International Journal of Approximate Reasoning 140 (2022) 298–313

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Conditional sum-product networks: Modular probabilistic circuits via gate functions

Xiaoting Shao [a,*], Alejandro Molina [a], Antonio Vergari [c], Karl Stelzner [a], Robert Peharz [d], Thomas Liebig [e], Kristian Kersting [a,b]

[a] CS Department, TU Darmstadt, Germany
[b] Centre for Cognitive Science, TU Darmstadt, Germany
[c] CS Department, University of Edinburgh, UK
[d] TU Graz, Austria
[e] Materna SE, Dortmund, Germany

A R T I C L E   I N F O

Article history:
Received 30 January 2021
Received in revised form 14 October 2021
Accepted 20 October 2021
Available online 4 November 2021

Keywords:
Conditional probabilistic modeling
Probabilistic circuits
Sum-product-networks
Tractable inference
Variational inference

A B S T R A C T

While probabilistic graphical models are a central tool for reasoning under uncertainty in AI, they are in general not as expressive as deep neural models, and inference is notoriously hard and slow. In contrast, deep probabilistic models such as sum-product networks (SPNs) capture joint distributions and ensure tractable inference, but still lack the expressive power of intractable models based on deep neural networks. In this paper, we introduce conditional SPNs (CSPNs)—conditional density estimators for multivariate and potentially hybrid domains—and develop a structure-learning approach that derives both the structure and parameters of CSPNs from data. To harness the expressive power of deep neural networks (DNNs), we also show how to realize CSPNs by conditioning the parameters of vanilla SPNs on the input using DNNs as gate functions. In contrast to SPNs whose high-level structure can not be explicitly manipulated, CSPNs can naturally be used as tractable building blocks of deep probabilistic models whose modular structure maintains high-level interpretability. In experiments, we demonstrate that CSPNs are competitive with other probabilistic models and yield superior performance on structured prediction, conditional density estimation, auto-regressive image modeling, and multilabel image classification. In particular, we show that employing CSPNs as encoders and decoders within variational autoencoders can help to relax the commonly used mean field assumption and in turn improve performance.

1. Introduction

Probabilistic models are a fundamental approach in machine learning to represent and distill meaningful representations from data with inherent structure. In practice, however, it has been challenging to come up with probabilistic models that balance three desirable goals:

* Corresponding author.
E-mail addresses: xiaoting.shao@cs.tu-darmstadt.de (X. Shao), molina@cs.tu-darmstadt.de (A. Molina), avergari@exseed.ed.ac.uk (A. Vergari), stelzner@cs.tu-darmstadt.de (K. Stelzner), robert.peharz@tugraz.at (R. Peharz), thomas.liebig@cs.tu-dortmund.de (T. Liebig), kersting@cs.tu-darmstadt.de (K. Kersting).

https://doi.org/10.1016/j.ijar.2021.10.011
0888-613X/© 2021 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

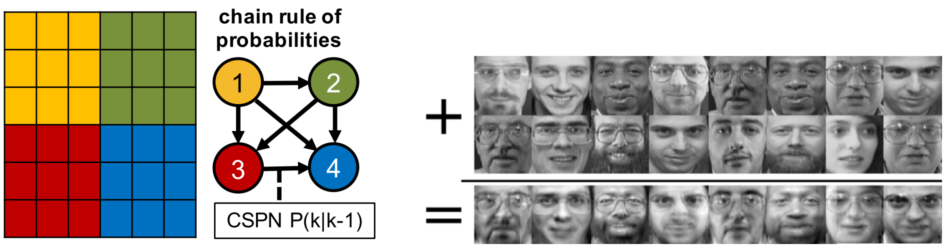


Fig. 1. Imposing structure on deep probabilistic architectures. (Left) An Autoregressive Block-wise CSPN (ABCSPN) factorizes a distribution over images along image patches. The first block is conditioned on a set of semantic labels, which is not shown in the figure. (Right) Olivetti faces generated by an ABCSPN. The new faces (bottom row) were sampled after conditioning on a set of class images (top and middle row) that is the mixing of two original classes in the Olivetti dataset. The generated images resemble the prominent features of both their parent images.

1. sufficient expressivity to capture the complexity of real-world data;
2. maintain—at least on a high level—interpretable domain structure; and
3. permit a rich set of tractable inference routines.

Probabilistic graphical models (PGMs), for example, admit an interpretable structure, but are known to achieve a bad trade-off between expressivity and tractable inference [1]. On the other hand, probabilistic models based on deep neural networks such as variational autoencoders (VAEs) [2] and generative adversarial networks (GANs) [3] are highly expressive, but generally lack an interpretable structure and have limited capabilities when it comes to probabilistic inference. Meanwhile, probabilistic circuits such as arithmetic circuits [4] and sum-product networks (SPNs) [5] allow for flexible density estimation with tractable inference, but, like neural networks, lack an easy-to-interpret structure.

Consequently, it is natural to ask the question of how we can combine these lines of research, and strive for a sensible balance between expressivity, structure, and tractable inference? This question has received surprisingly little attention. For example, Johnson et al. [6] combined VAEs with PGMs, and thus effectively equip expressive neural-based models with a rich structure, and Tan and Peharz [7] employed VAEs as leave distributions to probabilistic circuits. Unfortunately, inference remains intractable in these models. In contrast, Shen et al. [8] proposed structured Bayesian networks, which impose structure among clusters of variables using probabilistic sentential decision diagrams (PSDDs) to model high-dimensional conditional distributions [9]. While this yields well-structured models with a wide range of tractable inference scenarios, they are restricted to binary data and do not establish a link to deep neural networks. Moreover, PSDDs are more restricted than SPNs. Recently and independently of us, Rahman et al. [10] proposed conditional cutset networks, a strict sub-class of SPNs, but they neither employ conditional independence tests for learning the structure, nor do they link to deep neural networks.

