Probabilistic Flow Circuits: Towards Unified Deep Models for Tractable Probabilistic Inference

Sahil Sidheekh[1] **Kristian Kersting**[2,3] **Sriraam Natarajan**[1]

[1]Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas
[2]Department of Computer Science , TU Darmstadt
[3] Centre for Cognitive Science, TU Darmstadt, and Hessian Center for AI

Abstract

We consider the problem of increasing the expressivity of probabilistic circuits by augmenting them with the successful generative models of normalizing flows. To this effect, we theoretically establish the requirement of decomposability for such combinations to retain tractability of the learned models. Our model, called Probabilistic Flow Circuits, essentially extends circuits by allowing for normalizing flows at the leaves. Our empirical evaluation clearly establishes the expressivity and tractability of this new class of probabilistic circuits.

1 INTRODUCTION

Building flexible models for representing probability distributions has been a long standing goal in AI. Driven by the need for reliable decision making under uncertainty, the early efforts in this direction were aimed towards building efficient models [Koller and Friedman, 2009] and algorithms [Chow and Liu, 1968, Welch et al., 1995] that enabled tractable probabilistic inference. Several tractable generative models that described probability distributions in the form of graphs were recently developed [Poon and Domingos, 2011, Choi et al., 2020, Rahman et al., 2014, Darwiche, 2003].

In particular, Choi et al. [2020] introduced *Probabilistic Circuits* (PCs) as a unified notion encompassing such models that describe distributions in the form of computational graphs such as cutset networks [Rahman et al., 2014], arithmetic circuits [Darwiche, 2003] and sum-product networks [Poon and Domingos, 2011]. Several subsequent works have shown how to learn (a subset of) PCs [Gens and Pedro, 2013, Peharz et al., 2016, Trapp et al., 2019] and have increased their flexibility [Molina et al., 2018, 2017], reliability [Deratani Mauá et al., 2018] as well as scalability [Peharz et al., 2020b,a, Dang et al., 2022, Liu et al., 2023]. While tractable, these PCs are still less expressive than deep generative neural models (DGMs) such as GANs [Goodfellow et al., 2014], VAEs [Kingma and Welling, 2014] and Normalizing Flows (NFs) [Tabak and Turner, 2013, Papamakarios et al., 2021]. Among these, NFs have recently gained attention due to their use of invertible functions that enables maximum likelihood training, resulting in more stable models that avoid mode collapse and vanishing gradient problems of deep generative models. While powerful, these deep models are not efficient in performing inference tasks that tractable models such as PCs are quite adept at.

So it is not surprising that combinations of PCs with DGMs have been explored. Tan and Peharz [2019] explored the integration of VAEs with PCs, while Trapp et al. [2020], Yu et al. [2021] explored the integration of gaussian processes with PCs. Along similar lines, Correia et al. [2023] recently proposed building continuous mixtures of tractable probabilistic models to improve their representational flexibility. However, the added expressivity of PCs due to these models came at the cost of their tractability. Integrating new transformations within PCs was first explored in Sharir and Shashua [2018], who proposed the addition of *quotient* nodes to improve expressivity, while retaining tractability. Most recently, Pevný et al. [2020] proposed the addition of invertible *transform nodes*, representing normalizing flows. They augmented PCs by placing invertible affine transformations arbitrarily within the circuit, and demonstrated its added modeling flexibility. However, as we prove, this understanding of integrating flows with PCs is **incomplete, and does not guarantee tractability**.

To establish tractability, we first define the notion of **τ-decomposability** that specifically considers the decomposibility of transform nodes. Building upon this definition, we then propose a method for integrating PCs with NFs by defining a new class of circuits, called *Probabilistic Flow Circuits* (PFC). These models follow a similar structure to that of a PC by employing sum and product nodes in the circuit while using NFs at the leaves. The advantage is that since NFs allow for effectively modeling arbitrarily complex

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

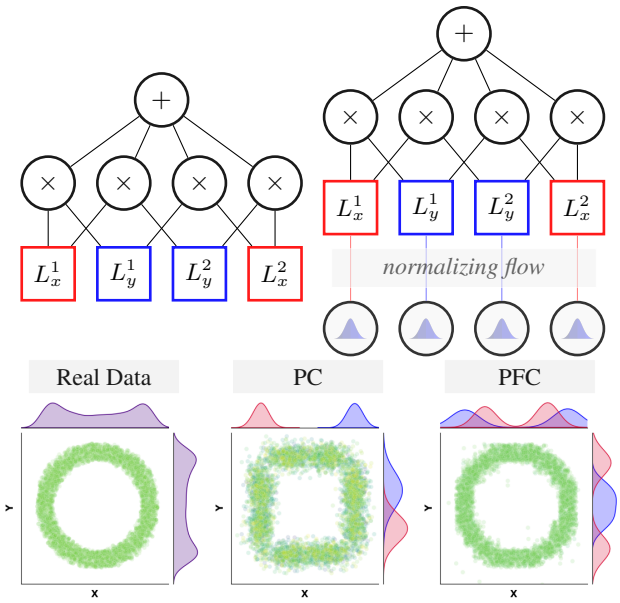


Figure 1: Modeling a 2D data distribution (bottom left) using a PC (top left) and a PFC (top right). The red and blue marginal plots show the distribution captured by the corresponding leaves (L). The PFC, with its multi-modal leaf densities is able to better model the data than the PC.

distributions, the resulting model enhances the modeling capabilities of a PC. As an example, consider the data shown in Figure 1. Using a simple PC that learns a mixture of four distributions, the depicted data distribution cannot be modeled faithfully. However, our method that uses the same structure can better model the data as it learns multimodal distributions at the leaves. The additional advantage is that the tractability of a PC is retained as the NFs are defined in the leaves rather than in the inner nodes.

Overall, we make the following key contributions:

- We present the formalism for combining PCs and NFs to develop probabilistic flow circuits that retain their respective advantages of tractability and flexibility.
- We show that the existing combination that uses invertible affine transformations breaks the decomposability property, and that **τ-decomposability** is a necessary condition for retaining the tractability of PCs.
- We present a recommendation for a family of invertible functions by employing linear rational splines at the leaves of PCs.
- We demonstrate the efficacy and efficiency of the flow circuits empirically in density estimation and sample generation tasks using high-dimensional data.

We proceed as follows: we start off by reviewing probabilistic circuits and normalizing flows. Then we introduce τ-decomposability and the probabilistic flow circuits. Before concluding, we present our empirical evaluation.

