of elementary events. Consider the powerset 2^E and let $\jmath \subset 2^E$ be a set of subsets of E. Elements of $\jmath$ are called random events. If $\jmath$ satisfies the following properties, it is called sigma-algebra (σ-algebra).

1. $E \in \jmath$
2. $(A, B) \in \jmath \Rightarrow (A - B) \in \jmath$
3. $(A_1, A_2, \ldots \in \jmath) \Rightarrow \left(\cup_{i=1}^{\mathbb{N}} A_i \in \jmath \land \cap_{i=1}^{\infty} A_i \in \jmath\right)$

Events in a probabilistic model are simply elements of the sigma algebra. These events can be used to query the model, obtaining probabilities associated with their occurrence. Probabilistic models can be categorized into two main types: symbolic and continuous. Symbolic models deal with discrete events, often represented by logical expressions or symbols. Continuous models, on the other hand, involve events defined over intervals or ranges of values.

Definition 7.2 (Interval [Schb]):
A simple interval is a subset of $\mathbb{R}$ denoted by

$$\begin{aligned}(a,b) &= \{x \in \mathbb{R} \mid a < x < b\},\\ [a,b) &= \{x \in \mathbb{R} \mid a \le x < b\},\\ (a,b] &= \{x \in \mathbb{R} \mid a < x \le b\},\\ [a,b] &= \{x \in \mathbb{R} \mid a \le x \le b\},\end{aligned}$$

A composite interval, or just interval, is a union of simple intervals.

$$\imath = \imath_1 \cup \imath_2 \cup \dots \cup \imath_n$$

Continuous events can be used to perform basic set operations like intersection, union, and difference. This allows for the construction of more complex events from simpler ones. An event can either contain symbolic element or a number range. Probabilistic models are constructed from RVs, each equipped with its own sigma-algebra and associated support space. When multiple RVs are integrated into a single model, a product sigma-algebra is employed to combine their individual sample spaces. This product sigma-algebra ensures a comprehensive representation of the joint probability space, enabling the analysis of complex probabilistic relationships between the constituent RVs.

Definition 7.3 (Product Sigma Algebra [Hun11]):
Let$(E_1, \jmath_1)$ and $(E_2, \jmath_2)$ be measurable spaces. The product sigma-algebra of $\jmath_1$ and $\jmath_2$ is denoted $\jmath_1 \otimes \jmath_2$, and defined as: $\jmath_1 \otimes \jmath_2 := \sigma(\{S_1 \times S_2 : S_1 \in \jmath_1 \wedge S_2 \in \jmath_2\})$ where σ denotes generated sigma-algebra and $\times$ denotes Cartesian product. This is a sigma-algebra on the Cartesioan product $E_1 \times E_2$

By leveraging the properties of the sigma algebra, events can encompass any portion of the model's sample space. This allows for the usage of meaningful queries about the model to obtain relevant probabilistic information.

7.1 Indicator as Event

An indicator function serves as a valuable tool to represent the occurrence or non-occurrence, in this instance, of an event.

Definition 7.1.1 (Indicator function for Events):
Let A be an set. The indicator $\mathbb{1}_A$ describes the function

$$\mathbb{1}_A := \begin{cases} 1 \text{ if } x \in A \\ 0 \text{ if } x \notin A \end{cases}.$$

This function effectively indicates whether an element x is a contained in an event A. By incorporating indicator functions into calculations, it is possible to selectively include or exclude elements based on their membership in specific events, ensuring that only relevant contributions are considered.

7.1.1 Indicator Expectation and Variance

Indicator functions are also suitable for calculating the probability of an event. Doing so requires the calculation of the expectation of the indicator function in a distribution p resulting in Definition 7.1.1.1.

Definition 7.1.1.1 (Indicator Expectation):
Let: e_p be a Indicator

$$\begin{aligned} E(e_p) &= \int_{x \in e_p} x \cdot p_{e_p}(x) \\ &= 1 \cdot p_{e_p}(1) + 0 \cdot p_{e_p}(0) \\ &= 1 \cdot p(e_p) + 0 \cdot p(e_p) \\ &= P(e_p) \end{aligned}$$

The variance of an indicator function is shown in Definition 7.1.1.2.

