Probabilistic Neural Circuits

Pedro Zuidberg Dos Martires

Centre for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden

Abstract

Probabilistic circuits (PCs) have gained prominence in recent years as a versatile framework for discussing probabilistic models that support tractable queries and are yet expressive enough to model complex probability distributions. Nevertheless, tractability comes at a cost: PCs are less expressive than neural networks. In this paper we introduce probabilistic neural circuits (PNCs), which strike a balance between PCs and neural nets in terms of tractability and expressive power. Theoretically, we show that PNCs can be interpreted as deep mixtures of Bayesian networks. Experimentally, we demonstrate that PNCs constitute powerful function approximators.

1 Introduction

In recent years probabilistic circuits (PCs) (also called sum-product networks) (Darwiche 2003; Poon and Domingos 2011) have emerged as an assembly language to talk about tractable probabilistic models and inference therein (Vergari et al. 2021). The core idea is quite simple: we start with a set of independent random variables and construct complex probability distribution by recursively adding and multiplying them together. There are two common ways of interpreting PCs. Firstly, we can consider them to be hierarchical mixture models. Secondly, we look at them as neural nets consisting of sums, products, and atomic probability distributions.

Most of the recent advances in the field adhere to the second perspective: use an overparametrized probabilistic model and fit it to data using gradient based methods by leveraging discrete GPUs (Peharz et al. 2019; Dang et al. 2021). The computation units of such circuits are organized in a layered fashion. We give an example in Figure 1.

A major advantage of PCs is their ability to answer certain queries in polynomial time – given that adequate restrictions are imposed on a circuit's structure (Vergari et al. 2021). An example of such a tractable query would be the computation of conditional probabilities for so-called *smooth and decomposable* PCs (Darwiche 2001, 2003).

This tractability, however, comes at a hefty price: the properties imposed on PCs in order to ensure polynomial time queries limit their expressive power (Martens and Medabalimi 2014; Sharir and Shashua 2018; Zhang, Juba, and Van den Broeck 2021). This is in contrast to general neural networks and even sum-product networks with fewer structural constraints (Delalleau and Bengio 2011; Kileel, Trager, and Bruna 2019). Martens and Medabalimi (2014) have shown that decomposability is in fact a necessary condition for tractable marginal inference.

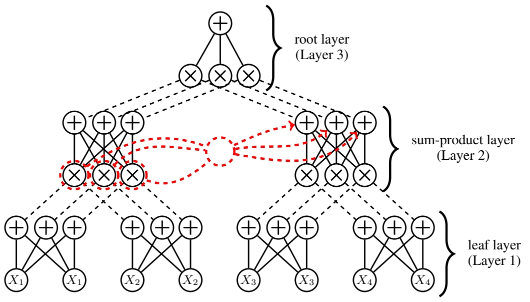

Figure 1: Layered probabilistic circuit following the construction of (Shih, Sadigh, and Ermon 2021). Data (modeled as random variables) is first fed into the leaf layer at the bottom. The output of the leaf layer is a mixture of distributions produced by the sum units. In the sum-product layer (Layer 2) mixtures of random variables are combined by taking pairwise products, these are then again mixed using sum units. Finally, the root layer (at the top) gives us the joint probability distribution. The red edges indicate functional dependencies not present in traditional probabilistic circuits but present in probabilistic neural circuits.

Nevertheless, using the concepts of *conditional smoothness* and *conditional decomposability* (Sharir and Shashua 2018), we study in this paper the space of models that lie in between probabilistic circuits and neural networks. Concretely, we make the following contributions:

1. We introduce conditional probabilistic circuits, from which we construct probabilistic neural circuits (PNCs), which we interpret as deep mixtures of Bayesian nets.
2. We provide a prescription to construct layered PNCs.
3. We provide an implementation of layered PNCs and experimentally study their expressive power.

Our work is influenced by that of Sharir and Shashua (2018). We discuss the relationship to their approach (dubbed sum-product-quotient networks) in Section 5.

2 Preliminaries

In the remainder of the paper we will denote random variables by uppercase X's, the corresponding values are lowercase x's. Sets of random variables and their corresponding values are typed in boldface: $\mathbf{X}$ and $\mathbf{x}$, respectively. The definitions and notions we introduce in this section are loosely based on the work of Vergari et al. (2021).

Definition 2.1 (Probabilistic Circuit)**.** *A probabilistic circuit over random variables $\mathbf{X}$ is a parametrized computational graph encoding a probability density function $p(\mathbf{X}{=}\mathbf{x})$. The circuit consists of three kinds of computational units: leaf, product, and sum. Each product or sum unit receives inputs from a set of input units denoted by $in(n)$. Each unit k encodes a function $p_k(\cdot)$ as follows:*

$$p_k(\mathbf{x}_n) = \begin{cases} f_k(\mathbf{x}_n) & \textit{if } k \textit{ leaf unit} \\ p_{k_l}(\mathbf{x}_{n_l}) p_{k_r}(\mathbf{x}_{n_r}) & \textit{if } k \textit{ product unit} \\ \sum_{j \in in(k)} w_{kj} p_j(\mathbf{x}_n) & \textit{if } k \textit{ sum unit} \end{cases}$$

where $f_k(\mathbf{x}_n)$ denotes a parametrized probability distribution having as support the sample space of the random variables in $\mathbf{X}_n$.

Definition 2.2 (Scope)**.** *The scope of a unit k, denoted by $\phi(k)$, is the set of random variables $\mathbf{X}_n$ for which the function $p_k(\cdot)$ encodes a probability distribution.*

Two important properties that are usually imposed on probabilistic circuits are smoothness and decomposability as they allow for tractable queries, e.g. computing marginals (Darwiche 2001, 2003).

Definition 2.3 (Smoothness)**.** *A circuit is smooth if for every sum unit k its inputs encode distributions over the same random variables: $\forall j_1, j_2{\in}in(k)$ it holds that $\phi(j_1){=}\phi(j_2)$.*

Definition 2.4 (Decomposability)**.** *A circuit is decomposable if the inputs of every product unit k encode distributions over disjoint sets of random variables: $\phi(k_l) \cap \phi(k_r) = \emptyset$ with $\{k_l, k_r\} = in(k)$.*

Definition 2.5 (Valid Probabilsitic Circuit)**.** *We call a probabilistic circuit valid if for every unit k we have that $p_k(\mathbf{x}_n){\geq}0$ and $\int p_k(\mathbf{x}_n)\mathrm{d}\mathbf{x}_n{=}1$.*[1]

As discussed by Peharz et al. (2015), probabilistic circuits are valid if they are smooth, decomposable, and that for the weights in the sum units we have $w_{kj} \in \mathbb{R}^+$ and $\sum_{j \in \text{in}(k)} w_{kj} = 1$ for every k. Note that the notation in Definition 2.1 already suggests that the circuit is smooth as the inputs to the sum units are functions over the same set of variables $\mathbf{X}_n$.