In this paper, we introduce conditional sum-product networks (CSPNs), a conditional variant of SPNs, which can harness the expressive power of universal function approximators such as neural networks, while maintaining a wide range of tractable inference routines. Specifically, we formally define CSPNs, provide a learning framework for them, and provide arguments for why CSPNs are more compact than SPNs to express conditional distributions. In addition, we propose *neural CSPNs*, which rely on (random) SPN structures, parameterized by gate functions, e.g., deep neural networks. While neural CSPNs do not have the benefit of carefully learned structures, they gain expressiveness through increased model capacity. In particular, CSPNs can naturally be used as (conditional) modules of generative models and impose a rich structure on high-dimensional joint distributions. We illustrate this by introducing autoregressive SPNs for conditional image generation, modeling images block by block, decomposing the joint image distribution into a product of (C)SPNs, cf. Fig. 1 (left). This model can be used to generate images of a specific category, cf. Fig. 1 (right). Finally, we employ CSPNs as variational inference network that is optimized together with a generative model end-to-end in an auto-encoding scheme. Unlike vanilla VAE [11] where the mean field assumption is made, CSPNs allow one to capture dependencies between latent variables. The improvement over vanilla VAE is illustrated empirically across several image datasets.

To summarize, we make the following contributions:

1. We introduce CSPNs as a deep tractable model for modeling multivariate, conditional probability distributions $P(\mathbf{Y}\,|\,\mathbf{X})$ over mixed variables $\mathbf{Y}$, by introducing gating nodes as mixtures with functional mixing coefficients.
2. We present a structure learning algorithm for CSPNs based on randomized conditional correlation tests (RCoT) which allows learning structures from heterogeneous data sources.
3. We connect CSPNs with deep neural networks, and demonstrate that the improved ability of the resulting neural CSPNs to model dependencies results in increased multilabel image classification performance.
4. We illustrate how CSPNs may be used to build deep structured probabilistic models by introducing Autoregressive Block-wise CSPNs (ABCSPNs).
5. We propose a differentiable sampling routine for CSPNs, which can be easily extended to SPNs as well.

6. We demonstrate that using CSPNs as variational inference network, to account for dependencies between output variables, results in tighter ELBO.

A preliminary version of this manuscript has been published as a conference paper [12]. The present paper significantly extends the conference version by providing more details and introducing differentiable sampling for CSPNs and Sum-Product Variational Auto-Encoders (SPVAE). In particular, the algorithms for learning structure of CSPNs are presented in pseudocode. In addition, we demonstrate how to use CSPNs for amortized posterior inference in auto-encoding context, which yields SPVAE. To train SPVAE via Monte Carlo estimation, we propose also differentiable sampling procedure for CSPNs.

We proceed as follows. We start off by introducing SPNs, the generative counterpart of CSPNs, in section 2. Afterwards, we introduce our new conditional probabilistic model in section 3, in which related work of conditional probabilistic modeling is discussed and definition of CSPNs as well as the learning procedure are presented. Then we show in section 4 how to use CSPNs as modular blocks for probabilistic modeling to impose a high-level structure explicitly. Before concluding, we touch upon our empirical evaluation of CSPNs in Section 5.

2. (Unconditional) sum-product networks

In the following, we denote random variables (RVs) as upper-case letters, e.g., V, their values as lower-case letters, e.g., $v \sim V$, and sets of RVs in bold, e.g., $\mathbf{v} \sim \mathbf{V}$. We employ $\mathbf{Y} \subset \mathbf{V}$ to denote the target RVs, also called labels, while we denote the disjoint set of observed RVs, the features, as $\mathbf{X} := \mathbf{V} \setminus \mathbf{Y}$.

Before introducing conditional sum-product networks (CSPNs), we briefly review classical (unconditional) SPNs. For further details on SPNs, we refer to [4,5,13].

Definition 1. A sum-product network (SPN) over a set of random variables $\mathbf{V}$ is defined via a directed acyclic graph (DAG) containing three types of nodes: distribution nodes, sum nodes and product nodes. All leaves of the DAG are distribution nodes, and all internal nodes are either sums or products.

An SPN leaf represents a univariate distribution $P(Y)$ for some RV $Y \in \mathbf{V}$. A sum node represents a *mixture* $\sum_k w_k P_k(\mathbf{Y})$, where P_k are the children of the sum node according to the DAG, and w_k satisfy $w_k \geq 0$ and $\sum_k w_k = 1$. We require that sum nodes are *complete*, i.e. that all children are defined over the same *scope* $\mathbf{Y}$. A product node represents a *factorization* $\prod_k P_k(\mathbf{Y}_k)$, where P_k are the children of the product node according to the DAG. We require that product nodes are *decomposable*, i.e. that all children are defined over *disjoint scopes*, such that for any two product children $P_i(\mathbf{Y}_i)$ and $P_j(\mathbf{Y}_j)$, it holds that $\mathbf{Y}_i \cap \mathbf{Y}_j = \emptyset$.

In comparison to classical probabilistic graphical models that exploit only conditional independence assumptions, SPNs have the advantage that they can represent certain distributions much more succinctly by exploiting context-specific independence (CSI) [14], even if the corresponding graphical model would have very high tree-width [1]. Furthermore, *inference routines* such as *marginalization* and *conditioning* can be tackled in linear time w.r.t. the network size [4,5,15]. These tasks can also be *compiled*, i.e., starting from an SPN representing $P(\mathbf{V})$, it is possible to generate a new SPN representing the marginal distribution for an arbitrary $\mathbf{X} \subset \mathbf{V}$, potentially conditioned on event $\mathbf{Y} = \mathbf{y}$, for any $\mathbf{Y} \subset \mathbf{V}$, $\mathbf{Y} \cap \mathbf{X} = \emptyset$.

Indeed, one way to represent conditional distribution $P(\mathbf{Y} \,|\, \mathbf{X})$ is to train a regular SPN on the joint $P(\mathbf{V})$, and to then compile it into a conditional distribution for each input $\mathbf{x}$ of interest. It is clear, however, that this will generally deliver sub-optimal results, since the network will not be specialized to the specific input-output pair of $\mathbf{X}$ and $\mathbf{Y}$. Intuitively, training the full joint $P(\mathbf{V})$ optimizes *all* possible conditional distributions one can derive from the joint. Thus, when we are interested in learning a conditional distribution for a set of input and output variables $\mathbf{X}$ and $\mathbf{Y}$ known a-priori, directly learning a conditional distribution, as discussed in the following, can be expected to deliver better results.