2 BACKGROUND

Notation. We use X to denote a random variable, x to denote its value, bold letters to denote sets of random variables i.e., $\boldsymbol{X} = \{X_i\}_{i=1}^d$, $\boldsymbol{x}$ to denote an assignment of values to all variables in $\boldsymbol{X}$, $\mathbf{val}(X)$ to denote the set of all values that X can take, and $\mathbf{val}(\boldsymbol{X})$ to denote the cartesian product $\mathbf{val}(X_1) \times \mathbf{val}(X_2) \ldots \mathbf{val}(X_d)$. As normalizing flows are most commonly defined over continuous spaces, we will consider only continuous random variables. With mild overload of notation, we interchangeably refer to $\boldsymbol{X}$ as a d-dimensional real random vector and $\boldsymbol{x} \in \mathbb{R}^d = (x_1, \ldots, x_d)$ the value it takes. We denote by $\boldsymbol{X}_A \subseteq \boldsymbol{X} = \{X_i\}_{i \in A}$ for $A \subseteq \{i\}_{i=1}^d$ the subset of random variables in $\boldsymbol{X}$ contained in A. Equivalently, we denote by $\boldsymbol{\Pi}_A(\boldsymbol{x}) \in \mathbb{R}^{|A|}$ the projection of the $\boldsymbol{x}$ on A, i.e., $\boldsymbol{x}_A = \boldsymbol{\Pi}_A(\boldsymbol{x}) = (x_i)_{i \in A}$ is the $|A|-$dimensional vector obtained by indexing $\boldsymbol{x}$ along the dimensions contained in A. We use $P(\boldsymbol{X})$ to denote a probability distribution over $\boldsymbol{X}$ and $p(\boldsymbol{x})$ its density.

Tractable Probabilistic Models: A generative model θ that approximates a probability distribution $P(\boldsymbol{X})$ is said to be tractable on a given probabilistic inference task $\boldsymbol{Q}$, where $\boldsymbol{Q} = f(P_\theta(\boldsymbol{X}))$, if $\boldsymbol{Q}$ can be computed in time polynomial in the size of the model. Tractability is thus a spectrum governed by both the nature of the inference tasks (or f) and properties of θ.

Probabilistic Inference Tasks: Utilizing a probabilistic generative model ($P_\theta(\boldsymbol{X})$) to make decisions in real life often requires querying the model to infer properties of the underlying distribution or extracting the uncertainty associated with events defined by its random variables. We elaborate on the prevalent types of inference queries below.

Let $\boldsymbol{X}_e, \boldsymbol{X}_q \subseteq \boldsymbol{X}$ such that $\boldsymbol{X} = \boldsymbol{X}_e \cup \boldsymbol{X}_q$ and $\boldsymbol{X}_e \cap \boldsymbol{X}_q = \emptyset$. First, given a full assignment of values $\boldsymbol{x}$ to all variables in $\boldsymbol{X}$, computing the density $p_\theta(\boldsymbol{X} = \boldsymbol{x})$ is called *evidential inference*. Second, we might be interested in computing the marginal distribution over a subset of variables $\boldsymbol{X}_e$, for instance, $p_\theta(\boldsymbol{X}_e = \boldsymbol{x}_e) = \int_{\boldsymbol{x}_q \in \mathbf{val}(\boldsymbol{X}_q)} p_\theta(\boldsymbol{X}_e = \boldsymbol{x}_e, \boldsymbol{X}_q = \boldsymbol{x}_q)$ which is known as *marginal inference*. Third, we might be interested in computing conditional distributions, for e.g., $p_\theta(\boldsymbol{X}_e = \boldsymbol{x}_e | \boldsymbol{X}_q = \boldsymbol{x}_q) = \frac{p_\theta(\boldsymbol{X}_e = \boldsymbol{x}_e, \boldsymbol{X}_q = \boldsymbol{x}_q)}{\int_{\boldsymbol{x}_e \in \mathbf{val}(\boldsymbol{X}_e)} p_\theta(\boldsymbol{X}_e = \boldsymbol{x}_e, \boldsymbol{X}_q = \boldsymbol{x}_q)}$ which is known as *conditional inference*. Finally, given a partial assignment $\boldsymbol{x}_e$ to a set of variables $\boldsymbol{X}_e$, finding the most probable assignment for the remaining variables $\boldsymbol{X}_q$, i.e. $\boldsymbol{x}_q = \text{argmax}_{\boldsymbol{x}'_q \in \boldsymbol{X}_q} p_\theta(\boldsymbol{X}_q = \boldsymbol{x}'_q | \boldsymbol{X}_e = \boldsymbol{x}_e)$ is *maximum a posteriori* or *MAP inference*. Note that while we consider only the above queries, more complex tasks such as computing marginal-MAP, moments, etc. exist and we refer to Choi et al. [2020] for an elaborate review.

Probabilistic Circuits (PCs): PCs are tractable probabilistic models, defined as rooted directed acyclic graphs (DAGs), in which leaf nodes represent univariate probabil-

ity distributions and non-terminal nodes represent either a mixture (or states of an observed variable in case of a deterministic circuit) or an independence relation of their children. More formally:

Definition 1. *A **probabilistic circuit** (PC or simply $\mathcal{C}$) is a computational graph that is composed of three types of nodes - sum nodes $\mathcal{S}$, product nodes $\mathcal{P}$ and leaf nodes $\mathcal{L}$. Each node in the graph computes a non-negative function over a set of variables $\psi \subset \{X_i\}_{i=1}^d$, which is defined as its scope. Taken together, $\mathcal{C}$ encodes a probability distribution over $\boldsymbol{X}$, the probability density (or mass) function of which is given by the value of its root node and is defined recursively as follows:*

1. *The value of a sum node($\mathcal{S}$) is equal to a convex combination of the values of its children, i.e. $\mathcal{S}(\boldsymbol{x}) = \sum_{N_i \in \boldsymbol{ch(S)}} w_i N_i(\boldsymbol{x})$; $0 \leq w_i \leq 1$ and $\sum_i w_i = 1$*
2. *The value of a product node ($\mathcal{P}$) is equal to the product of the values of its children, i.e., $\mathcal{P}(\boldsymbol{x}) = \prod_{N_i \in \boldsymbol{ch(P)}} N_i(\boldsymbol{x}_{\boldsymbol{\psi}_{\boldsymbol{N_i}}})$*
3. *The leaf node $\mathcal{L}$ represents a simple probability distribution over its scope with an analytically computable leaf density, such as a Gaussian. For a given $\boldsymbol{x}$, the value of the leaf node equals the probability density (or mass if discrete) of $\boldsymbol{x}$ w.r.t the leaf distribution.*

where $\mathcal{N}$ represents an arbitrary node, $\boldsymbol{ch(N)}$ refer to the children of $\mathcal{N}$ in the graph and $\boldsymbol{\psi_N}$ refer to the scope of $\mathcal{N}$.