Furthermore, we can assume, without loss of generality, that sum and product units occur in an alternating fashion in the circuit (Peharz et al. 2020). This observation naturally leads to a layer-wise construction of probabilistic circuits where consecutive layers alternate between sum and product layers. Such layered probabilistic circuits (Peharz et al. 2019) have the advantage that the computations within a layer can be trivially parallelized. We can further abstract the layers in a circuit by fusing together alternating sums and products into a single sum-product layer (Peharz et al. 2020).

In Figure 1 we give a graphical representation of a layered circuit. Layers consist of blocks of computational units that are processed sequentially in a bottom-up fashion. Layers are themselves constituted of so-called partitions. The circuit in Figure 1 has four partitions in the leaf layer, two in the sum-product layer, and a single partition in the root layer. By construction, partitions in the same layer have disjoint scopes. Moreover, partitions are further subdivided into input components and output components, which constitute the elemental computing units. The circuit in Figure 1 has, except at the very bottom and top, three such input and output components in each partition.

We can uniquely identify each computational unit (or component) in the circuit by specifying the layer, the partition, whether it is an input or an output, and its position within a partition. Counting layers from bottom to top, and partitions and units from left to right. Each component can be identified using 4 indices: $\kappa_{l,p,i,c}$. The first index l identifies the layer, the second p the partition, the third $i \in \{1, 2\}$ whether it is an input or output, and the fourth c the horizontal position within a partition. For instance, the symbol κ_{2223} corresponds to the upper-right unit in the sum-product layer.

3 Conditional Probabilistic Circuits

We will first introduce the notion of posets (partially ordered sets) of random variables (Section 3.1). This will allow us to generalize probabilistic circuits to conditional probabilistic circuits, which we interpret as deep mixtures of Bayesian networks (Section 3.2). We then introduce probabilistic neural circuits and their tractable queries (Section 3.3).

3.1 Partially Ordered Random Variables

Consider a set of random variables $\mathbf{X}{=}\{X_1, \dots, X_N\}$ on which we impose the *parents* relationship $pa(\cdot)$. The parents relationship induces a directed acyclic graph on the random variables $\mathbf{X}$, where the nodes are the random variables themselves and an edge is present between X_i and X_j if $X_i{\in}pa(X_j)$. This gives us a partial ordering of the variables $\mathbf{X}$. We also define the ancestor relationship $an(\cdot)$ to be the transitive closure of $pa(\cdot)$. That is, the ancestors of a random variable are its direct parents and recursively their parents. Furthermore, we denote the poset for the random variables $\mathbf{X}$ by $\mathcal{O}(\mathbf{X})$. We say that the relationship $\mathbf{X}_i{\sqsubset}\mathbf{X}_j$ between two sets holds if $\forall X_r{\in}\mathbf{X}_i, X_q{\in}\mathbf{X}_j : X_q{\notin}an(\mathbf{X}_r)$. We also define the relation $X_i \sqsubset X_j$ on random variables as $\{X_i\}{\sqsubset}\{X_j\}$.

Example 3.1 (Bayesian Network)**.** *Partially ordered random variables induce a factorization of a joint probability distribution. A prominent example of such a factorization are Bayesian networks (cf. Figure 2) where we have:*

$$p(\mathbf{X}{=}\mathbf{x}) = \prod_{n:X_n \in \mathbf{X}} p_n(X_n{=}x_n \mid \mathbf{X}_{pa(n)}{=}\mathbf{x}_{pa(n)}). \quad (1)$$

In the example above we denote $pa(X_n)$ by $\mathbf{X}_{pa(n)}$. This will allow us to omit the random variable X_n when writing down a probability distribution and only use the instantiation x_n instead.

[1]Note that our notion of validity is slightly stricter than in its original definition, cf. (Poon and Domingos 2011)

$$\begin{array}{ll} X_1 \sqsubset X_2 & X_1 \sqsubset X_4 \\ X_1 \sqsubset X_3 & X_3 \sqsubset X_1 \\ X_3 \sqsubset X_2 & X_3 \sqsubset X_4 \\ X_2 \sqsubset X_3 & X_2 \sqsubset X_4 \end{array}$$

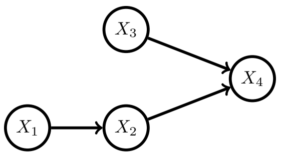

Figure 2: Right: Bayesian network. Left: partial order relations that hold.

3.2 Deep Mixtures of Bayesian Networks

Definition 3.2 (Conditinal Probabilsitic Circuit (CPC)). *A CPC p_k over a poset $\mathcal{O}(\mathbf{X})$ is a parametrized computational graph encoding a probability density function $p(\mathbf{X}{=}\mathbf{x})$. The CPC consists of leaf, product, and sum units. Each unit k encodes a function p_k as follows:*

$$p_k(\mathbf{x}_n \mid \mathbf{x}_{pa(n)}) \tag{2}$$
$$= \begin{cases} f_k(\mathbf{x}_n \mid \mathbf{x}_{pa(n)}) & \textit{if leaf} \\ p_{k_l}(\mathbf{x}_{n_l} \mid \mathbf{x}_{pa(n_l)}) p_{k_r}(\mathbf{x}_{n_r} \mid \mathbf{x}_{pa(n_r)}) & \textit{if product} \\ \sum_{j \in in(k)} w_{kj} p_j(\mathbf{x}_n \mid \mathbf{x}_{pa(n)}) & \textit{if sum} \end{cases}$$

where $f_k(\mathbf{x}_n)$ denotes a parametrized probability distribution having as support the sample space of the random variables in $\mathbf{X}_n$.