Definition 7.1.1.2 (Indicator Variance):
Let: e_p be a indicator. The variance of e_p is given by

$$\begin{aligned} \operatorname{Var}(e_p) &= E\left((e_p)^2\right) - E(e_p)^2 \\ &= E((e_p)) - E(e_p)^2 && \mid 0^n = 0 \text{ and } \ 1^n = 1 \\ &= P(e_p) - P(e_p)^2 && \mid \text{Definition 7.1.1.1} \\ &= P(e_p) \cdot (1 - P(e_p)) \\ &< 1 \end{aligned}$$

Definition 7.1.1.2 shows that any possible value combination that $p(e_q)$ and $q(e_q)$ can express will always result in a value < 1. Hence, the variance of the MC estimator of an indicator function is small.

8 L_1-metric

The L_p-metric is a well-established metric for quantifying the distance between probability distributions defined over the same RVs. For a countable sample space, the L_p-metric between densities is defined in Definition 8.1:

Definition 8.1 (L_p-metric between densities [Ele09]):
The L_p-metric between densities is a metric on P (for a countable X), defined, for any $p \geq 1$, by

$$\left(\int_x |p_1(x) - p_2(x)|^p\right)^{\frac{1}{p}}$$

In this thesis, the focus is on just the L_1-metric, which simplifies to:

$$p = p_1$$
$$q = p_2$$

$$\int_x |p(x) - q(x)|$$

This metric provides a clear and interpretable distance measure, with 0 indicating identical distributions and 2 representing completely disagreeing distributions. The result can be divided by 2 to achieve the intuitive 0 to 1 range. Compared to other divergence measures such as the Kullback-Leibler divergence, the L_1-metric yields more intuitive results. This can be seen in the result range, which is from 0 to inf and possibly undefined. Although its computational complexity can be high for distributions with complex PDFs. The PCs considered in this thesis employ uniform distributions in their leaves, mitigating this computational burden.

8.1 L_1-metric in Probabilistic Circuits

The L_1-metric can be effectively applied to shallow circuits $p(x)$ and $q(x)$ that share the same random variables. This compatibility arises from the inherent properties of shallow circuits, namely smoothness and decomposability, which align with the requirements of the L_1-metric employed here. The crux of this work lies in proving the validity of applying the L_1-metric to shallow circuits. This is achieved by directly substituting the shallow circuits into the metric formula:

Definition 8.1.1 (L_1-metric in PC):
Let $(E, \mathfrak{I}, P)$ and $(E, \mathfrak{I}, Q)$ be two probability spaces with density p and q.
Let $E_p = \{x \mid p(x) > q(x)\}$
Let $E_q = E_p^c$

$$\begin{aligned}
L_1(p,q) &= \int |p(x) - q(x)| dx \\
&= \int \mathbb{1}_{E_p}(p(x) - q(x)) + \mathbb{1}_{E_q}(q(x) - p(x)) dx \mid \text{split into disjunct parts} \\
&= \int \mathbb{1}_{E_p}(p(x) - q(x)) dx + \int \mathbb{1}_{E_q}(q(x) - p(x)) dx \\
&= \int \mathbb{1}_{E_p} p(x) - \mathbb{1}_{E_p} q(x) dx + \int \mathbb{1}_{E_q} q(x) - \mathbb{1}_{E_q} p(x) dx \\
&= \int \mathbb{1}_{E_p} p(x) dx - \int \mathbb{1}_{E_p} q(x) dx + \int \mathbb{1}_{E_q} q(x) dx - \int \mathbb{1}_{E_q} p(x) dx \\
&= P(E_p) - Q(E_p) + Q(E_q) - P(E_q) \\
&= P(E_p) - P(E_q) + Q(E_q) - Q(E_p) \\
&= P(E_p) - (1 - P(E_p)) + (1 - Q(E_p)) - Q(E_p) \\
&= P(E_p) - 1 + P(E_p) + 1 - Q(E_p) - Q(E_p) \\
&= 2P(E_p) - 2Q(E_p) \\
&= 2(P(E_p) - Q(E_p))
\end{aligned}$$