3. A new conditional probabilistic model

Although SPNs show appealing properties at probabilistic inference, especially when the domain is high dimensional and complex, their representation does not maintain a high-level structure that is easily interpretable. Consider Structured Bayesian Networks (SBN) [16] where each node is a cluster of variables. This cluster DAG specifies conditional independencies between sets of variables, but makes no claims about the independencies between variables in the same cluster [8]. This modularity can help high-level understanding of the domain and integrating background knowledge or assumptions in highly complex spaces. Modularity can be formed using the chain rule in probability theory and the building blocks are conditional probability distributions. Therefore we introduce a new conditional probabilistic model, that maintains both expressive representation and tractable inference, in this section to compactly represent conditional probability distributions.

3.1. Conditional probabilistic modeling

We are interested in predicting a collection of random variables $\mathbf{Y}$ given inputs $\mathbf{X}$, i.e. we are interested in the problem of *structured output prediction* (SOP). In the probabilistic setting, SOP translates to modeling and learning a high-dimensional conditional distribution $P(\mathbf{Y}\,|\,\mathbf{X})$. While one could learn a univariate predictor for each variable $Y \in \mathbf{Y}$ separately, this approach assumes complete independence among $\mathbf{Y}$. This fully factored mean field assumption is often violated but still frequently used. Indeed, when used for variational inference, mean field assumption encourages representations which reduce dependence between the latent dimensions. However, the expressivity of mean field variational distribution is limited, which is especially a problem when the disentanglement assumption does not hold. Exploring richer variational families would enable a tighter bound of the log partition function, and in turn bound the probability of evidence [17]. Gaussian Processes (GPs) [18] and Conditional Random Fields (CRFs) [19] are more expressive alternatives. However, they have serious shortcomings when inference has to scale to high-dimensional data or many samples. One approach to scale GPs to larger datasets is deep mixtures of GPs, introduced in [20], which can be seen as a combination of GPs and SPNs. However, they are limited to continuous domains. In comparison, CSPNs can learn conditional distributions over heterogeneous data, i.e., where $\mathbf{Y}$ might contain discrete or continuous random variables, or even mixed data types. In a nutshell, CSPNs can tackle *unrestricted* SOP in a principled probabilistic way.

As an alternative within the family of tractable probabilistic models, logistic circuits (LCs) have been recently introduced as discriminative models [21], showing classification accuracy competitive to neural nets on a series of benchmarks. However, LCs and discriminative learning of SPNs [22] are limited to single output prediction. Discriminative arithmetic circuits (DACs) also directly model conditional distributions [23]. They are learned via compilation of CRFs, requiring sophisticated and potentially slow structure learning routines. Also related are sum-product-quotient networks (SPQNs) [24], which extend SPNs by introducing quotient nodes. This enables SPQNs to represent a conditional distribution $P(\mathbf{Y}\,|\,\mathbf{X})$ as the ratio $P(\mathbf{Y},\mathbf{X})/P(\mathbf{X})$ where the two terms are modeled by two SPNs. However, CSPNs can include more complex conditional models allowing for a more compact representation. Furthermore, SPQNs yield intractable conditional distributions, i.e., efficient marginalization and conditioning are not facilitated.

Our neural CSPNs are close in spirit to probabilistic models based on neural networks. Generally, this line of research faces the challenge of how to parameterize distributions using the outputs of deterministic neural function approximators. Frequently, the mean field assumption is made, interpreting the output of the network as the parameters of primitive univariate distributions, assuming independence among random variables. Modeling complex distributions must then involve sampling as in VAEs [2] or hierarchical variational models [25], causing significant computational overhead and yielding highly intractable models. Other approaches for conditional density estimation based on neural networks include (conditional) normalizing flows, which yield tractable likelihoods, but are limited to continuous distributions and come with significant computational costs for computing the determinant of the Jacobian [26]. Generally, CSPNs are most closely related to two classic approaches, namely mixture density networks (MDNs) [27] and (hierarchical) mixtures of experts [28]. These models use the output of neural networks to parameterize a (typically Gaussian) mixture model. Shallow mixture models, however, are often inadequate in high dimensions, as the number of components required to accurately model the data may grow exponentially with the number of dimensions. SPNs address this by encoding a hierarchy of mixtures, alternating between sums and products. Neural CSPNs may consequently be seen as a deep, hierarchical version of MDNs and Mixture of Experts (MoEs).

3.2. Conditional sum-product networks

We now introduce a notion of conditional SPNs (CSPNs), which structurally reflect conditional independencies, and propose a learning framework to induce CSPNs from data.

Definition 2. A CSPN is defined as a rooted DAG containing three types of nodes: leaf, gating, and product nodes. It encodes a conditional probability distribution $P(\mathbf{Y}\,|\,\mathbf{X})$. All leaves of the DAG are normalized univariate conditional distributions, and all internal nodes are either gate or product nodes.

Fig. 2 shows an illustrative example of a CSPN. Each leaf encodes a normalized univariate conditional distribution $P(Y\,|\,\mathbf{X})$ over a target RV $Y \in \mathbf{Y}$, where Y is denoted as the leaf's *conditional scope*. A product node factorizes a conditional probability distribution over its children, i.e., $\prod_k P_k(\mathbf{Y}_k\,|\,\mathbf{X})$ where $\mathbf{Y}_k \subset \mathbf{Y}$. To encode functional dependencies on the input features, a *gating* node computes $\sum_k g_k(\mathbf{X}) P_k(\mathbf{Y}\,|\,\mathbf{X})$ where g_k is the k^{th} output of a non-negative function $g(\mathbf{X}) = (g_1(\mathbf{X}), \ldots, g_K(\mathbf{X}))$ which satisfies $\sum_k g_k(\mathbf{X}) = 1$, where K is the number of children of the gating node. One may also choose a constant gating function, in which case the gating node becomes a standard sum node. The conditional scope of a non-leaf node is the union of the conditional scopes of its children.

Gating nodes are akin to gates in *mixtures of experts* [29,30], which motivates the name. The notions of completeness and decomposability of SPNs [5] naturally carry over to CSPNs, and we can reuse the efficient SPN inference routines, guaranteeing that any conditional marginal probability may be computed exactly in linear time [23]. This can easily be verified by considering that for a fixed input $\mathbf{x}$, a CSPN reduces to a standard SPN over the labels $\mathbf{Y}$, allowing any inference routine for standard SPNs [5,15,31] to be employed. However, CSPNs are more powerful than SPNs, as discussed next.