Definition 3.3 (Scope (CPC)). *The scope $\phi(k)$ of a unit k encoding the probability distribution $p_k(\mathbf{x}_n \mid \mathbf{x}_{pa(n)})$ is the set of random variables $\mathbf{X}_n$.*

Corollary 3.4. *A conditional probabilistic circuit over an unordered set of random variables is a (non-conditional) probability circuit.*

Proof. Having no order means that $pa(\mathbf{X}) = \emptyset$ for every $X \in \mathbf{X}$. This then means that the conditioning sets in Definition 3.2 are all empty and that we recover a (non-conditional) probabilistic circuit. □

We can now also introduce the notions of *conditional smoothness* and *conditional decomposability*:

Definition 3.5 (Conditional Smoothness). *A CPC is conditionally smooth if for every sum unit k it holds that $\forall j_1, j_2 {\in} in(k) : \phi(j_1){=}\phi(j_2)$*

Definition 3.6 (Conditional Decomposability). *A CPC is conditionally decomposable if for every product unit k it holds that $\phi(k_l) \cap \phi(k_r) = \emptyset$ with $\{k_l, k_r\} = in(k)$.*

Corollary 3.7. *A conditionally smooth and conditionally decomposable CPC is smooth and decomposable if its random variables $\mathbf{X}$ are unordered.*

Definition 3.8 (Valid CPC). *We call a CPC valid if for every unit k we have that $p_k(\mathbf{x}_n|\mathbf{x}_{pa(n)}){\geq}0$ and $\int p_k(\mathbf{x}_n|\mathbf{x}_{pa(n)})\mathrm{d}\mathbf{x}_n{=}1$.*

Theorem 3.9 (Validity for CPCs). *A CPC is valid if it is conditionally smooth, conditionally decomposable, and for every sum unit n it holds that $w_{nm}{\geq}0$ and $\sum_{m \in in(n)} w_{nm}{=}1$.*

Proof. We start by rewriting the alternating sums and products of a CPC in its flat representation using the fact that products distribute over summations:

$$p_k(\mathbf{x}_n) = \sum_{\tau \in \mathcal{T}} w_\tau \prod_{\rho \in \tau} \rho(\mathbf{x}_\rho) \tag{3}$$

Here, $\mathcal{T}$ denotes the set of all products of leaf distributions in the CPC and $\rho \in \tau$ denotes a factor in one of these products (we refer to (Zhao, Poupart, and Gordon 2016) for a more detailed account).

Invoking decomposability of the product units we have that each random variable X_n only picks up a single factor $\rho(x_\rho)$, which means that we can identify each $\rho(x_\rho)$ with a specific $f_k(x_n \mid \mathbf{x}_{pa(n)})$. This lets us rewrite Equation 3 as:

$$p_k(\mathbf{x}_n) = \sum_{\tau \in \mathcal{T}} w_\tau \prod_{k: f_k \in \tau} f_k(\mathbf{x}_n \mid \mathbf{x}_{pa(n)}) \tag{4}$$

Given that the f_k are by definition (conditional) probability distributions their product forms a joint probability distribution as well. Next, we exploit smoothness, which states that the inputs to sum units have identical scope. This means that all the terms in the flat representation of $p(\mathbf{X}_n)$ mention the identical set of random variables and each term in the flat representation forms indeed a joint probability over the random variables $\mathbf{X}_n$.

Lastly, Peharz et al. (2015) have shown that having normalized weights in the sum units of a circuit results in normalized weights w_τ in the flat representation. This lets us conclude that the circuit $p_k(\mathbf{x}_n)$ is a probability distribution. Note that we did not make any reference in our reasoning to any specific unit in the circuit. This means that our argument holds for all units in a conditionally smooth and conditionally decomposable circuit with normalized weights, which means in turn that such a circuit is valid. □

In light of Equation 4 and comparing it to Equation 1, we can interpret CPCs as deep (or hierarchical) mixtures of Bayesian networks. This is analogous to interpreting probabilistic circuits as deep mixtures of fully factorized distributions (Poon and Domingos 2011).

3.3 PNCs and Their Tractable Queries

The computational efficiency of probabilistic circuits stems from the fact the circuits evaluations are broken down into sub-evaluations, which are then cached and reused. Inspecting, however, the functional form of the sum units in a CPC (cf. Equation 2), this is not the case: each term in the sum over j requires a separate conditional probability for each instantiation of the variables $\mathbf{X}_{pa(n)}$. This means that we would need (assuming binary random variables) $2^{|\mathbf{X}_{pa(n)}|}$ functions to encode the conditional probabilities. We alleviate this issue as follows. First, we rewrite the functional form of the sum units using Bayes' rule:

$$\sum_{j \in \mathrm{in}(k)} w_{kj} \frac{p_j(\mathbf{x}_{pa(n)} \mid \mathbf{x}_n)}{p_j(\mathbf{x}_{pa(n)})} p_j(\mathbf{x}_n) \tag{5}$$

Second, we make the following approximation:

$$w_{kj} \frac{p_j(\mathbf{x}_{pa(n)} \mid \mathbf{x}_n)}{p_j(\mathbf{x}_{pa(n)})} \approx \omega_{kj}(\mathbf{x}_{an(n)}), \tag{6}$$

where $\omega_{kj}(\cdot)$ is a neural network depending on the set of ancestors $an(x_n)$. This now allows us to formally introduce probabilistic neural circuits.

Definition 3.10 (Probabilistic Neural Circuit (PNC))**.** *A PNC is a conditionally smooth and conditionally decomposable CPC where sum units take the following functional form:*

$$p_k(\mathbf{x}_n \mid \mathbf{x}_{pa(n)}) = \sum_{j \in in(k)} \omega_{kj}(\mathbf{x}_{an(n)}) p_j(\mathbf{x}_n), \qquad (7)$$

with $\omega_{kj} : \Omega(an(x_n)) \to [0,1]$ being a neural network mapping from the sample space of the random variables $an(x_n)$ to a real between zero and one, and where it holds that $\sum_{j \in in(k)} \omega_{kj} = 1$.

It can easily be seen that PNCs, already encode valid circuits (cf.. Definition 3.8) by construction. Intuitively, we interpret PNCs as neural approximations of CPCs. Note that it is this approximation that makes PNCs tractable: PNCs only need a single circuits for each j while CPCs need a circuit for every j and every instantiation of $\mathbf{x}_{pa(n)}$.

Proposition 3.11. *Probabilistic circuits are PNCs.*

Proof. If the values $\omega(\mathbf{x}_{an(n)})$ do not depend on $\mathbf{x}_{an(n)}$ we have $|\text{in}(k)|$ constants that sum up to 1. This means that the weights in the sum units do not depend on the conditioning sets from the antecedent product layer, and we can omit any conditioning sets. The definition of a PNC in this case is then equivalent to the definition of a probabilistic circuit. □

Given the definition of PNCs we can now determine tractable queries that we can perform.