Given a pair of shallow circuits, this approach computes the distance between them. The core challenge lies in generating the event E_p. With it the L_1-metric can be calculated. To represent the points in the event, there needs to be a convention for each type of distributions. For symbolic distributions, enumerating points in the support suffices. For uniform distributions, select the midpoint of each interval in the support. This choice is arbitrary, as any point within the interval yields the same probability. Given the event, compute the probability of each model and calculate the distance between them. This thesis only handles the scope of uniform/ symbolic leaves. While this quantity may be calculated for other families of distributions, it is out of the scope of this work.

8.2 L_1 as Pseudo-Code

This section delves into the practical application of the L_1-metric within the context of PCs. Leveraging the previously introduced shallow circuit architecture as input, this demonstrates the utility of the L_1-metric through a concrete example in the subsequent pseudocode.

Definition 8.2.1 (L_1 in Pseudo-Code):

```
function L1(p, q, tolerance)
    ▷ “p and q are shallow circuits”
    p_event $\leftarrow$ events_of_higher_density(p, q, tolerance)
    distance $\leftarrow$ $2 * (P(\text{p\_event}) - Q(\text{p\_event}))$
    return distance
function EVENTS_OF_HIGHER_DENSITY(p, q, tolerance)
    p_result $\leftarrow$ p.support() - q.support()
    for p_product_unit $\in$ p.successors() do
        for q_product_unit $\in$ q.successors() do
            intersection $\leftarrow$ p_product_unit.support().union(q_product_unit.support())
            if intersection $\neq \emptyset$ then
                center_points $\leftarrow$ intersection.get_centerponints()
                p_likelihood $\leftarrow$ p.likelihood(center_points)
                q_likelihood $\leftarrow$ q.likelihood(center_points)
                diff_likelihood $\leftarrow$ p_likelihood - q_likelihood
                if diff_likelihood $>$ tolerance then
                    p_result $\leftarrow$ p_result.union(intersection)
```

Definition 8.2.2 (L_1 in Psssssssssseudo-Code):

```
function $L_1$(p, q, tolerance)
    ▷ “p and q are shallow circuits”
    p_event $\leftarrow$ events_of_higher_density(p, q, tolerance)
    distance $\leftarrow$ $2 * (P(\text{p\_event}) - Q(\text{p\_event}))$
    return distance
function EVENTS_OF_HIGHER_DENSITY(p, q, tolerance)
    p_result $\leftarrow$ p.support() / q.support()
    for p_product_unit $\in$ p.successors() do
        for q_product_unit $\in$ q.successors() do
            intersection $\leftarrow$ p_product_unit.support() $\cap$ (q_product_unit.support())
            if intersection $\neq \emptyset$ then
                center_points $\leftarrow$ intersection.get_centerponints()
                p_likelihood $\leftarrow$ p.likelihood(center_points)
                q_likelihood $\leftarrow$ q.likelihood(center_points)
                diff_likelihood $\leftarrow$ p_likelihood - q_likelihood
                if diff_likelihood $>$ tolerance then
                    p_result $\leftarrow$ p_result $\cup$ (intersection)
```

The computational bottleneck in the construction of events for the L_1-metric lies in the exhaustive enumeration of all possible distribution combinations in a shallow circuit. This pseudocode capitalizes on the structure of the circuit to make this enumeration feasible.

However, the computational complexity grows exponentially with the number of leaves and the number of product units in the circuit grows quadratically with the number of product nodes. One reason for this is the need to calculate the supports of the units. This approach is only possible because the discrete distributions were split up before shallowing. Additionally, since the p_{event} is consistent of $p(x)$ greater $q(x)$ for x, a tolerance is necessary to account for computational deviations near zero when p_likelihood = q_likelihood. In the following passage, this thesis will go into a MC approximation for the L_1-metric that avoids the shallow circuit requirement altogether.