To ensure that $\mathcal{C}$ encodes a valid distribution and enables tractability for inference tasks, there are certain structural properties that it often needs to satisfy:

P. 1 (Smoothness). *$\mathcal{C}$ is smooth if its sum nodes are defined over children having the same scope, i.e. $\forall \mathcal{S} \in \mathcal{C}, \psi(\boldsymbol{N_i}) = \psi(\boldsymbol{N_j}) \forall \boldsymbol{N_i}, \boldsymbol{N_j} \in \boldsymbol{ch(S)}$.*

P. 2 (Decomposability). *$\mathcal{C}$ is decomposable if its product nodes are defined over children having disjoint scopes, i.e., $\forall \mathcal{P} \in \mathcal{C}$, and $\forall \boldsymbol{N_i}, \boldsymbol{N_j} \in \boldsymbol{ch(P)}$, $\psi(\boldsymbol{N_i}) \cap \psi(\boldsymbol{N_j}) = \emptyset$.*

Smoothness and decomposability are necessary and sufficient conditions for $\mathcal{C}$ to enable tractability for marginal and conditional inference [Vergari et al., 2021]. Circuits that support MAP inference are additionally required to satisfy determinism.

P. 3 (Determinism). *$\mathcal{C}$ is deterministic if for all the sum nodes, the value of only one of its children is non zero for a given fully observed input, $\boldsymbol{x}$ i.e., $\forall \mathcal{S} \in \mathcal{C}, \sum_{\boldsymbol{N_i} \in \boldsymbol{ch(S)}} \mathbb{1}_{\boldsymbol{N_i}(\boldsymbol{x})>0} = 1$.*

Normalizing Flows (NFs): NFs constitute a class of deep-generative models that build flexible probability distributions over continuous spaces by utilizing the change of variables formula and invertible transformations. Specifically, a normalizing flow transforms a complex distribution $p_{\boldsymbol{X}}$ defined over $\boldsymbol{x} \in \mathbb{R}^d$ into a simple base distribution $p_{\boldsymbol{Z}}$, through a bijective transformation g, such that $\boldsymbol{z} = g(\boldsymbol{x})$. Further, g should satisfy that both g and g^{-1} are continuous and differentiable, a.k.a diffeomorphism. The probability density in the $\boldsymbol{X}$ space is given by the change of variables formula as:

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = p_{\boldsymbol{Z}}(\boldsymbol{z}) |\det J_g| \tag{1}$$

$p_{\boldsymbol{Z}}(\boldsymbol{z})$ is assumed to be a simple distribution such as a Gaussian and $|\det J_g|$ denotes the absolute value of the determinant of the Jacobian of the transformation g evaluated at $\boldsymbol{x}$. Sampling from the distribution $p_{\boldsymbol{x}}$ can be achieved by sampling $\boldsymbol{z}^* \sim p_{\boldsymbol{z}}$ and applying the transformation g^{-1}, i.e. $\boldsymbol{x}^* = g^{-1}(\boldsymbol{z}^*)$. Since diffeomorphisms are closed under compositions, we can "flow" the base distribution through multiple compositions of invertible transformations to attain higher representational flexibility. Building efficient (neural) parameterizations for the invertible transformations (g) with efficiently computable Jacobian determinants is the key research question in the field of normalizing flows.

3 INTEGRATION OF PCs AND FLOWS

A key factor that distinguishes NFs from other deep models is that they support exact density evaluation, i.e., NFs are tractable models for evidential inference tasks. On the other hand, PCs are less expressive than NFs but tractable for more inference tasks under appropriate structural constraints.

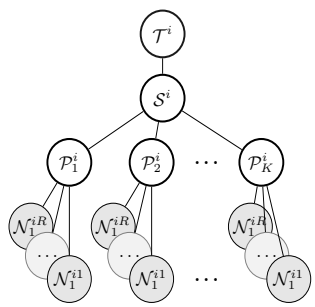
One way to combine NFs and PCs would be to integrate the invertible transformations of flows within PCs, which was first explored by Pevný et al. [2020]. They called the resulting model a sum product transform network (SPTN).

Definition 2. *A SPTN is a PC obtained by augmenting $\mathcal{C}$ with an additional node type, called transform node. Each transform node ($\mathcal{T}$) has a single child $\mathcal{N}$ and can be introduced arbitrarily over any node in $\mathcal{C}$, and $\psi_{\mathcal{T}} = \psi_{\mathcal{N}}$. The value of a transform node is defined as, $\mathcal{T}(\mathcal{N}(\boldsymbol{x})) = \mathcal{N}(g(\boldsymbol{x}))|\det J_g|$, where g denotes a diffeomorphic transformation associated with $\mathcal{T}$ and $|\det J_g|$ denotes the absolute value of the determinant of the Jacobian of g.*

Pevný et al. [2020] proposed to model the transformations g associated with $\mathcal{T}$ as invertible affine transformations. To enable tractability, they considered PCs that are smooth and decomposable, with Gaussian distributions at the leaves, and argued that the addition of transform nodes maintained the properties. They reasoned that Gaussian distributions are closed under product operations and affine transformations, i.e., the product of two Gaussians is a Gaussian, an affine transformed Gaussian is a Gaussian, and hence in the induced tree view [Choi et al., 2020] of a PC, the addition of transform nodes does not introduce any change.

Though it appears intuitive, we argue that this construction is **incomplete** and can result in violating the structural properties of the PC. We formalize this issue first before discussing the solution to overcome it.

Lemma 1. *An SPTN with invertible affine transformations, as constructed by Def. 2, can violate the decomposability property of a PC and is thus not guaranteed to be a tractable model for marginal and conditional inference.*



Proof. Consider an arbitrary section of an SPTN comprising a sum node $\mathcal{S}^i$, product nodes $\{\mathcal{P}_j^i\}_{j=1}^K$, a transform node $\mathcal{T}^i$ as shown in the figure above.

We will show that the transform node causes the scope of the children of P^i to overlap, thus violating the decomposability property. Let g_i denote the invertible affine transformation associated with $\mathcal{T}^i$. Let Ω be the scope of $\mathcal{T}^i$, i.e $\psi_{\mathcal{T}^i} = \Omega = \psi_{\mathcal{S}^i} = \psi_{\mathcal{P}_j^i} \forall j \in \{i\}_{i=1}^K$ and $\boldsymbol{x} \in \mathbb{R}^{|\Omega|} = (x_1, x_2, \ldots, x_{|\Omega|})$. For $\psi \subset \Omega$ let $\mathbf{\Pi}_\psi(\boldsymbol{x}) \in \mathbb{R}^{|\psi|}$ denote the projection of the $\boldsymbol{x}$ on ψ, i.e., $\boldsymbol{x}_\psi = \mathbf{\Pi}_\psi(\boldsymbol{x}) = (x_i)_{i\in\psi}$ is the $|\psi|-$dimensional vector obtained by indexing $\boldsymbol{x}$ along the dimensions contained in ψ. We have,

$$\begin{aligned}
\mathcal{T}^i(\mathcal{S}^i(\boldsymbol{x})) &= \mathcal{S}^i(g^i(\boldsymbol{x}))|\det J_{g^i}| \\
&= \sum_{j=1}^K w_j \mathcal{P}_j^i(g^i(\boldsymbol{x}))|\det J_{g^i}| \\
&= \sum_{j=1}^K w_j \left[\prod_{k=1}^R \mathcal{N}_j^{ik}(\mathbf{\Pi}_{\psi_{\mathcal{N}_j^{ik}}}(g^i(\boldsymbol{x})))\right] |\det J_{g^i}|
\end{aligned}$$