Proposition 3.12 (Density Evaluation)**.** *Given a probabilistic circuit p_k over the random variables $\mathbf{X}$. We can evaluate the circuit at the instantiation $\mathbf{x}$ in linear time with respect to the size of the circuit.*

Proof. This follows simply from the fact that a circuit is a (non-recurrent) computation graph and that we can simply evaluate it by computing input units before output units. □

Proposition 3.13 (Ordered Marginals)**.** *Consider a PNC $p_k(\mathbf{x}_m, \mathbf{x}_e)$ we can then compute the marginal $p_k(\mathbf{x}_e)$ in polynomial time if $\mathbf{X}_e \sqsubset \mathbf{X}_m$.*

Proof. We start by writing out explicitly the single elements in the set $\mathbf{x}_m$:

$$p_k(\mathbf{x}_m, \mathbf{x}_e) = p_k(x_\mu, \dots, x_1, \mathbf{x}_e) \qquad (8)$$

where the order of $\{x_\mu, \dots, x_1\} = \mathbf{x}_m$ is arbitrary but respects the partial order $\mathcal{O}(\mathbf{X}_m \cup \mathbf{X}_e)$. Given that the circuit is conditionally smooth and conditionally decomposable, we know that it encodes a proper probability distribution over its variables. We can hence obtain the marginal $p(\mathbf{x}_e)$ by integrating over the possible values $\mathbf{x}_m$:

$$p_k(\mathbf{x}_e) = \int \cdots \int p_k(x_\mu, \dots, x_1, \mathbf{x}_e) \mathrm{d}x_\mu \cdots \mathrm{d}x_1 \qquad (9)$$

As x_μ does not appear in any of the conditioning sets and as any product unit q is decomposable we can simply push the summation to the input unit r of q for which we have $X_\mu \in \phi(r)$. For summation units we exploit the linearity of the integral and distribute the integral over the terms in the sum. Performing this recursively brings the integral to the leaves where we have $\int f_i(x_\mu \mid x_{pa(\mu)}) \mathrm{d}x_\mu = 1$.

Up to this point the marginalization in CPC is identical to marginalization in probabilistic circuits. Contrary, to probabilistic circuits, however, we now need to propagate back up the result of the marginalization. Assuming, without loss of generality that leaf units feed into sum units, we then have

$$\sum_{g \in \text{in}(h)} \omega_{hg}(\mathbf{x}_{an(\mu)}) \int p_g(x_\mu) \mathrm{d}x_\mu = \sum_{g \in \text{in}(h)} \omega_{hg}(\mathbf{x}_{an(\mu)}) = 1$$

The next product node that we encounter on our way up through the circuit is of the form:

$$\begin{aligned} &p_l(\mathbf{x}_{\mu-1} \mid \mathbf{x}_{pa(\mu-1)}) \int p_r(\mathbf{x}_\mu \mid \mathbf{x}_{pa(\mu)}) \mathrm{d}\mathbf{x}_\mu \\ &= p_l(\mathbf{x}_{\mu-1} \mid \mathbf{x}_{pa(\mu-1)}) \end{aligned}$$

At this point we have integrated out the variable x_μ from the circuit by traversing a number of units linear in the size of the circuit (assuming proper caching (Vergari et al. 2021)). Repeating this procedure for the remaining set of ordered random variables $\{X_{\mu-1}, \dots, X_1\}$ gives us the distribution $p_k(\mathbf{x}_e)$ in polynomial time. □

The proof follows a similar reasoning to the case of probabilistic circuits. The delicate point was to show that in the product unit one of the factors drops out. This is important as $X_{pa(\mu)}$ might include $X_{\mu-1}$. Retaining such a dependency would prevent us from performing tractable ordered marginalization.

Corollary 3.14 (Ordered Conditionals)**.** *Assuming a PNC $p_k(\mathbf{x}_m, \mathbf{x}_o, \mathbf{x}_e)$ where $\mathbf{X}_e \sqsubset \mathbf{X}_o \sqsubset \mathbf{X}_m$ holds lets us compute the conditional $p(\mathbf{x}_o \mid \mathbf{x}_e)$ in polynomial time.*

Proof. We first apply the definition of the conditional probability: $p(\mathbf{x}_o \mid \mathbf{x}_e) = {}^{p(\mathbf{x}_o, \mathbf{x}_e)}/_{p(\mathbf{x}_e)}$. Using the law of total probability we rewrite the ratio as

$$p(\mathbf{x}_o \mid \mathbf{x}_e) = \frac{\int p(\mathbf{x}_m, \mathbf{x}_o, \mathbf{x}_e) \mathrm{d}\mathbf{x}_m}{\int p(\mathbf{x}_m, \mathbf{x}_o, \mathbf{x}_e) \mathrm{d}\mathbf{x}_m \mathrm{d}\mathbf{x}_o}$$

and Proposition 3.13 tells us that we can perform both marginalizations in polytime. □

4 Layered Probabilistic Neural Circuits

While Equation 7 provides a generic functional expression to compute the sum units in PNCs, it is not clear how to construct a PNC in the first place. That is, how do we link up the individual computation units such that they form a (valid) CPC. For (non-conditional) probabilistic circuits potent structure learning algorithms have been developed in recent years (e.g. hidden Chow-Liu trees (Liu and Van den Broeck 2021) or random probabilistic circuits (Di Mauro et al. 2021)). It is not entirely clear how to adapt these to the setting of conditional probabilistic circuits. For this reason we study problems where, informed by the structure of the data itself, a structure for a PNC can be constructed. Concretely, we will study PNC structures tailored towards image data: features, i.e. pixels, that are close to each other should also be close to

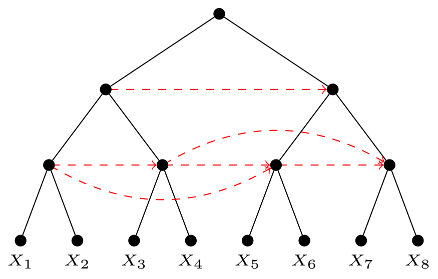

Figure 3: A balanced partition tree of a probabilistic neural circuit with eight variables. The partition tree (in black) describes how the variables decompose (in terms of the scope function). The edges in red indicate functional (neural) dependencies between partitions.

each other in the circuit – a fact already exploited by Poon and Domingos (2011).

The structure we propose in this paper is rather simple and inspired by simple feed-forward neural networks and also layered (non-conditional) probabilistic circuits (Peharz et al. 2020). More concretely, the computation units in the current layer only depend on computation units in the previous layer. Furthermore, we wish the units within each layer to be computable in parallel (given the previous layer). To construct such a probabilistic neural circuit we take the circuit structure introduced by Shih, Sadigh, and Ermon (2021) as a backbone and add additional edges to the computation graph in order to obtain a PNC from a probabilistic circuit. For ease of exposition we detail our approach using one-dimensional data instead of two-dimensional data.