8.3 Monte Carlo L_1-metric

Due to the potential complexity of a PC, the transformation to shallow circuits and the subsequent L_1-metric calculation can be computationally expensive. To accelerate this process, the trade-off between precision and speed by introducing an approximation is to be explored. A minor loss in precision is acceptable, as it maintains the comparability of distances within the L_1-metric. The MC method to approximate the generation of events is required for the L_1 -metric algorithm. Importantly, this approach eliminates the need for constructing shallow circuits entirely. The following section details how the L_1-metric can be approximated with MC estimation.

Definition 8.3.1 (L_1-metric as Monte Carlo):

$$\begin{aligned}
L_1(p,q) &= 2P(E_p) - 2Q(E_p) \\
&= 2(P(E_p) - Q(E_p)) \mid \text{insert Monte Carlo Estimator Definition 5.3.1} \\
&\approx 2\frac{\int \mathbb{1}_{(p(x)>q(x))} p(x)dx - \int \mathbb{1}_{p(x)>q(x)} q(x)dx}{\text{sampel\_size}}
\end{aligned}$$

The MC approximation of the event involves sampling from both circuits, with the total number of samples from both circuits constituting the sample size. Employing the same indicator variables as in the L_1-metric, the occurrences of E_p and E_q within the sample are summed. The differences can then be calculated from their occurrences. The final approximation is computed by dividing the sum of these event counts by the sample size.

There is one more variation for the MC estimator in this context:

Definition 8.3.2 (L_1-metric in a PC as Monte Carlo estimator by Alessandro Santonicola and Tom Schierenbeck):

$$\text{Let } r = U_{\text{supp}(p) \cup \text{ supp}(q)}.$$

$$\begin{aligned}
E_r(|p(x) - q(x)|) &= \int |p(x) - q(x)| \ r(x) dx \\
L_1(p, q) &= \int |p(x) - q(x)| \ dx \\
&= \int |p(x) - q(x)| \ \frac{r(x)}{r(x)} dx \\
&= \frac{\int |p(x) - q(x)| \ r(x) dx}{r(x)} \\
&= \frac{E_r(|p(x) - q(x)|)}{r(x)}
\end{aligned}$$

The MC estimator proposed by Alessandro Santonicola and Tom Schierenbeck introduces a novel approach by incorporating a relation and dependency between the combined support of p and q. While initially it might seem that this estimator employs rejection sampling, this assumption is incorrect. The estimator actually samples from the union of p and q, necessitating a division by the total union limitation to ensure that the result is not dependent on any specific distribution and remains a metric assumption. All samples are in the space of the models, and thus there is no rejection taking place.

Definition 8.3.3 (Variance of MC-L_1):

$$\begin{aligned}
\operatorname{Var}(2(P(E_p) - Q(E_p))) &= 4(\operatorname{Var}(P(E_p)) - \operatorname{Var}(Q(E_p))) \\
&\text{Check Definition 7.1.1.2} \\
&= \underbrace{4\left(\cdot \ P(E_p) 1 - \dot{P}(E_p) - Q(E_p) 1 - \dot{Q}(E_p)\right)}_{\text{max if } P(E_p)=0.5 \text{ and } Q(E_q)=0} \\
&\leq 2
\end{aligned}$$

The MC-L_1 variance shows that the variance is bounded, thus the MC estimate will estimate the L_1 value accurately, even within a manageable sample size.

9 Evaluation

To evaluate the performance of the MC estimator in the L_1-metric, an experiment using models trained on a dataset for robotic arm movement is conducted. These models were taught using the JPT learning method. The time calculations were performed on a virtual machine equipped with 6 CPU threads, powered by an AMD Ryzen 7 7700X processor with a clock rate of 4.5GHz (boosting up to 5.4GHz). Furthermore, the result was divided by two as stated to create a human intuitive range of 0 to 1.

Two models were trained on the same dataset, with the first using a minimum of 800 samples per quantile and 0.01 samples for the leaf, while the second employed 600 samples per quantile and 0.1 samples for the leaf. This results in two PCs with 519 nodes and 518 edges, and 56 nodes and 55 edges respectively. The L_1-metric calculated a distance between these two models of 0.4215562. This shows a significant difference in the belief of the two models. The MC estimator then estimated this value over multiple samples.