Let $\boldsymbol{z}_j^{ik}$ denote the input to $\mathcal{N}_j^{ik}$. Thus, for $g^i(\boldsymbol{x}) = \boldsymbol{W^i}(\boldsymbol{x}) + \boldsymbol{b^i}$, we have

$$\begin{aligned}
\boldsymbol{z}_j^{ik} &= \mathbf{\Pi}_{\psi_{\mathcal{N}_j^{ik}}}(g^i(\boldsymbol{x}))) = \mathbf{\Pi}_{\psi_{\mathcal{N}_j^{ik}}}(\boldsymbol{W^i}(\boldsymbol{x}) + \boldsymbol{b^i}) \\
&= (\sum_{l=1}^{|\Omega|} \boldsymbol{W}_{ml}^i x_l + b_m)_{m\in\psi_{\mathcal{N}_j^{ik}}}
\end{aligned}$$

where we use the notation $\boldsymbol{z} = (z_m)_{m=1}^n$ for an n-dimensional vector $\boldsymbol{z} \in \mathbb{R}^n$. Clearly, each child $\{\mathcal{N}_j^{ik}\}_{k=1}^R$ of $\mathcal{P}_j^i$ computes a function over $\boldsymbol{x} \in \mathbb{R}^{|\Omega|}$ and thus have overlapping scopes, $\implies$ $\mathcal{P}_j^i$ is not decomposable $\implies$ the SPTN is not decomposable. □

Note that while we showed that SPTNs with affine transformations violate decomposability, this is true for any invertible transformation $\boldsymbol{y} = g(\boldsymbol{x})$ that computes each dimension y_i of $\boldsymbol{y}$ as some function of $\boldsymbol{x}$. Thus we need to define further structural properties when transform nodes are introduced to a PC to maintain its decomposability.

A close look at the proof for Lemma 1 tells us that we can overcome this issue, if $\mathcal{T}$ was constrained in a manner such that $\mathcal{T}(\mathcal{P}(x))$ independently transforms the subset of variables involved in the scope of the children of $\mathcal{P}$. We formalize this notion as a property, which we call τ-decomposability.

Definition 3. *Let $\boldsymbol{x} \in \mathbb{R}^d$, $g : \mathbb{R}^d \to \mathbb{R}^d$ and $\boldsymbol{y} = g(\boldsymbol{x})$. We call g to be decomposable w.r.t a collection (Ω) of disjoint subsets of $\{1, \ldots, d\}$, if $\boldsymbol{y}_{\psi_i} \perp \boldsymbol{x}_{\psi_j}$ $\forall \psi_i, \psi_j \in \Omega, i \neq j$, where $\boldsymbol{y}_\psi \in \mathbb{R}^{|\psi|}$ denotes the vector obtained by indexing y along the dimensions contained in ψ. We use $\boldsymbol{y}_{\psi_i} \perp \boldsymbol{x}_{\psi_j}$ to imply that $\frac{\partial y_a}{\partial x_b} = 0$ $\forall a \in \psi_i, b \in \psi_j$ i.e., it transforms the sets of dimensions contained in Ω independently, i.e. $\boldsymbol{y}_{\psi_i} = g_{\psi_i}(\boldsymbol{x}_{\psi_i})$, where $g_{\psi_i} : \mathbb{R}^{|\psi_i|} \to \mathbb{R}^{|\psi_i|}$.*

P. 4 (τ-decomposability). *A sum product transform network $\mathcal{C}_{\mathcal{SPTN}}$ is τ-decomposable iff all of its transform nodes are τ-decomposable. A transform node $(\mathcal{T})$ is τ-decomposable if, when defined over a decomposable product node $\mathcal{P}$, the transformation g associated with $\mathcal{T}$ is decomposable w.r.t the collection of the scopes of children of $\mathcal{P}$.*

Lemma 2. *τ-decomposability is a necessary condition for a SPTN to be decomposable.*

The proof is a generalization of Lemma 1 and is presented in the appendix. Since decomposability is a necessary condition for a PC to be tractable over marginal and conditional queries, Lemma 2 implies that τ-decomposability is a necessary condition for a SPTN to be tractable w.r.t conditional and marginal inference tasks. We know that all element-wise transformations are, by definition, τ-decomposable. But how does τ-decomposability affect the expressiveness of $\mathcal{C}_{\mathcal{SPTN}}$? The following theorem provides some insight.

Theorem 1. *A τ-decompasable SPTN is equivalent to a PC comprising sum nodes and product nodes retaining the same graph structure, but with all transform nodes from the root to each leaf now pushed down as a composition of transformations (i.e. a normalizing flow) defined just over the corresponding leaf nodes.*

Proof. Let $\mathcal{C}_{\mathcal{SPTN}}$ be τ-decomposable sum product transform network, and $\mathcal{S}, \mathcal{P}, \mathcal{T}, \mathcal{L} \in \mathcal{C}_{\mathcal{SPTN}}$ denote a sum node, product node, transform node, and leaf node respectively. For any arbitrary node $\mathcal{N}$, we will show that defining transformations over $\mathcal{N}$ reduces to transformations over the children of $\mathcal{N}$.

Case 1: When $\mathcal{N} = \mathcal{S}$, we have,

$$\begin{aligned}\mathcal{T}(\mathcal{S}(\boldsymbol{x})) &= \mathcal{S}(g(\boldsymbol{x}))|\det J_g| \\ &= \sum_{\mathcal{N}_i \in ch(\mathcal{S})} w_i \mathcal{N}_i(g(\boldsymbol{x}))|\det J_g| \\ &= \sum_{\mathcal{N}_i \in ch(\mathcal{S})} w_i \mathcal{T}(\mathcal{N}_i(\boldsymbol{x}))\end{aligned}$$

Case 2: When $\mathcal{N} = \mathcal{P}$, we have,

$$\begin{aligned}\mathcal{T}(\mathcal{P}(\boldsymbol{x})) &= \mathcal{P}(g(\boldsymbol{x}))|\det J_g| \\ &= \left[\prod_{\mathcal{N}_i \in ch(\mathcal{P})} \mathcal{N}_i(\Pi_{\psi_{\mathcal{N}_i}}(g(\boldsymbol{x})))\right]|\det J_g| \\ &= \left[\prod_{\mathcal{N}_i \in ch(\mathcal{P})} \mathcal{N}_i(g_{\psi_{\mathcal{N}_i}}(\boldsymbol{x}_{\psi_{\mathcal{N}_i}}))\right]|\det J_g|\end{aligned}$$