4.1 Structure for One-Dimensional Data

In order to study PNC structures, we introduce the concept of a partition graph, which is a hypergraph of a probabilistic circuit with the partitions of a circuit being the nodes and edges encoding the sub-partitions[2]. We give an example of a partition tree in Figure 3.

Ignoring for now neural dependencies in PNCs (i.e. the red edges in the partition tree in Figure 3) we describe the layer-wise operations. Let us assume, for the sake of simplicity, that the number of variables N is a power of 2. For instance, the circuits in Figure 1 has $N = 4$ and the partition diagram in Figure 3 has $N = 8$. Given that we merge partitions pairwise at each layer via multiplication, we obtain for the number of layers in a circuit $N_L = \log_2 N + 1$. Formally, we express the layer-wise product units as follows:

$$\kappa_{l,p,1,c} = \kappa_{l-1,2(p-1)+1,2,c} \times \kappa_{l-1,2(p-1)+2,2,c}, \tag{ProductLayer}$$

[2]In the circuit structure introduced by Shih, Sadigh, and Ermon (2021), all the product nodes decompose in the same fashion, i.e. for product nodes with the same scope it is the sames variables that come from the left and right inputs, respectively. This is also called structured decomposabiltiy (Darwiche 2011). Partition trees are related to the concept of *variable trees* in *(probabilistic) sentential decision diagrams* (Darwiche 2011; Kisa et al. 2014). However, nodes in a variable tree do not constitute abstractions of a specific group of sum and product units and are a more general concept.

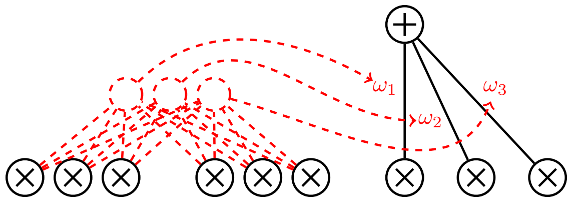

Figure 4: Detailed graphical representation of neural dependencies in a PNC. The sum unit at the top outputs the weighted sum of the three product units at the bottom right. The weights for the sum are the outputs of a neural network for which it holds that $\sum_{i=1}^{3} = 1$. They are computed using a neural network that takes as input the values of the six product units at the bottom left.

which describes how to compute the value of a component given the components of the previous layer. The meaning of the indices is described in Section 2. Using this notation we can also express the sum units at the leaves and at the root:

$$\kappa_{1,p,2,c} = \sum_{c'=1}^{N_D} w_{1,p,c,c'} \times \kappa_{1,p,1,c'} \tag{SumLeaf}$$

$$\kappa_{N_L,1,2,1} = \sum_{c'=1}^{N_C} w_{N_L,1,1,c'} \times \kappa_{N_L,1,1,c'} \tag{SumRoot}$$

In the equations above, N_D denotes the number of initial components in the leaves and N_C is the number of components throughout the circuits. For the circuit in Figure 1 we have N_D=2 and N_C=3. The weights $w_{l,p,c.c'}$ are real valued constants and normalized over the c' dimension. Note that, in contrast to the formulation in Section 3, weights (and also computation units in general) are not identified by a single index (e.g. k in $p_k(\cdot)$) but by four indices. To make this distinction explicit we denote the computation units by $\kappa_{l,p,i,c}$ instead of $p_k(\cdot)$. Observe also that the root layer has only a single component. Hence, the 1 as the last index instead of c.

Next, we describe the neural sum units in layered PNCs. To gain some intuition, consider the partial computation graph in Figure 4, where we show in more detail, compared to Figures 1 and 3, the neural dependencies present at the sum units. Formally, we express the values of neural sum units as follows:

$$\kappa_{l,p,2,c} = \sum_{c'=1}^{N_C} \omega_{l,p,c,c'}(\kappa_{l,p-\nu_{l,p}:p-1,1,1:N_C}) \times \kappa_{l,p,1,c'} \tag{NeuralSumLayer}$$

Here, $\omega_{l,p,c,c'}(\cdot)$ denotes a neural network and N_C denotes again the number of components. The input to the neural net are the values of the set of units $\kappa_{l,p-\nu_{l,p}:p-1,1,1:N_C}$. The notation 1:$N_C$ denotes the range of components $[1, \dots, N_C]$ (including the first and last element). Similarly, $p-\nu_p$:$p-1$ denotes the range of partitions $[p-\nu_{l,p}, \dots, p-1]$. The parameter $\nu_{l,p}$ is a hyperparameter and describes how many partitions *'to the left'* are taken into consideration when computing the neural weights. For instance, in the partial computation

Algorithm 1: Layer-wise circuit evaluation

```
    Input: $\mathbf{x}_o$. $\mathbf{X}_m$
    Output: $p(\mathbf{x}_o)$
    Require: $\mathbf{X}_o \cup \mathbf{X}_m = \mathbf{X}$, $\mathbf{X}_o \sqsubset \mathbf{X}_m$
 1: $\kappa_{p,c} \leftarrow init(p, c, \mathbf{x}_o)$
 2: $\kappa_{p,c} \leftarrow \sum_{c'=1}^{N_D} w_{1,p,c,c'} \times \kappa_{p,c'}$            ▷ LeafLayer
 3: $l \leftarrow 2$
 4: while $l < N_L$ do
 5:     $\kappa_{p,c} \leftarrow \kappa_{2p,c} \times \kappa_{2p+1,c}$            ▷ ProductLayer
 6:     $\kappa_{p,c} \leftarrow \sum_{c'=1}^{N_C} \omega_{l,p,c,c'} \times \kappa_{p,c'}$   ▷ NeuralSumLayer
 7:     $l \leftarrow l + 1$
 8: end while
 9: $\kappa_{p,c} \leftarrow \kappa_{2p,c} \times \kappa_{2p+1,c}$                ▷ ProductLayer
10: $\kappa_{1,1} \leftarrow \sum_{c'=1}^{N_C} w_{N_L,1,c,c'} \times \kappa_{1,c'}$            ▷ RootSum
11: return $\kappa_{1,1}$
```

graph in Figure 4, we have $\nu_{l,p}$=2. When setting $\nu_{l,p}$=0 we recover the special case of layered (non-conditional) probabilistic circuits.

Observe that it is those *intra-layer* neural dependencies that induce an order on the random variables: we can only perform the computations of the units in the right branch of a partition tree if the values in the left branch are known. This holds recursively.