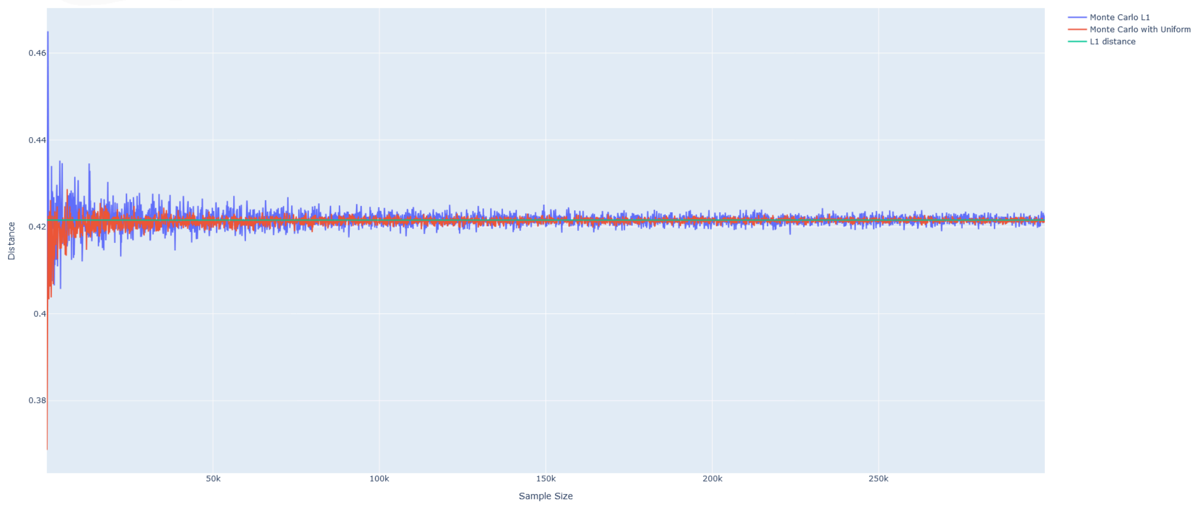

Figure 17: Monte Carlo estimator of the L_1 metric between a JPT with 56 nodes and a PC with 519 nodes learned from the same dataset.

The estimators demonstrated a good approximation of the original value even with low sample sizes, indicating that their variance is quite low. This highlights the high precision of the MC estimate. The L_1-metric, while offering absolute precision, poses significant computational challenges for many models. Even with relatively simple models, as illustrated in Figure 17, the L_1 estimator demands substantially more time. This computational burden becomes even more pronounced for complex models. This also counts for the calculation of the uniform MC estimator. For the following additional example, only the MC estimator is given, because the calculation for the exact L_1-metric and the combined support took an infeasible amount of time (>24 hours) on the example computer. Now with the utilizing of toy datasets available in the scikit-learn library [Ped+11], the estimated distances are shown in Figure 18, Figure 19 and Figure 20.