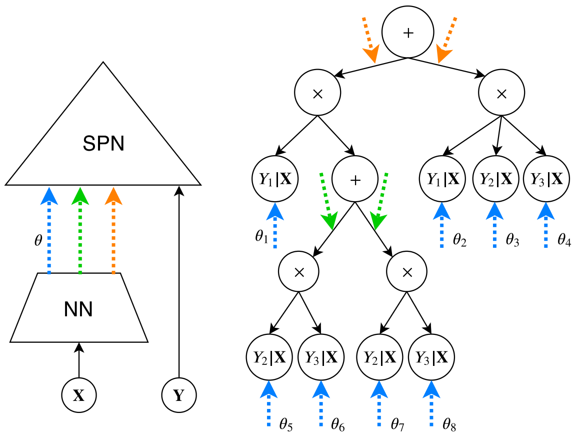


Fig. 2. Overview of the architecture (left) and a concrete CSPN example encoding $P(\mathbf{Y}\,|\,\mathbf{X})$ (right). $\mathbf{X}$ is the set of conditional variables and $\mathbf{Y}$ consists of Y_1, Y_2 and Y_3. Each color of the arrow represents one data flow. The gating weights, possibly also the leaves, are parameterized by the output of a function given $\mathbf{X}$. Specifically, when the gating function is represented by a neural network, we retrieve neural CSPN. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

3.3. *Representational power of CSPNs*

As Choi and Darwiche [32] argue, a probabilistic model[1] over finite-discrete random variables represents not only one function, but a function for each possible probabilistic query — the (conditional) marginals. In other words, a query posed to a Bayesian network model can be viewed as inducing (and evaluating) a function, which can be represented using an Arithmetic Circuit (AC) [4,33]. Consider a Bayesian network, some evidence $\mathbf{e}$ for variables $\mathbf{E}$, and a query variable Y. The probability $P(y, \mathbf{e})$ can be viewed as the output of a function $f(\mathbf{E})$ that maps evidence $\mathbf{e}$ into a number in [0,1]. Unfortunately, the functions corresponding to queries, unlike neural networks, are generally not universal approximators. They are restricted to multi-linear functions of primitive distributions or quotients thereof. Take (unconditional) SPNs encoding $P(\mathbf{Y}, \mathbf{X})$ as an example, under mild conditions, they are universal density approximators since mixture of Gaussians without weights conditioned on the input are already so, see e.g. [34]. As Choi and Darwiche [32] have shown, its joint density is a polynomial, actually a multi-linear function. In turn, any conditional query $P(\mathbf{Y}\,|\,\mathbf{X}) = P(\mathbf{Y}, \mathbf{X})/P(\mathbf{X})$ to the SPN is a quotient of multi-linear functions. By adding gating nodes, however, CSPNs extend SPNs in a way that allows them to induce universal approximators. This can be seen as follows. A Testing Arithmetic Circuit (TAC) is an Arithmetic Circuit (AC) that includes testing units, whose output is determined dynamically based on the given evidence. Its output is computed as

$$f(x, T) = \begin{cases} \theta^+, & \text{if } x \geq T \\ \theta^-, & \text{otherwise.} \end{cases} \tag{1}$$

Using threshold functions $x_i \leq c$ $(c \in \mathbb{R})$ as gates, one can encode *testing arithmetic circuits* [32] as CSPNs. A gate decides which expert component should be activated for that mixture according to some evidence. Here we use neural networks as the threshold functions, but other threshold functions may also be used. The expressivity of CSPNs comes from the power of the gating nodes. Specifically, neural networks, or other threshold functions, at the gating nodes serves as a testing unit for switching on and off its children component. Testing arithmetic circuits, however, have been proven to encode piece-wise multi-linear functions and in turn ResNets, which are universal approximators. Moreover, the class of Gaussian CSPNs mean functions is dense in the class of all continuous functions over arbitrary compact domains. To see this, we note that one can use single-output softmax gates in order to facilitate modeling mixtures of experts models $\sum_k g_k(\mathbf{X}) P_k(\mathbf{Y}\,|\,\mathbf{X})$ in the form of Nguyen et al. [35]. Consequently, their result carries over.

3.4. *Learning conditional sum-product networks*

A simple way to construct CSPNs representing $P(\mathbf{Y}\,|\,\mathbf{X})$ is to start from a standard or even random SPN over $\mathbf{Y}$, and to make its parameters (i.e., the sum weights and parameters of leaf distributions) functional of input $\mathbf{x}$, defining $P(\mathbf{Y}\,|\,\mathbf{X} = \mathbf{x}) := P(\mathbf{Y}; \theta)$, where parameters $\theta := g(\mathbf{x})$ are a function of the input $\mathbf{x}$. For g we might use arbitrary function representations, such as deep neural networks. This architecture, which we call *neural* CSPN, works very well as we demonstrate in the experimental section, but might fail to exploit some conditional independencies among $\mathbf{Y}$. Consequently, we now introduce

[1] In their exposition, they focus on Bayesian networks and arithmetic circuits, but their arguments carry over to SPNs.

Algorithm 1 LearnCSPN ($\mathcal{D}$, η, α).

```
Require: data $\mathcal{D} = \{(\mathbf{y}_i, \mathbf{x}_i) | \mathbf{y}_i \in \mathbf{Y}, \mathbf{x}_i \in \mathbf{X}\}_{i=1}^N$ where $\mathbf{y}_i = \left(y_i^1, \cdots, y_i^{c_y}\right)$, and $\mathbf{x}_i = \left(x_i^1, \cdots, x_i^{c_x}\right)$; $\eta$: min number of instances to split; $\alpha$: threshold of significance
Ensure: a CSPN $S$ encoding $P(\mathbf{Y} | \mathbf{X})$ learned from $\mathcal{D}$
 1: if $|\mathbf{Y}| = 1$ then
 2:     $S \leftarrow$ LearnConditionalLeaf($\mathcal{D}$) using, e.g., GLMs
 3: else if $|\mathcal{D}| < \eta$ then
 4:     $S \leftarrow \prod_{j=1}^{|\mathbf{Y}|}$ LearnCSPN($\{(y_i^j, \mathbf{x}_i)\}_{i=1}^N, \eta, \alpha$)
 5: else
 6:     $\{\mathbf{V}_k\}_{k=1}^K \leftarrow$ SplitLabels($\mathcal{D}, \alpha$) // compare Algorithm 2
 7:     if $K > 1$ then
 8:         $\mathcal{D}_k \leftarrow \{\mathbf{v}_k^m | \mathbf{v}_k^m \sim \mathbf{V}_k\}_{m=1}^M$
 9:         $S \leftarrow \prod_{k=1}^K$ LearnCSPN($\mathcal{D}_k, \eta, \alpha$)
10:     else
11:         $\{\mathcal{D}_k\}_{k=1}^K \leftarrow$ SplitInstances($\mathcal{D}$) e.g. using random splits or k-Means with an appropriate metric
12:         $S \leftarrow \sum_{k=1}^K g_k(x) \cdot$ LearnCSPN($\mathcal{D}_k, \eta, \alpha$), $x \in D$ where $g_k(x)$ is a gating function
13: return $S$
```