Definition 4.1. *Given a poset of random variables $\mathcal{O}(\mathbf{X})$, we call a neural sum layer valid if the (partial) variable order it induces respects $\mathcal{O}(\mathbf{X})$.*

Using the equations above to compute the leaf, product, sum, and root layers, we can also write down the pseudocode for marginal inference in layered PNCs, which we give in Algorithm 1. For ease of exposition we assume again that the number of variables $\mathbf{X}$ is a power of two, and we refer to our implementation for the general case[3].

The algorithm takes as input a set of random variable instantiation $\mathbf{x}_o$ and a set of random variables $\mathbf{X}_m$. The latter ought to be marginalized out, and the circuit evaluation computes the probability $p(\mathbf{x}_o)$.

The first step is to initialize the $\kappa_{p,c}$, which we do with the following function:

$$init(p, c, \mathbf{x}_o) = \begin{cases} f_{p.c}(x) & \text{if } x \in \mathbf{x}_o \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

Each combination of p and c corresponds to an index k in Equation 2. Note that we forego in Algorithm 1 the possibility of introducing conditional dependencies in the leaves. The 1's in the second case result from marginalizing out the probability distributions in the leaves for the variables in $\mathbf{X}_m$.

The algorithm then proceeds by first performing the computations in the leaf, before looping through the internal sum and product layers, and finishes with computing the root layer. In contrast to the 4-index notation introduced in Section 2 we only use two indices here. This is because the identification of the layers happens implicitly using the l counter of the while loop.

[3]https://github.com/pedrozudo/ProbabilisticNeuralCircuits.git

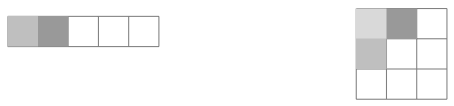
Figure 5: Graphical representation of *half kernels* used for neural sum layers in layered PNCs. On the left we see a kernel used for one-dimensional data while on the right we have a 3×3 kernel for two-dimensional data. The gray blocks indicate the learnable parameters of the *half kernels*, while a white square indicates a parameter fixed to zero. Effectively, the convolutional layer is blind with regard to the inputs for these zero elements of the kernel.

4.2 Implementation Using Convolutions

An efficient way of implementing neural sum layers is by means of convolutional neural networks. We can readily see this if we interpret the components within a layer as the channels of the convolutional neural network and the partitions as the input dimension over which we perform the convolution.

Assuming that our one-dimensional data is ordered left to right, we can make sure that this order is respected throughout the neural sum layers by using convolutional networks with a *half kernel* as depicted in the left of Figure 5. The convoluted channels can then be passed on to further layers. As a final activation function we use a softmax layer that normalizes the outputs such that they sum up to 1, per partition that is. Importantly, the number of output channels has to match the number of components in the summation.

5 Related Work

A first attempt at relaxing decomposability in probabilistic circuits was made by Sharir and Shashua (2018) with the introduction of sum-product-quotient networks (SPQNs). SPQNs introduce the quotient unit (in addition to sum and products), which encode conditional probabilities within a circuit. We show that introducing extra units is unnecessary as the same effect can be obtained by generalizing sum and product units while retaining two types of computation units.

On a theoretical level this allows us to forego the introduction of expendable concepts such as conditional soundness (Sharir and Shashua 2018, Definition 5) or conditional and effective scopes of a unit (Sharir and Shashua 2018, Section 3).

On the practical side we will show in Section 6 that neural sum layers outperform the *conditional mixing operator* (CMO) proposed as a building block for SPQNs (Sharir and Shashua 2018, Definition 6). As a matter of fact, PNCs strictly generalize SPQNs constructed with CMOs.

Corollary 5.1. *CMO-SPQNs are PNCs.*

Proof. This can trivially be shown by picking $\omega(\cdot)$ such that

$$\kappa_{l,p,2,c} \qquad \text{(QuotientSumLayer)}$$
$$= \frac{\sum_{c'=1}^{N_C} w_{l,p,c,c'} \times \left(\prod_{\kappa' \in \kappa_{l,p-\nu_{l,p}:p-1,1,c'}} \kappa'\right) \times \kappa_{l,p,1,c'}}{\sum_{c'=1}^{N_C} w_{l,p,c,c'} \times \left(\prod_{\kappa' \in \kappa_{l,p-\nu_{l,p}:p-1,1,c'}} \kappa'\right)}$$

where we use our notation from Section 3.3 to write down the CMO of Sharir and Shashua (2018). □

Guided by Corollary 5.1 we implemented CMO-SPQNs using a convolutional layer by simply fixing the non-zero elements of the kernel (cf. Figure 5) to one. Performing the convolution on probabilities in log-space then simply corresponds to multiplying them in linear space. In this fashion we easily obtain the product present in the quotient sum layer. An important difference between CMO-SPQNs and PNCs is that for the former components (or channels) must not mix with each other, which hinders their performance in terms of function approximation.

The limited expressive power of sum units was also noted by Shao et al. (2022), which led them to introduce *conditional sum-product networks*. The idea is to condition the weights of the sum units in a probabilistic circuit on the value of a random value (using neural networks). The main difference to our work is that these random variables live out-side of the circuits itself. Speaking in terms of Bayesian networks the approach of Shao et al. (2022) is only capable of encoding a single Bayesian network while CPC encode hierarchical mixtures of Bayesian networks.

6 Experimental Evaluation

For our experimental evaluation we used the MNIST family of dataset. That is, the original MNIST (Deng 2012), FashionMNIST (Xiao, Rasul, and Vollgraf 2017), and also EMNIST (Cohen et al. 2017). We implemented PNCs (and also SPQNs) in PyTorch and Lightning[4], and ran all our experiments on a DGX-2 machine with V100 Nvidia cards.

6.1 How Do PNCs Fair Against PQCs and PCs?

Setup To answer the first question we trained a PNC, an SPQNs and a PC on the MNIST family of datasets and minimize the negative log-likelihood. In order to render the three models commensurable, all three have the same underlying circuit structure. That is, we start with a grid of 28×28 pixels and merge, in an alternating fashion, rows and columns. This corresponds to product nodes in the circuits. Between each merge we perform a summation. For probabilistic (sum) circuits (PSCs) this is the usual sum unit, for probabilistic quotient circuits (PQCs or SPQNs) we use the conditional mixing operator, and for PNCs we use a neural sum layer. The kernel type used for PQCs and PNCs is the one depicted in the right of Figure 5. For the leaves, which encode the 28×28 pixels, we use one categorical distribution with 256 categories for each pixel. This allows us to represent all possible pixel values. Within the circuits we used 12 components per partition. Merging rows and columns in an alternating fashion and using the kernel from Figure 5 induces a specific variable ordering on a 2-dimensional grid, which we graphically represent in Figure 6.