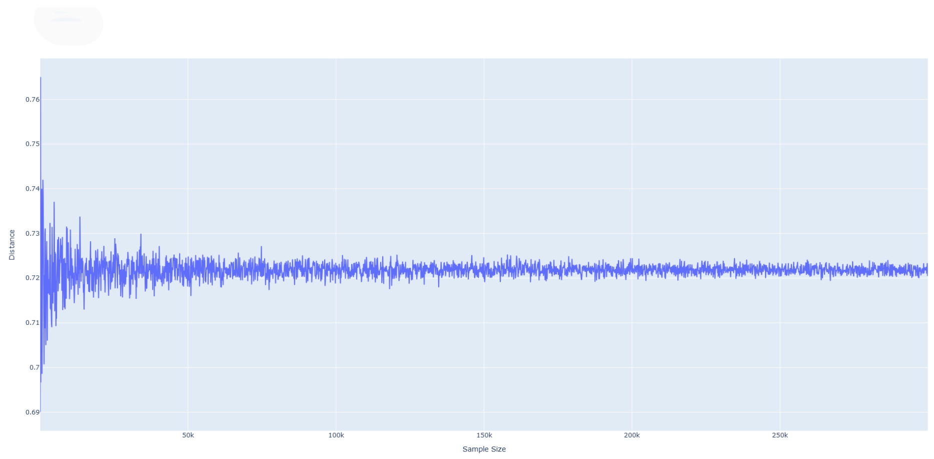

Figure 18: Monte Carlo estimator for L1 metric of two models on the Iris dataset [Ped+11]. The two PCs for had 145 nodes and 144 edges, and 73 nodes and 72 edges.

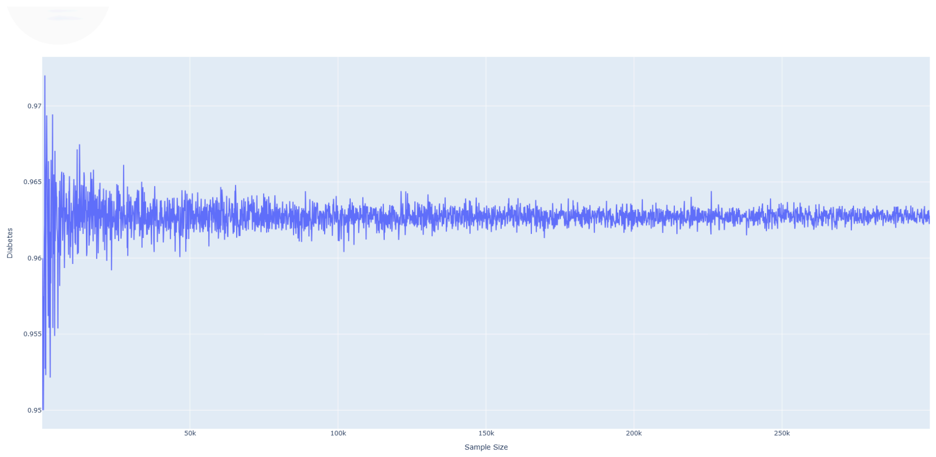

Figure 19: Monte Carlo estimator for L1 metric of two models on the diabetes dataset [Ped+11]. One JPT had 64 nodes and 63 edges, the other JPT had 337 nodes and 336 edges.

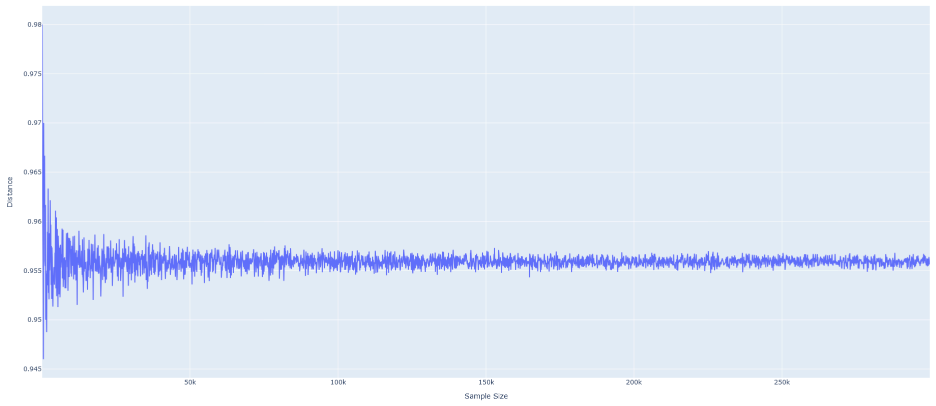

Figure 20: Monte Carlo estimator for L1 metric of two models on the breast cancer dataset [Ped+11]. The JPTs had 208 nodes and 207 edges, and 162 nodes and 161 edges, respectively.

To focus on the computational efficiency of the estimators themselves, Figure 21 initially excludes the L_1-metric and combined support calculation from the analysis.