Algorithm 2 SplitLabels ($\mathcal{D}$, α).

```
Require: data $\mathcal{D} = \{(\mathbf{y}_i, \mathbf{x}_i) | \mathbf{y}_i \in \mathbf{Y}, \mathbf{x}_i \in \mathbf{X}\}_{i=1}^N$ where the label RVs are $\mathbf{Y} = \{Y_1, \ldots, Y_P\}$, $\alpha$: threshold of significance
Ensure: a label partitioning $\{\mathcal{P}_{\mathcal{D}}\}$
1: $\mathcal{G} \leftarrow$ Graph({})
2: for each $Y_i, Y_j \in \mathbf{Y}$ do
3:     $S_{i,j} \leftarrow M \| \hat{\sum}_{Y_i Y_j \cdot \mathbf{X}} \|_F^2$
4:     if LindsayPillaBasak($S_{i,j}$) $> \alpha$ then
5:         $\mathcal{G} \leftarrow \mathcal{G} \cup \{(i, j)\}$
6: return ConnectedComponents($\mathcal{G}$)
```

a structure learning strategy LearnCSPN, see Algorithm 1, extending the established LearnSPN algorithm [36] which has been instantiated several times for learning (unconditional) SPNs under different distributional assumptions [31,37].

Our LearnCSPN routine builds a CSPN in a top-down fashion by introducing nodes while partitioning a data matrix in a recursive and greedy manner. It creates one of the three node types at each step — (1) a leaf, (2) a product, or (3) a gating node. When applied to a single target RV Y, LearnCSPN fits a conditional probability distribution as a leaf. To generate product nodes, LearnCSPN aims to find conditionally independent subsets of target RVs $\mathbf{Y}$ by means of a statistical test. If no such partitioning is found, then training samples are partitioned into clusters (conditioning) to induce a gating node. Finally, to calibrate all parameters, capturing cross-covariances between $\mathbf{Y}$ and $\mathbf{X}$, we apply end-to-end parameter learning. Let us now review the three steps of LearnCSPN more in detail.

3.4.1. Inducing leaves

In order to allow for tractable inference, we require conditional models at the leaves to be normalized. Apart from this requirement, *any* such univariate tractable conditional model may be plugged in a CSPN effortlessly to model $P(Y | \mathbf{X})$. This can be a simple (parametric) univariate model or the joint output of a deep model. For the sake of simplicity, we here use Generalized Linear Models (GLMs) [38] as leaves, due to their simple and flexible nature. We compute $P(Y | \mu)$ with $\mu := \text{glm}(\mathbf{X})$ by regressing parameters μ from features $\mathbf{X}$, for a given exponential family.

3.4.2. Inducing product nodes

For product nodes, we are interested in decomposing the labels $\mathbf{Y}$ into conditionally-independent subsets via conditional independence (CI) tests. In terms of density functions, testing that Y_i is independent of Y_j given $\mathbf{X} = \mathbf{x}$, for any value of $\mathbf{x}$, i.e., $Y_i \perp\!\!\!\perp Y_j | \mathbf{X}$, can equivalently be characterized as $P(Y_i, Y_j | \mathbf{X}) = P(Y_i | \mathbf{X}) P(Y_j | \mathbf{X})$. To accommodate arbitrary conditional distributions at the leaves, regardless of their parametric likelihood models, we adopt a non-parametric pairwise CI test procedure to decompose labels $\mathbf{Y}$. Specifically, the randomized conditional correlation test (RCoT) [39] is used. It computes the squared Hilbert-Schmidt norm of the partial cross-covariance operator and uses the Lindsay-Pilla-Basak method [40] to approximate the asymptotic distribution. We then create a graph where the nodes are RVs in $\mathbf{Y}$ and put an edge between two nodes Y_i, Y_j if we cannot reject the null hypothesis that $Y_i, Y_j \perp\!\!\!\perp \mathbf{X}$ for a given threshold α. The conditional scopes of product children are then given by the connected components of this graph, akin to [36].

3.4.3. Inducing gating nodes

A probabilistic interpretation of a gating node can be given in the context of mixture models for conditional probability distributions $\sum_k g_k(\mathbf{X}) P_k(\mathbf{Y} | \mathbf{X})$ [29]. Estimating them is a form of *conditional clustering* [41] to accommodating for known cross-covariates between $\mathbf{Y}$ and $\mathbf{X}$. Here, we approach this by inducing the gating node structure using clustering of the features $\mathbf{X}$ only and calibrating the cross-covariates afterwards by end-to-end parameters estimation. The clustering scheme can be instantiated based on the available knowledge of the data distribution (e.g., k-Means for Gaussians); one can also leverage random splits, as in random projection trees [42]. Then, we select a functional form for the gating function $g_k(\mathbf{X})$, ideally, a differentiable parametric one such as logistic regression or a (deep) neural network with a softmax layer such that constraints of a proper mixture of distributions, i.e., $\sum_k g_k(\mathbf{X}) = 1$ and $\forall_{\mathbf{X}} g_k(\mathbf{X}) >= 0$ are fulfilled. We denote the corresponding cluster assignment as a one-hot coded vector $\mathbf{Z}$ and fit the gating function to predict $\mathbf{Z}_k = g_k(X)$.