Since g is decomposable w.r.t the collection (Ω) of scopes of children of $\mathcal{P}$, the Jacobian of g is a block diagonal matrix, each block corresponding to the jacobian of the independent transformations, $J_{g_{\psi_i}}$ for $\psi_i \in \Omega$. Thus, $|\det J_g| = \prod_{\mathcal{N}_i \in ch(\mathcal{P})} |\det J_{g_{\psi_{\mathcal{N}_i}}}|$. Hence,

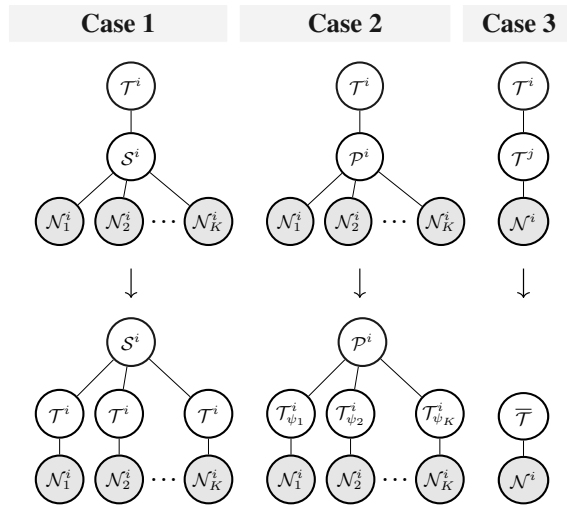
$$\begin{aligned}\mathcal{T}(\mathcal{P}(\boldsymbol{x})) &= \prod_{\mathcal{N}_i \in ch(\mathcal{P})} \left[\mathcal{N}_i(g_{\psi_{\mathcal{N}_i}}(\boldsymbol{x}_{\psi_{\mathcal{N}_i}}))|\det J_{g_{\psi_{\mathcal{N}_i}}}|\right] \\ &= \prod_{\mathcal{N}_i \in ch(\mathcal{P})} \mathcal{T}_{\psi_{\mathcal{N}_i}}(\mathcal{N}_i(\boldsymbol{x}))\end{aligned}$$

Case 3: When $\mathcal{N} = \mathcal{T}^{i-1}$, let us denote by $\mathcal{N}^{i-1}$ the child node of $\mathcal{T}^{i-1}$. We have,

$$\begin{aligned}\mathcal{T}^i(\mathcal{T}^{i-1}(\mathcal{N}^{i-1}(\boldsymbol{x}))) &= \mathcal{T}^i(\mathcal{N}^{i-1}(g^{i-1}(\boldsymbol{x}))|\det J_{g^{i-1}}| \\ &= \mathcal{N}^{i-1}(g^{i-1}(g^i(\boldsymbol{x})))|\det J_{g^{i-1}}||\det J_{g^i}| \\ &= \mathcal{N}^{i-1}(\overline{g}(\boldsymbol{x}))|\det J_{\overline{g}}|, \text{ where } \overline{g} = g^{i-1} \circ g^i \\ &= \overline{\mathcal{T}}(\mathcal{N}^{i-1}(\boldsymbol{x}))\end{aligned}$$

i.e. $\mathcal{T}^i$ with transformation g^i and $\mathcal{T}^{i-1}$ with transformation g^{i-1} can be combined into a single transform node $\overline{\mathcal{T}}$ with an associated transformation $\overline{g} = g^{i-1} \circ g^i$. This follows from the fact that compositions of invertible transformations are invertible and the jacobian of composition of two transformations can be written as the product of the jacobians of the individual transformations.

Thus recursively pushing down transformations along the path from the root to each leaf node, we end up with compositions of transformations defined only over the leaf nodes. Let l_i be the number of transform nodes along the path from the root to the leaf $\mathcal{L}^i$ and let $\overline{\mathcal{T}}$ denote the composition $\mathcal{T}^1 \circ \mathcal{T}^2 \ldots \mathcal{T}^{l_i}$. $\mathcal{C}_{\mathcal{SPTN}}$ has now reduced to containing only sum nodes, product nodes and a transformed leaf node, where the new leaf node $\overline{\mathcal{L}}$ is defined as: $\overline{\mathcal{L}}(\boldsymbol{x}) = \overline{\mathcal{T}}(\mathcal{L}(\boldsymbol{x})) = \mathcal{L}(g^1 \circ g^2 \circ \ldots \circ g^{l_i}(\boldsymbol{x})) \prod_{i=1}^{l} |\det J_{g^i}|$



Thus, each leaf node now becomes a normalizing flow. □

Each leaf in a PC typically encodes a tractable distribution over a single random variable. Thus, though it is intuitive to think of integrating NFs with PCs by the introduction of transform nodes arbitrarily in the circuit as done in Pevný et al. [2020], Thm. 2 shows that, when ensuring tractability, such circuits reduce to a PC with a new type of leaf node.

Definition 4 (**Probabilistic Flow Circuit**). *A probabilistic flow circuit (PFC) is a computational graph that defines a probability distribution over a set of continuous random variables ($\boldsymbol{X}$). It comprises of three types of nodes: (1) Sum nodes $\mathcal{S}$ that computes a convex combination of the values of its children. (2) Product nodes $\mathcal{P}$ that computes a product of the values of its children. (3) Leaf nodes ($\mathcal{L}_{\mathcal{F}}$) defined as normalizing flows over individual features $X_i \in \boldsymbol{X}$.*

NFs are expressive deep models that can effectively model arbitrarily complex probability distributions. Thus, a PFC augments the expressivity of PCs by allowing complex distributions to be modeled at the leaf nodes. It also retains the tractability of PCs as the transformations are restricted only to the leaves. However does integrating any invertible transformation at the leaf provide the same added expressivity?

To understand this, consider the invertible affine transformations that were proposed by Pevný et al. [2020]. We know that compositions of affine transformations can be reduced to a single affine transformation. We also know that an affine transformed Gaussian is still a Gaussian. Thus, the addition of invertible affine transformations at leaf nodes, still only gives us the ability to model Gaussian distributions at the leaf, and does not seem to help improve expressivity. Now, with the framework for integrating NFs with PCs constructed, let us define expressive flow transformations.

Probabilistic Flow Circuits via Linear Rational Splines The requirement at the leaf node for a PFC is that it should enable tractable computation of probability densities. Any

diffeomorphic transformation enables computing the exact density using the change of variables formula. Thus, any expressive normalizing flow architecture can, in principle, be used to implement a probabilistic flow circuit.