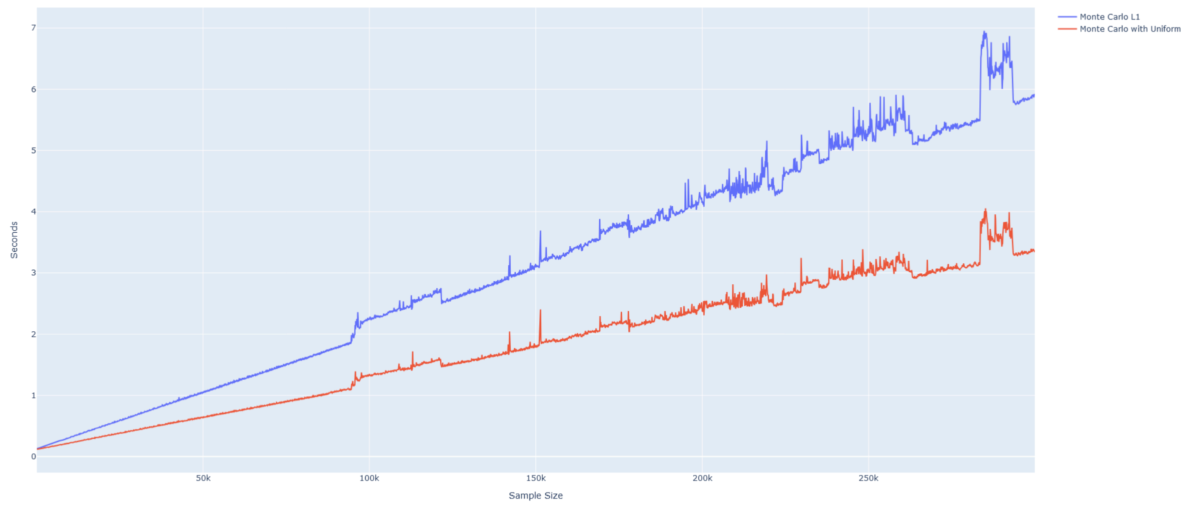

Figure 21: Computation time for the Monte Carlo estimator in Figure 17.

This graph reveals a clear trend: the MC estimator with uniform sampling exhibits a slower growth in computational time. This can be attributed to the fact that the estimator is focused on the support space union. The precalculation of the uniform distribution over the union of the models support spaces achieves the perceived trend. Without precalculation there is a significant increase in the computational time as shown in the following graph.

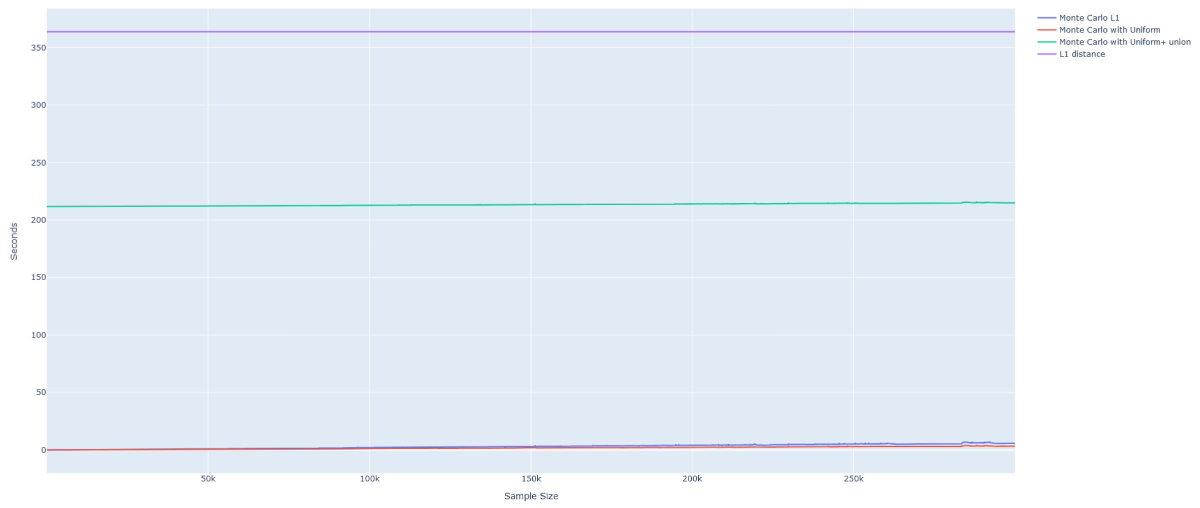

Figure 22: Computation time for the Monte Carlo estimator with the exact L_1 and uniform support approximation in Figure 17.