All three models were trained for 100 epochs using Adam (Kingma and Ba 2014) with a learning rate of 0.001 and a batch size of 50. The best model was selected using a $90 - 10$ train-validation data split where we monitored the negative log-likelihood on the validation set. We refer to the configuration files of the experiments for more details.

[4]https://lightning.ai/

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

Figure 6: For simplicity's sake, assume that we have a 4×4 grid of pixels (instead of the 28×28 MNIST grid). Recursively merging rows and columns and using the kernel from Figure 5 then gives us the total pixel order as indicated in the grid. We can then marginalize out pixels with higher numbers before pixels with lower numbers. For the specific case here we could marginalize out the lower half of the 4×4 image, while retaining a probability distribution for the upper half.

Results We compare the three architectures using *bits per dimension*, which are calculated from the average negative log-likelihood ($\overline{NLL}$) as follows: $bpd = \overline{NLL}/(\log 2 \times D)$, here $D = 28^2$ for MNIST datasets.

The results are reported in Table 1 in the first three columns. We see that quotient circuits outperform sum circuits when it comes to minimizing the negative log-likelihood (minimizing *bpd*). We also see that neural circuits, with their data dependent weights, outperform both other methods. Note that PNCs and PQCs allow for the same set of tractable queries, while PNCs are more performant.

6.2 How Do PNCs Fair Against State of the Art?

Setup We use again the same PNCs as in Section 6.1 and compare them to (decomposable) probabilistic circuits from the literature: hidden Chow-Liu trees (HCLT) (Liu and Van den Broeck 2021), sparse HCLT (SHCLT) (Dang, Liu, and Van den Broeck 2022), random sum-product networks (RAT-SPN) (Peharz et al. 2019), and continuous mixture circuits (CMC) (Correia et al. 2023). For completeness, we also include the bpd for IDF (a flow-based approach) (Hoogeboom et al. 2019) and for BitSwap (a hierarchical latent variable model) (Kingma, Abbeel, and Ho 2019). Note that the results reported for RAT-SPN were taken from (Dang, Liu, and Van den Broeck 2022).

Results We see again that PNCs outperform the other methods in terms of bpd. On the one hand this is due to the increased expressive power of PNCs obtained by relaxing decomposability (and thereby losing tractability). On the other hand, this is also due to the fact that PNCs use neural sum units. This can be seen by comparing the results for PQC and SHCLT. While PQC are in theory more expressive than SHCLT we don't see this in practice: the learned structure of SHCLTs overcomes this expressivity gap. However, we see that PNCs manage to outperform SHCLTs. Interestingly our implementation of PCs (PSC) does produce lower bpd than RAT-SPNs, suggesting that the latter are a rather weak baseline when it comes to density estimation on image data.

	PNC	PQC	PSC	SHCLT	HCLT	CMC	RAT-SPN	IDF	BitSwap
MNIST	**0.87**	1.20	1.32	1.14	1.20	1.28	1.67	1.90	1.27
FashionMNIST	**2.51**	3.47	3.66	3.27	3.34	3.55	4.29	3.47	3.28
EMNIST (mnist)	**1.36**	1.84	2.07	1.52	1.77	–	2.56	2.07	1.88
EMNIST (letters)	**1.33**	1.83	2.07	1.58	1.80	–	2.73	1.95	1.84
EMNIST (balanced)	**1.35**	1.86	2.16	1.60	1.82	–	2.78	2.15	1.96
EMNIST (byclass)	**1.27**	1.76	2.02	1.54	1.85	–	2.72	1.98	1.87
# parameters	$2.8M$	$2.6M$	$2.6M$	$7.0M$	$7.0M$	$0.1M$	$7.0M$	$24.1M$	$2.8M$

Table 1: Test set bpd for MNIST datasets (lower is better). The last row shows the number of parameters for each model (the symbol M stands for millions).

6.3 Can PNCs Perform Discriminative Learning?

Setup Assume that we have a data point $\mathbf{x}$ for the random variables in $\mathbf{X}$ and a label y for this data point. Using Bayes rule we can rewrite the discriminative probability using generative distributions: $p(Y{=}y|\mathbf{X}{=}\mathbf{x}){=}\frac{p(x|y)}{\sum_{z\in\Omega(Y)} p(x|y)}$, where we assume that the class prior is identical for each of the classes belonging to the sample space $\Omega(Y)$. This means that for every class in $\Omega(Y)$ we have a separate distribution. That is, a separate circuit. In the case of MNIST and FashionMNIST we have ten classes. The resulting ten circuits are jointly optimized using cross-entropy (Peharz et al. 2019). Architecturally, we used again 12 components per partition and the 10 different circuits share all parameters but the parameters in the leaf layer and root layer.

Furthermore, instead representing pixels as categorical random variables (with sample space $\{0, \ldots, 255\}$) we model them as continuous random variables with samples belong to the interval $[0, 1]$. We obtain this value v by dividing the pixel by 255. Each leaf has then two inputs: the corresponding value v of of the pixel itself and $1 - v$. This follows the protocol of (Liang and Van den Broeck 2019). Apart from optimizing the cross-entropy instead of the log-likelihood the training protocol was identical to the one for density estimation.

Results We report the comparison in terms of classification accuracy on the test set, which we show in Table 2. We compare PNCs and PQCs to logistic circuits (LCs) (Liang and Van den Broeck 2019) and RAT-SPNs (Peharz et al. 2019). We see that PNCs perform better than PQC. However, neither reaches the accuracies of LCs nor RAT-SPNs. We hypothesize that this is due to a lack of regularization. For instance, PNCs reach perfect train accuracy on MNIST and near perfect train accuracy on FashionMNIST. Furthermore, the authors of LCs and RAT-SPNs reported having used aggressive regularization techniques – for the former on their Github page[5] and the latter in (Peharz et al. 2019, Section 4.2). While we experimented with various regularization techniques such as weight decay (Loshchilov and Hutter 2018) or the stochastic delta rule (Hanson 1990), we were not able to obtain consistent improvements. We leave the study of effective regularization techniques for discriminative learning with PNCs for future work.

[5]https://github.com/UCLA-StarAI/LogisticCircuit

	PNC	PQC	LC	RAT-SPN
MNIST	98.04	97.38	99.4	98.29
FashionMNIST	88.84	87.63	91.3	89.89

Table 2: Test accuracies for MNIST and FashionMNIST.