3.4.4. *Calibrating cross-covariances*

Given the induced structure of the complete CSPN, we calibrate the cross-covariances between $\mathbf{X}$ and $\mathbf{Y}$ by final joint parameter estimation in an end-to-end fashion. That is, we fit each gating function to predict $\mathbf{Z}_k := \arg\max g_k(\mathbf{X})$ (using e.g. softmax) and the parameters of the leaf distributions jointly. This is a proper generalization of traditional sum nodes in SPNs: a sum node is a special case of a gating node where the mixtures are constants independent of $\mathbf{X}$. Moreover, this functional mixing does not break the tractability guarantees over $\mathbf{Y}$ as $\mathbf{X}$ is always assumed to be evidence in the conditional case.

To summarize, LearnCSPN automatically sets the parameters of the gating function as described above. The parameters of the GLMs are obtained by an Iteratively Reweighted Least Squares (IRWLS) algorithm as described in [43], on the instances available at the *leaf* node. However, those parameters are locally optimized and usually not optimal for global distribution. Fortunately, CSPNs are differentiable as long as the leaf models and gating functions are differentiable. Hence, one can optimize the conditional likelihood in an end-to-end fashion using gradient-based optimization techniques.

3.5. *Learning with random structure: neural CSPNs*

In high-dimensional domains, such as images, the structure learning procedure introduced above may be too expensive. In this case, CSPNs may still be applied by starting from a random SPN structure [44], resulting in a flexible distribution $P(\mathbf{Y}; \theta := g(\mathbf{X}))$. When g is represented by a deep neural network, we obtain a highly expressive conditional density estimator which can be trained end-to-end, see again Fig. 2. We call this architecture *neural CSPN*. It resembles a deep version of Bishop's classic mixture density networks [27]. Now, a conditional SPN can be obtained by simply conditioning *all* of the SPN parameters θ on $\mathbf{X}$, i.e. $\theta := g(\mathbf{X})$, yielding a conditional distribution $P(\mathbf{Y} | \mathbf{X})$.

Let $L(Y; X)$ denote the likelihood of Y given X, which is also the loss we use for end-to-end optimization. Let ω be the parameters of the neural networks $g(\cdot)$ whose output is θ. Then the gradient of the loss w.r.t the parameters ω is given by

$$\frac{dL(Y; X)}{d\omega} = \frac{dL(Y; X)}{dg(X)} \frac{dg(X)}{dX}$$

according to the chain rule. $\frac{dL(Y; X)}{dg(X)}$ is the gradient through the SPN, and $\frac{dg(X)}{dX}$ is the gradient through the neural network.

This flexible way of learning CSPNs is not only useful when the output dimensions are too high to learn structure efficiently, it also makes CSPNs fit for building modular structures. In addition, the resulting modular structures may be learnt as a full end-to-end system.

4. Modular probabilistic modeling via CSPNs

In this section we illustrate how to use CSPNs as modular blocks for probabilistic modeling. In particular, we use CSPNs to build generative models with flexible high-level structure based on the chain rule in probability theory. In addition to generative models alone, we also employ neural CSPNs as modules in other system. Specifically, we use neural CSPNs for amortized posterior inference together with generative models to yield a more expressive auto-encoding structure that allows for full end-to-end training.

4.1. *Imposing structure on deep generative models: autoregressive block-wise CSPN*

To illustrate how to impose structure on generative models by employing CSPNs as building blocks, we construct an autoregressive model composed of CSPNs. This is basically done by representing a joint distribution as a factorization of conditional models, as in Bayesian networks. Specifically, by applying the chain rule of probabilities, we can decompose a joint distribution as the product $P(\mathbf{Y}, \mathbf{X}) = P(\mathbf{Y} | \mathbf{X}) P(\mathbf{X})$. Then, one could learn an SPN to model $P(\mathbf{X})$ and a CSPN for $P(\mathbf{Y} | \mathbf{X})$. By combining both models using a single product node, we represent the whole joint as a computational graph. Now, if one applies the same operation several times by repeatedly partitioning $\mathbf{Y}$ in a series of disjoint sets $\mathbf{Y}_1, \mathbf{Y}_2, \ldots$ we can obtain an *autoregressive* model representation, which we call *Autoregressive Block-wise CSPN* (ABCSPN). Inspired by image autoregressive models like PixelCNN [45] and PixelRNN [46], we use ABCSPN for conditional image generation to illustrate the benefits of imposing structure.

For one ABCSPN, we divide images into pixel blocks, hence factorizing the joint distribution block-wise instead of pixel-wise as in PixelC/RNN, see Fig. 1 (left). Each factor accounting for a block of pixels is then a CSPN representing the

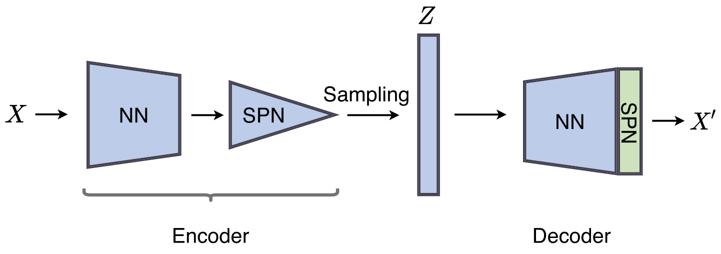


Fig. 3. Overview of SPVAE (with naive decoder), the VAE structure using neural CSPNs. The SPN is parameterized by a neural network, whose distribution is sampled during training time to transform the observation X to the latent code Z for a Monte Carlo estimation of the ELBO. Decoder is a standard neural network that transforms the latent code Z to the parameters for a conditional distribution of the observation.

distribution of those pixels as conditioned on all previous blocks and on the class labels.[2] We factorize blocks in raster scan order, row by row and left to right, but any other ordering is also possible. The complete generative model over image **I** encodes

$$p(\mathbf{I}) = \prod_{i=1}^{n} p(\mathbf{B}_i \mid \mathbf{B}_1, \ldots, \mathbf{B}_{i-1}, \mathbf{C}) \cdot p(\mathbf{C})$$

where $\mathbf{B}_i$ denotes the pixel RVs of the i-th block and $\mathbf{C}$ the one-hot coded image class. Learning each conditional block as a CSPN can be done by the structure learning routine, see subsection 3.4.