However, the nature of inference tasks that we need tractability for can demand additional characteristics that are desirable of the diffeomorphisms defined at the leaves. For example, in order to support MAP inference, deterministic PCs require that the argmax over the leaf density is easily computable. In PCs, since leaf distributions over continuous variables are typically parameterized as Gaussians, the argmax (or the mode of the distribution) corresponds to the mean. However, by defining distributions using normalizing flows, the leaf nodes in a PFC now model arbitrarily complex probability distributions. Since they are no longer restricted to be uni-modal, computing the argmax over this distribution becomes a challenging task. Thus, one design principle for probabilistic flow circuits could be defining diffeomorphisms that support easy computation of modes, without imposing significant limitations on its expressivity.

We propose to parameterize g using a family of invertible piecewise functions, known as *linear rational splines (LRS)*. Spline-based normalizing flows [Dolatabadi et al., 2020, Durkan et al., 2019] have attracted much research interest in recent years and are among the state-of-the-art. We will adapt the computationally efficient LRS transformations defined by Dolatabadi et al. [2020] to define expressive leaf distributions with easily computable modes.

Definition 5. *An invertible linear rational spline transformation is a piecewise function that divides the input space (X) into bins (or intervals) and defines a monotonic rational function of the form $y = \frac{a_1 x + b_1}{a_2 x + b_2}$ within each bin, such that they are knotted together at the boundaries so as to define a differentiable function. Formally, given a set of monotonically increasing points $\{(x^k, y^k)\}_{k=0}^{K}$ known as knots, derivatives $\{d^k > 0\}_{k=0}^{K}$ at the knots, and parameters $\{\lambda^k \in (0,1)\}_{k=0}^{K}$, the LRS transformation within each bin, say for $x \in [x^k, x^{k+1}]$ is defined as:*

$$y = g_{lrs}(\phi), \textit{where } \phi = \frac{(x - x^k)}{(x^{k+1} - x^k)}, \textit{and}$$

$$g_{lrs}(\phi) = \begin{cases} \frac{w^k y^k (\lambda^k - \phi) + w^m y^m \phi}{w^k (\lambda^k - \phi) + w^m \phi} & 0 \leq \phi \leq \lambda^k \\ \\ \frac{w^m y^m (1-\phi) + w^{k+1} y^{k+1} (\phi - \lambda^k)}{w^m (1-\phi) + w^{k+1} (\phi - \lambda^k)} & \lambda^k \leq \phi \leq 1 \end{cases}$$

Given the knots and the derivatives, the parameters $w^k, w^m, w^{k+1}, \lambda^k$ and y^m are constructed in such a manner [Dolatabadi et al., 2020] that the overall function defined is differentiable everywhere. The width and height of each bin, the derivatives at the knots, and λ are defined as learnable parameters for an LRS flow. The form for the analytical inverse of g_{lrs} and its jacobian are given in Dolatabadi et al. [2020]. Defining a flow at the leaf also requires specifying a base distribution. While any distribution with an analytically computable density can be used, for theoretical purposes, we will use a Student's-t distribution with 3 degrees of freedom, as the form of its density function simplifies the analysis when used with g_{lrs}. We summarize the tractability of our model below before we present empirical results.

Lemma 3. *A PFC ($\mathcal{C}_{\mathcal{F}}$) with leaf distributions defined using g_{lrs} transformations and a Student's-t distribution with $\nu = 3$ as the base distribution is a tractable model for (a) evidential inference if $\mathcal{C}_{\mathcal{F}}$ is smooth, (b) Marginal and conditional inference if $\mathcal{C}_{\mathcal{F}}$ is smooth and decomposable, (c) MAP inference if $\mathcal{C}_{\mathcal{F}}$ is smooth, decomposable, and deterministic.*

Proof. Deferred to the supplementary. □

4 EXPERIMENTS & RESULTS

We experimentally validated the efficacy of probabilistic flow circuits in modeling a wide range of data distributions. We used einsum networks [Peharz et al., 2020a] with Gaussian leaves to parameterize PCs. Einsum networks vectorize the leaf distributions in a PC and implement the sum and product operations using a monolithic *einsum*-operation defined over a probability tensor. The balanced tree structure of its computational graph allows computing the leaf densities in parallel and processing it through stacked einsum-layers, similar to transforming an input through a deep neural network. We instantiated our PFC by integrating a linear rational spline flow as the input layer within einsum networks. To enable parallel computation of all the leaf densities, we implemented our input layer as a single conditional NF, conditioned on the index of the leaf distribution.

We will refer to the base PC as EinsumNet and our probabilistic flow circuits as EinsumNet+LRS. As our objective is to evaluate the added expressivity brought in by the flexible leaf distributions, we used the same circuit structure for both EinsumNet and EinsumNet+LRS, and compared their performances on a multitude of datasets. The two key parameters constraining the structure and expressivity of an einsum network are the number of vector components (k) and the number of replicas (r) (see Peharz et al. [2020a] for more details). We used $k = r = 10$ for our 3D data experiments and $k = r = 20$ for our experiments on higher dimensional data. As einsum networks are differentiable like neural networks, we used end-to-end backpropagation and trained all our models using an Adam optimizer, with a learning rate of $1e-3$. We defer further implementation details to the supplementary. Our code is implemented using pytorch and pyro [Bingham et al., 2019], adapted from Peharz et al. [2020a] and is publicly available[1].

[1] https://github.com/sahilsid/probabilistic-flow-circuits

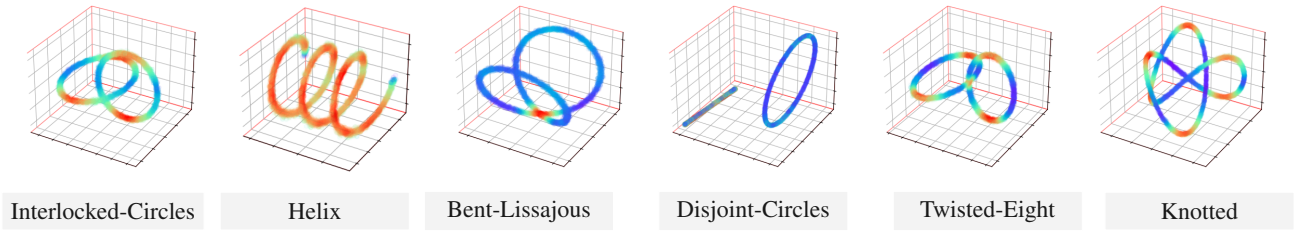


Figure 2: Visualizations of the 6 **3D data distributions** over complex manifolds considered in this work.