7 Conclusions & Future Work

We first introduced the concept of a conditional probabilistic circuit, from which we were then able to construct probabilistic neural circuits, which generalize probabilistic circuits and sum-product-quotient networks. Note that the construction of PNCs would not have been possible from the formulation of Sharir and Shashua (2018). Furthermore, our formulation allows us to intuitively interpret PNCs as neural approximations of deep mixtures of Bayesian networks.

Experimentally, we have shown that for density estimation PNCs deliver on the promise made by SPQNs. That is, giving up on tractability improves function approximation in practice. For the discriminative case the situation is more nuanced. While PNCs achieve perfect accuracy on the training set a lack of proper regularization techniques prevents them from matching accuracies obtained by competing methods. We would also like to note that more sophisticated architecture designs for PNCs could possibly further improve their performance. For instance, using different numbers of components per partition dramatically increased the performance of SHCLT when compared to HCTL.

In future work we would like to explore the potential of PNCs for sampling, a task that probabilistic models usually struggle with (Lang et al. 2022) as good likelihood estimates do not correlate with sample quality (Theis, van den Oord, and Bethge 2016). This would also establish a tighter link to autoregressive models (ARM) (e.g. PixelCNN (Van den Oord et al. 2016)) and we might use ideas developed there for PNCs. In this regard, any-order ARM (Uria, Murray, and Larochelle 2014; Shih, Sadigh, and Ermon 2022) seem to be of particular interest to PNCs as this could allow for arbitrary conditioning sets in PNCs.

Other open questions concern structure learning for PNCs and applying them to tabular data or finding applications of PNCs – an obvious candidate would be lossless compression with circuits (Liu, Mandt, and Van den Broeck 2022), as any-order marginalization is not necessary.

Acknowledgements

This project received funding from the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg-Foundation, as well as from the TAILOR Connectivity Fund (part of the TAILOR project funded by the EU Horizon 2020 research and innovation program under GA No 952215). The author would like to thank the anonymous reviewers for the valuable comments.

References

Cohen, G.; Afshar, S.; Tapson, J.; and Van Schaik, A. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks*.

Correia, A. H.; Gala, G.; Quaeghebeur, E.; de Campos, C.; and Peharz, R. 2023. Continuous mixtures of tractable probabilistic models. In *AAAI Conference on Artificial Intelligence*.

Dang, M.; Khosravi, P.; Liang, Y.; Vergari, A.; and Van den Broeck, G. 2021. Juice: A julia package for logic and probabilistic circuits. In *AAAI Conference on Artificial Intelligence*.

Dang, M.; Liu, A.; and Van den Broeck, G. 2022. Sparse probabilistic circuits via pruning and growing. In *Advances in Neural Information Processing Systems*.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4): 608–647.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3): 280–305.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Delalleau, O.; and Bengio, Y. 2011. Shallow vs. deep sum-product networks. *Advances in neural information processing systems*, 24.

Deng, L. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine*.

Di Mauro, N.; Gala, G.; Iannotta, M.; and Basile, T. M. 2021. Random probabilistic circuits. In *Uncertainty in Artificial Intelligence*.

Hanson, S. J. 1990. A stochastic version of the delta rule. *Physica D: Nonlinear Phenomena*, 42(1-3): 265–272.

Hoogeboom, E.; Peters, J.; Van Den Berg, R.; and Welling, M. 2019. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems*.

Kileel, J.; Trager, M.; and Bruna, J. 2019. On the expressive power of deep polynomial neural networks. *Advances in neural information processing systems*, 32.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, F.; Abbeel, P.; and Ho, J. 2019. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Lang, S.; Mundt, M.; Ventola, F.; Peharz, R.; and Kersting, K. 2022. Elevating perceptual sample quality in PCs through differentiable sampling. In *NeurIPS 2021 workshop on pre-registration in machine learning*. PMLR.

Liang, Y.; and Van den Broeck, G. 2019. Learning logistic circuits. In *AAAI Conference on Artificial Intelligence*.

Liu, A.; Mandt, S.; and Van den Broeck, G. 2022. Lossless Compression with Probabilistic Circuits. In *International Conference on Learning Representations*.

Liu, A.; and Van den Broeck, G. 2021. Tractable regularization of probabilistic circuits. In *Advances in Neural Information Processing Systems*.

Loshchilov, I.; and Hutter, F. 2018. Fixing weight decay regularization in adam.

Martens, J.; and Medabalimi, V. 2014. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*.

Peharz, R.; Lang, S.; Vergari, A.; Stelzner, K.; Molina, A.; Trapp, M.; Van den Broeck, G.; Kersting, K.; and Ghahramani, Z. 2020. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*.

Peharz, R.; Tschiatschek, S.; Pernkopf, F.; and Domingos, P. 2015. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*.

Peharz, R.; Vergari, A.; Stelzner, K.; Molina, A.; Shao, X.; Trapp, M.; Kersting, K.; and Ghahramani, Z. 2019. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*.

Poon, H.; and Domingos, P. 2011. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops*. IEEE.

Shao, X.; Molina, A.; Vergari, A.; Stelzner, K.; Peharz, R.; Liebig, T.; and Kersting, K. 2022. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140: 298–313.

Sharir, O.; and Shashua, A. 2018. Sum-product-quotient networks. In *International Conference on Artificial Intelligence and Statistics*.

Shih, A.; Sadigh, D.; and Ermon, S. 2021. Hyperspns: Compact and expressive probabilistic circuits. *Advances in Neural Information Processing Systems*.

Shih, A.; Sadigh, D.; and Ermon, S. 2022. Training and Inference on Any-Order Autoregressive Models the Right Way. *Advances in Neural Information Processing Systems*.

Theis, L.; van den Oord, A.; and Bethge, M. 2016. A note on the evaluation of generative models. In *International Conference on Learning Representations*.

Uria, B.; Murray, I.; and Larochelle, H. 2014. A deep and tractable density estimator. In *International Conference on Machine Learning*.

Van den Oord, A.; Kalchbrenner, N.; Espeholt, L.; Vinyals, O.; Graves, A.; et al. 2016. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems*.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Zhang, H.; Juba, B.; and Van den Broeck, G. 2021. Probabilistic generating circuits. In *International Conference on Machine Learning*.

Zhao, H.; Poupart, P.; and Gordon, G. J. 2016. A unified approach for learning the parameters of sum-product networks. *Advances in neural information processing systems*.