4.2. Neural CSPNs for variational inference

Consider variational auto-encoders (VAEs) [11], which combine a generative neural model and a variational inference network in an auto-encoding structure. The generative model of VAEs postulates latent factors $\mathbf{Z}$ with known prior distribution $p(\mathbf{Z})$ (often assumed to be isotropic Gaussian), and a *decoder network* parametrized by ψ which outputs (the parameters of) the data generating distribution $p_\psi(\mathbf{X}|\mathbf{Z})$. The overall generative VAE model, including latent variables, is given as $p_\psi(\mathbf{Z},\mathbf{X}) = p_\psi(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})$, and learning VAEs involves the intractable posterior $p_\psi(\mathbf{Z}|\mathbf{X}) \propto p_\psi(\mathbf{Z},\mathbf{X})$.

The *encoder network*, parameterized with ω, approximates the intractable posterior $p_\psi(\mathbf{Z}|\mathbf{X})$ with a variational distribution $q_\omega(\mathbf{Z}|\mathbf{X})$. In particular, the marginal log-likelihood $\log p_\psi(\mathbf{X})$ is usually intractable, which makes maximum likelihood estimation (MLE) of ψ not directly feasible. Therefore, the lower bound of the marginal log-likelihood [47], a.k.a. evidence lower bound (ELBO),

$$\log p_\psi(\mathbf{X}) \geq -\mathbb{D}_{\mathrm{KL}}[q_\omega(\mathbf{Z}|\mathbf{X})||p(\mathbf{Z})] + \mathbb{E}_q[\log p_\psi(\mathbf{X}|\mathbf{Z})],$$

is maximized instead, where optimization happens jointly over both networks, i.e., ψ and ω. This simultaneously fits the generative model and performs amortized posterior inference. Taking the perspective of auto-encoders, the KL term in the ELBO can be interpreted as a regularizer, and the second term can be interpreted as negative reconstruction error [11].

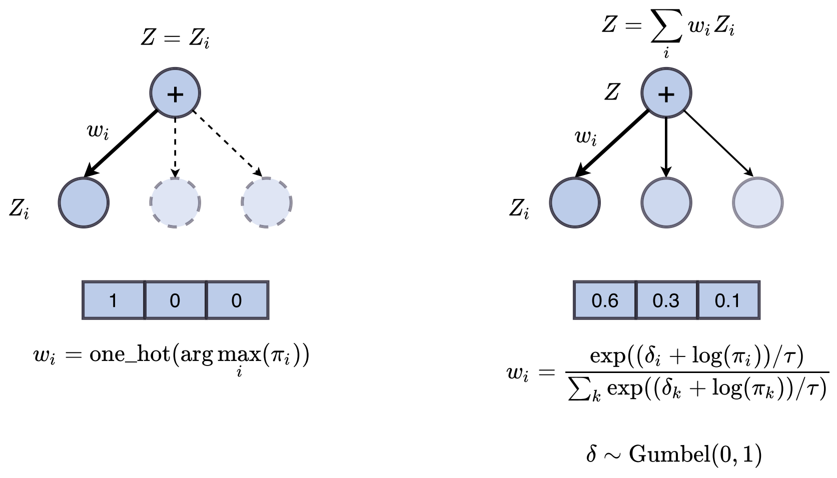
As a simplified choice, the latent factors are often assumed to be independent of each other, and the encoder network hence also makes the mean field assumption [11,48]. But this assumption can be easily violated in practical use. Consider a collection of human face images with a variation of facial expression, orientation, symmetry etc. By definition, symmetry is dependent on both facial expression and orientation. In this case, the simple assumption on the approximate posterior distribution limits the capacity of the inference network on approximating the true posterior distribution.

Therefore, we propose to use CSPNs for variational inference on the latent variables to approximate the posterior distribution, which yields what we call *SPVAE (Sum-Product Variational Auto-Encoders)*. See Fig. 3 for an illustration of SPVAE. CSPNs are a natural fit for this purpose since posterior distributions are essentially conditional distributions. In addition, CSPNs (universal approximators) come with rich representational power, flexible learning routine, and efficient inference routine. More importantly, they capture conditional dependencies between output variables in tree structures, breaking the mean field assumption. Here, we do not only consider isotropic multivariate Gaussian $p(\mathbf{Z}) = \mathcal{N}(\mathbf{Z}; \mathbf{0}, \mathbf{I})$ as the prior distribution of the latent variables [11], but also more complex prior distributions where mean field assumption is not appropriate any more. We focus on continuous latent variables.

In order to allow for optimizing the ELBO with standard stochastic gradient methods, i.e. in an end-to-end fashion, one needs a Monte Carlo estimation of the expectation $\mathbb{E}_q[\log p_\psi(\mathbf{X}|\mathbf{Z})]$ that is differentiable w.r.t. ω. However, backpropagating through stochasticity is not straightforwardly feasible. In particular, the source of stochasticity in standard sampling process in CSPNs stems from leaf nodes (sampling of a univariate Gaussian distribution) and sum nodes (sampling of a categorical distribution, and hard selection of one category).

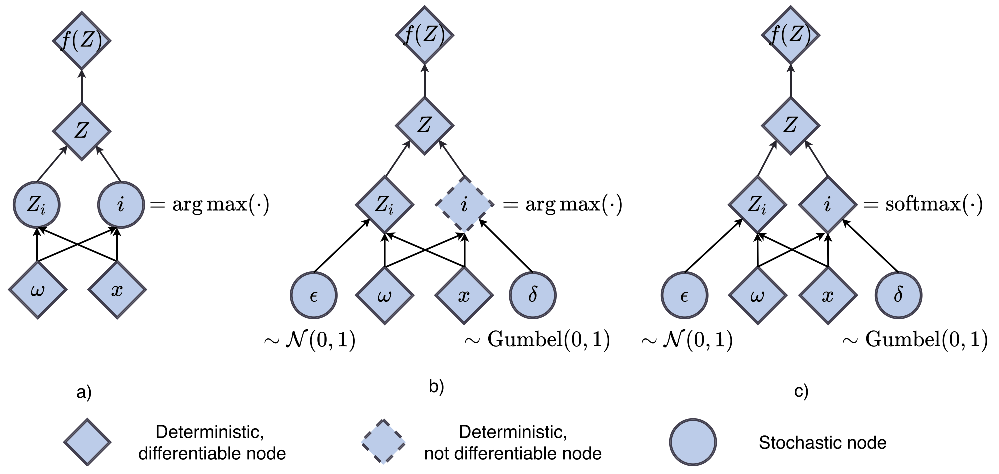
The reparameterization trick, which reformulates the sampling as a deterministic base distribution and independent noise with a fixed distribution, is commonly used for backpropagating through stochasticity [11,49,50]. Specifically, to induce

[2] Here, image labels play the role of observed RVs in **X**, conditioning the generative model on some semantic context.