	Helix	Knotted	Disjoint-Circles	Interlocked-Circles	Twisted-Eight	Bent-Lissajous
EinsumNet	-2.94 ± 0.24	-5.23 ± 0.07	-1.71 ± 0.19	-2.88 ± 0.07	-3.02 ± 0.24	-3.02 ± 0.06
EinsumNet + Affine	-2.47 ± 0.09	-5.18 ± 0.11	-1.18 ± 0.06	-2.52 ± 0.05	-2.79 ± 0.07	-2.69 ± 0.05
EinsumNet + LRS	-1.04 ± 0.02	-4.09 ± 0.03	-0.81 ± 0.03	-2.44 ± 0.02	-2.43 ± 0.01	-2.53 ± 0.02

Table 1: **Performance Evaluation** of - (a) *EinsumNet* (b) *EinsumNet+Affine*, and (c) *EinsumNet+LRS* on the **3D datasets**, in terms of mean test log-likelihood (↑), ± the standard deviation across 3 trials.

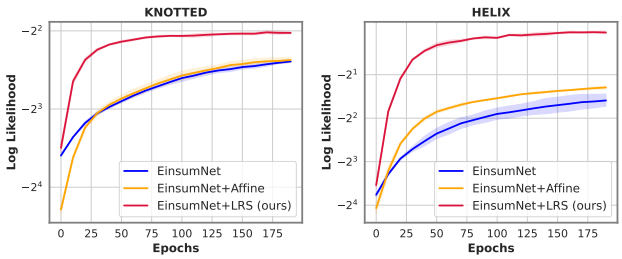


Figure 3: **Learning curves** of - (a) *EinsumNet* (b) *EinsumNet+Affine*, and (c) *EinsumNet+LRS* on the **3D datasets** (mean validation log-likelihood across training epochs). The shaded regions depict the standard deviation across 3 trials.

We aim to answer the following questions empirically:

Q1 Do probabilistic flow circuits provide more accurate density estimates for complex distributions?

Q2 If so, are the inference tasks still tractable with these models while generating better quality samples?

Q3 Can we expect this added flexibility to hold in general, or is it closely coupled with the expressivity of the underlying PC ?

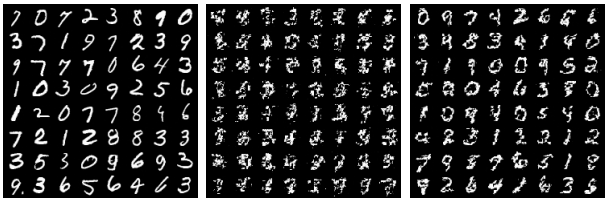
4.1 DENSITY ESTIMATION

3D Manifold Data. We first considered 6 data distributions over complex 3D manifolds, adapted from Sidheekh et al. [2022]. These data distributions are easy to visualize (see Fig. 2), being in a low dimensional space, but also challenging to model, owing to their complex manifold structure. They are thus useful distributions to evaluate and compare generative models. We utilized these data distributions to empirically validate (a) the performance improvement achieved by having flows as leaf distributions and (b) the better expressivity offered by linear rational spline transformations over the invertible affine transformations proposed in Pevný et al. [2020]. We generated $20,000$ data points for each of the 6 3D datasets, $10,000$ of which we used for training and $5,000$ each for validation and testing. We trained three models: (a) EinsumNet, (b) EinsumNet + Affine, and (c) EinsumNet + LRS on each dataset, where EinsumNet + Affine refers to the probabilistic flow circuit obtained by integrating invertible affine transformations as the leaf flow. To ensure invertibility, we implemented the affine transformations in their SVD decomposed form, as proposed in Pevný et al. [2020], utilizing the householder parameterization for generating unitary matrices.
Fig. 3 shows the validation log-likelihood achieved by each model across the training epochs on two 3D datasets. As one can see, EinsumNet + Affine performs similar to EinsumNet in many cases or slightly better. This is expected as the affine transformed gaussian leaf distributions are still gaussians. EinsumNet + LRS on the other hand, not only achieves significantly better performance than both EinsumNet and EinsumNet + Affine, but it is able to do so much faster. Similar learning curves on the other 4 datasets are provided in the supplementary. Tab. 1 provides further quantitative evidence in terms of the test log-likelihood of each of the three models on the 3D datasets, thus validating the added flexibility of EinsumNet + LRS.

High dimensional data. Having established the utility of probabilistic flow circuits for modeling 3D data distributions, we now proceed to study how the expressivity scales when learning data distributions in higher dimensional spaces. We considered 2 image datasets - MNIST [Deng, 2012], Fashion-MNIST [Xiao et al., 2017] and 4

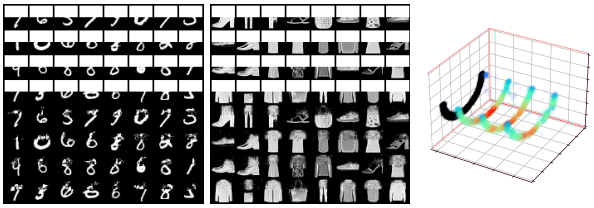
	POWER	GAS	MINIBOONE	HEPMASS	MNIST	Fashion-MNIST
MADE	-3.08 ± 0.03	3.56 ± 0.04	-15.59 ± 0.50	-20.98 ± 0.02	-1380.8 ± 4.8	-
Real NVP	-0.02 ± 0.01	4.78 ± 1.80	-13.55 ± 0.49	-19.62 ± 0.02	-1323.2 ± 6.6	-
MAF	0.14 ± 0.01	9.07 ± 0.02	-11.75 ± 0.44	-17.70 ± 0.02	-1300.5 ± 1.7	-
EinsumNet	0.20 ± 0.01	3.57 ± 0.08	-35.93 ± 0.06	-22.79 ± 0.05	-1015.1 ± 0.9	649.1 ± 0.3
Einsum+LRS	0.36 ± 0.01	4.79 ± 0.04	-34.21 ± 0.01	-22.46 ± 0.01	-959.4 ± 1.4	655.6 ± 0.6

Table 2: **Performance Evaluation** of - (a) *EinsumNet* and (b) *Einsum Net+LRS* on the **tabular** & **image** datasets, in terms of mean test log-likelihood (↑), ± the standard deviation across 3 trials.



(a) Ground-Truth (b) Einsum Net (c) Einsum+LRS

Figure 4: **Qualitative Evaluation** - samples generated by *EinsumNet* and *EinsumNet+LRS* on the MNIST dataset.



(a) MNIST (b) Fashion-MNIST (c) Helix

Figure 5: Generating samples from conditional distributions using *EinsumNet+LRS*.