However, once this precalculation is performed, it can be reused for subsequent estimations, which makes it a one time cost. Which makes this estimator an option in scenarios where the support union is already calculated or repeated executions are needed. Both estimators exhibit notable dips in the computational time graph. These dips are likely due to the underlying code base [Scha] and CPU cycling, which are beyond the scope of this analysis.

9.1 Conclusion

Continuous probabilistic models are powerful architectures to represent a variety of probabilistic models, especially JPTs, as used in this thesis. The L_1-metric provides a comparable and understandable measure for the difference between two models. While the L_1-metric might encounter performance issues in real-world applications involving more complex models, the MC estimator emerges as a viable alternative to approximate the exact L_1-metric output. This approximation can significantly aid in understanding and studying models with the correct requirements. However, this thesis leaves some questions unanswered. One such question pertains to non-finite PDFs like the Gaussian distribution, and all uniform distributions, which are inherently non-finite. Addressing this limitation would extend the applicability of the metric to a wider range of cases. Additionally, the performance of the metric with non-shallow tree-near models remains an open question, as JPTs are relatively close to shallow PCs.

Bibliography

[Sha+22] M. Shaygan, C. Meese, W. Li, X. (George) Zhao, and M. Nejad, "Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities," *Transportation Research Part C: Emerging Technologies*, vol. 145, p. 103921, 2022, doi: https://doi.org/10.1016/j.trc.2022.103921.

[Wan+24] X. Wang, J. Zhao, E. Marostica, and others, "A pathology foundation model for cancer diagnosis and prognosis prediction," *Nature*, vol. 634, pp. 970–978, 2024, doi: 10.1038/s41586-024-07894-z.

[Dec] J. Dech, "PyCRAM: A Python framework for cognition-enbabled robtics." [Online]. Available: https://github.com/cram2/pycram

[CVB20] Y. Choi, A. Vergari, and G. Van den Broeck, "Probabilistic circuits: A unifying framework for tractable probabilistic models," *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, p. 6, 2020.

[Sch] T. Schierenbeck, "probabilistic_models: A Python package for probabilistic models." [Online]. Available: https://github.com/tomsch420/probabilistic_model

[Nyg+23] D. Nyga, M. Picklum, T. Schierenbeck, and M. Beetz, "Joint Probability Trees." [Online]. Available: https://arxiv.org/abs/2302.07167

[Hen20] P. Hennig, "Probabilistic Machine Learning," 2020.

[HO07] J. R. Hershey and P. A. Olsen, "Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, 2007, p. IV–317–IV–320. doi: 10.1109/ICASSP.2007.366913.

[Sch] T. Schierenbeck, "Random-Events." [Online]. Available: https://github.com/tomsch420/random-events

[Hun11] B. Hunter, "Data Mining Compressed, Incomplete and Inaccurate High Dimensional Data," 2011. [Online]. Available: https://www.proquest.com/dissertations-theses/data-mining-compressed-incomplete-inaccurate-high/docview/897905271/se-2

[Ele09] M. M. D. Elena Deza, *Encyclopedia of Distances*. Springer Berlin, Heidelberg, 2009. doi: https://doi.org/10.1007/978-3-642-00234-2.

[Ped+11] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[Kur20] N. Kurt, *Stochastik für Informatiker: Eine Einführung in einheitlich strukturierten Lerneinheiten*, 1st ed. Springer Vieweg, 2020.

[TTN13] T. Thordis, G. Tilmann, and G. Nadine, "Using proper divergence functions to evaluate climate models." [Online]. Available: https://arxiv.org/abs/1301.5927

10 AI acknowledgement

Google's Gimini was used as support for the writing of the initial draft, of this thesis.