UCI tabular datasets commonly used to evaluate density estimation in the normalizing flow literature. We followed the preprocessing done in Papamakarios et al. [2017] for each of these datasets. We trained the two models - EinsumNet and EinsumNet+LRS for 100 epochs over the tabular datasets and 50 epochs over the image datasets. Tab. 2 reports the mean test log-likelihood achieved by both models on these datasets. One can clearly see EinsumNet+LRS consistently achieves better performance as compared to EinsumNet on all the 6 datasets. We also analyzed how these tractable models compare against deep models that are expressive but not quite tractable, in Tab. 2 (values taken from Papamakarios et al. [2017]). Specifically, we considered MADE [Germain et al., 2015], which is a deep autoregressive generative model, and two classic normalizing flow models, RealNVP [Dinh et al., 2017] and MAF [Papamakarios et al., 2017]. Interestingly, EinsumNet+LRS achieves similar performance as RealNVP on the GAS dataset and even performs better than the deep models on the POWER and MNIST datasets. We would like to add that even though these deep models may not be the most state of the art, they are still models that trade off tractability for expressivity. These results clearly suggest that integrating NFs as leaf distributions within PCs can help achieve better of both worlds and bridge the gap between tractable and expressive models.

4.2 SAMPLE GENERATION

The ability to sample new data points from the underlying distribution is one of the key desirable features of any generative model. To understand how the integration of flows affects the quality of sample generation, we show samples generated randomly from an EinsumNet and EinsumNet+LRS trained on the MNIST dataset in Fig. 4. As one can see, EinsumNet+LRS is able to generate samples of higher fidelity than EinsumNet. Further, as probabilistic flow circuits are tractable models, one can utilize their tractability to gain better control over the data generation process. This can be helpful in many real world scenarios, for example in the case of missing data or generating data from regions having certain properties. In Fig. 5 (a) and (b), we occlude the top half of images taken from MNIST and Fashion-MNIST datasets, and use an EinsumNet+LRS to generate samples from the conditional distribution over the occluded pixels given the non-occluded pixels. Similarly in Fig. 5 (c) we used an EinsumNet+LRS trained on the 3D Helix dataset to generate datapoints having the property that their projection to the XY plane equals the black arc plotted in the Figure. Note that due to the lack of tractability, controlled sample generation like these are not possible for other deep generative models like GANs, VAEs or NFs.

4.3 ABLATION STUDY

We also conducted ablation experiments to study how the expressivity of a probabilistic flow circuit varies when the underlying PC is made more complex. To this end, we considered $K_s = \{1, 2, 4, 8, 16, 32\}$ and trained EinsumNet and EinsumNet+LRS on the MNIST dataset by setting the structure parameters $k = r = K$ for each $K \in K_s$.

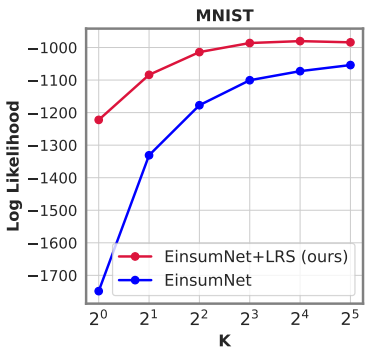


Figure 6: Ablation study demonstrating how the effectiveness of *Einsum+LRS* scales as we increase the expressivity of the base PC on the MNIST dataset.

The test log-likelihood of the models for each parameter setting is shown in Fig. 6. We observe that irrespective of the complexity of the underlying PC, the integration of flows helps better model the data. EinsumNet+LRS performs considerably well even when $K = 1$. This suggests that the expressivity of a probabilistic flow circuit is not tightly coupled with the complexity of the PC.

5 SUMMARY & FUTURE WORK

We presented a theoretically grounded approach to construct deep and tractable generative models using probabilistic circuits and normalizing flows. We empirically validated its added expressivity and tractability in modeling complex data distributions. In many real-world scenarios, tractability might be of interest only for a subset of random variables. In such cases, defining multivariate leaf flows over the remaining variables can help further the expressivity while retaining only the tractability *needed* for the task, which we leave for future work. Our work thus lays the formalisms and foundations that can enable seamless interpolation on the tractability-expressivity spectrum. It also paves the way towards compositional learning, where hybrid data distributions modeled via normalizing flows can now be integrated as leaves in a unified probabilistic circuit.

Acknowledgements

SS and *SN* acknowledge the support by the U. S. Army Research Laboratory and the U. S. Army Research Office (ARO) under grant number W911NF2010224.

KK acknowledges the support by the Hessian Ministry of Higher Education, Research, Science and the Arts (HMWK; projects "The Third Wave of AI" and "The Adaptive Mind"), and the Hessian research priority programme LOEWE within the project "WhiteBox".

References

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1), 2019.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

Alvaro H. C. Correia, Gennaro Gala, Erik Quaeghebeur, Cassio de Campos, and Robert Peharz. Continuous mixtures of tractable probabilistic models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Meihua Dang, Anji Liu, and Guy Van den Broeck. Sparse probabilistic circuits via pruning and growing. In *Advances in Neural Information Processing Systems*, 2022.

Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, may 2003.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 2012.

Denis Deratani Mauá, Diarmaid Conaty, Fabio Gagliardi Cozman, Katja Poppenhaeger, and Cassio Polpo de Campos. Robustifying sum-product networks. *International Journal of Approximate Reasoning*, 101:163–180, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

Hadi Mohaghegh Dolatabadi, Sarah Erfani, and Christopher Leckie. Invertible generative modeling using linear rational splines. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 4236–4246. PMLR, 26–28 Aug 2020.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

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

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR*, 2014.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

Anji Liu, Honghua Zhang, and Guy Van den Broeck. Scaling up probabilistic circuits by latent variable distillation. In *The Eleventh International Conference on Learning Representations*, 2023.

Alejandro Molina, Sriraam Natarajan, and Kristian Kersting. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1):2617–2680, 2021.

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.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *ICML*, 2020a.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *UAI*, 2020b.

Tomáš Pevný, Václav Smídl, Martin Trapp, Ondřej Poláček, and Tomáš Oberhuber. Sum-product-transform networks: Exploiting symmetries using invertible transformations. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 341–352. PMLR, 23–25 Sep 2020.

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.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*, pages 630–645. Springer, 2014.

Or Sharir and Amnon Shashua. Sum-product-quotient networks. In *International Conference on Artificial Intelligence and Statistics*, pages 529–537. PMLR, 2018.

Sahil Sidheekh, Chris B. Dock, Tushar Jain, Radu Balan, and Maneesh K. Singh. Vq-flows: Vector quantized local normalizing flows. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180, pages 1835–1845. PMLR, 2022.

Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 2013.

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

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Martin Trapp, Robert Peharz, Franz Pernkopf, and Carl Edward Rasmussen. Deep structured mixtures of gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 2251–2261. PMLR, 2020.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems*, 2021.

Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Zhongjie Yu, Mingye Zhu, Martin Trapp, Arseny Skryagin, and Kristian Kersting. Leveraging probabilistic circuits for nonparametric multi-output regression. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.