$Z = Z_i$
w_i
Z_i
1 0 0
$w_i = \text{one\_hot}(\arg\max_i(\pi_i))$
$Z = \sum_i w_i Z_i$
Z
w_i
Z_i
0.6 0.3 0.1
$w_i = \frac{\exp((\delta_i + \log(\pi_i))/\tau)}{\sum_k \exp((\delta_k + \log(\pi_k))/\tau)}$
$\delta \sim \text{Gumbel}(0,1)$

Fig. 4. Sampling at sum nodes in original form selects the component with the highest sampled weight and propagates the corresponding samples upwards (left). To make sampling differentiable at sum nodes, Gumbel-Softmax interpolates the samples of all the children components whose weight is sampled via softmax with reparameterization trick (right). π are parameters of the categorical distribution.



$f(Z)$
Z
Z_i
i
$= \arg\max(\cdot)$
ω
x
a)
ϵ
δ
$\sim \mathcal{N}(0,1)$
$\sim \text{Gumbel}(0,1)$
b)
$= \text{softmax}(\cdot)$
c)
Deterministic, differentiable node
Deterministic, not differentiable node
Stochastic node

Fig. 5. Differentiable sum nodes explained in computation graphs. Arrow indicates data flow. Z are the latent variables, Z_i are leaves of the SPN. ω is the parameters of the encoder's neural networks, x is observation. a) the original form. b) reparameterizing a sum node and a leaf node. c) making sum node differentiable by replacing $\arg\max(\cdot)$ as $\text{softmax}(\cdot)$.

differentiable sampling in leaf nodes, we reformulate a univariate Gaussian distribution $\mathcal{N}(Z|\mu, \sigma^2)$ as $Z = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$. Note that μ and σ are the outcome of a CSPN conditioned on the input $\mathbf{X}$. To make sampling differentiable in sum nodes, we use the reparameterization trick for the categorical distribution with Gumbel-Softmax proposed by Jang et al. [50]:

$$w_j = \frac{\exp((\delta_j + \log(\pi_j))/\tau)}{\sum_k \exp((\delta_k + \log(\pi_k))/\tau)}, \quad \text{for } j = 1, \ldots, k,$$

where π are parameters of the categorical distribution, δ are independent noises with a Gumbel distribution, i.e. $\delta \sim \text{Gumbel}(0, 1)$, and τ is the softmax temperature. The reparameterization trick refactors the softmax distribution with a deterministic function and δ. See Fig. 4 for illustration. In addition, we replace the hard selection of one category by interpolating between all components based on the sampled weights w_j, which is, in turn, resulted from a relaxed version of the categorical distribution. The sampling routine makes one time bottom-up pass of the network and propagate the samples from the leaves up to the root. See Algorithm 3 for the routine and Fig. 5 for illustration in computation graphs. For the KL term $\mathbb{D}_{\text{KL}}[q_\omega(\mathbf{Z}|\mathbf{X})||p(\mathbf{Z})]$ we use Monte Carlo approximation as well.

Learning highly non-Gaussian posterior densities in variational inference has also been explored by Rezende and Mohamed [51]. They do so by learning transformations of simple densities to more complex ones through a normalizing flow. However, normalizing flow comes with significant computational costs for computing the determinant of the Jacobian. Apart from normalizing flows, probabilistic circuits have also been used in variational inference. Shih and Ermon [17]

Algorithm 3 Sample (n_i, τ, x) // differentiable sampling.

```
Require: node $n_i$, temperature $\tau$, observations $x$
Ensure: samples $z$
 1: if $n_i$ is leaf node then
 2:     $\mu, \sigma \leftarrow g_\omega(x)$
 3:     $\epsilon \leftarrow$ i.i.d samples from $\mathcal{N}(0, 1)$
 4:     return $z_i = \mu + \sigma\epsilon$
 5: else if $n_i$ is sum node then
 6:     $\pi \leftarrow g_\omega(x)$
 7:     for all child $n_{ij}$ of $n_i$ do
 8:         $z_{ij}$ = Sample($n_{ij}$, $\tau$, $x$)
 9:         $\delta \leftarrow$ i.i.d samples from Gumbel(0, 1)
10:         $w_{ij} = \frac{\exp((\delta_j + \log(\pi_j))/\tau)}{\sum_k \exp((\delta_k + \log(\pi_k))/\tau)}$
11:     return $\sum_j z_{ij} w_{ij}$
12: else
13:     for all child $n_{ij}$ of $n_i$ do
14:         $z_{ij}$ = Sample($n_{ij}$, $\tau$, $x$)
15:     $z_i \leftarrow$ concatenate $z_{i0}, \ldots, z_{ij}$
16:     return $z_i$
```

used probabilistic circuits to compute ELBO gradients exactly in discrete graphical models. Lowd and Domingos [52] introduced approximate compilation methods to construct effective ACs to induce tractable approximate inference for previously intractable distribution, as in variational inference methods.

5. Experiments

Here we investigate CSPNs in experiments on real-world data. Specifically, we aim to answer the following questions: **(Q1)** Can CSPNs perform better than regular SPNs? **(Q2)** How accurate are CSPNs for SOP? **(Q3)** Do neural CSPNs outperform baseline neural conditional models such as MDNs on image SOP tasks? **(Q4)** How do ABCSPNs perform on real data? **(Q5)** Are CSPNs useful for variational inference in auto-encoding system? To this end, we implemented CSPNs in Python calling TensorFlow and R.

5.1. Multivariate traffic data prediction

We employ CSPNs for multivariate traffic data prediction, comparing them against SPNs with Poisson leaf distributions [53]. This is an appropriate model as the traffic data represents counts of vehicles. We considered temporal vehicular traffic flows in the German city of Cologne [54]. The data comprises 39 RVs whose values are from stationary detectors located at the 50 km long Cologne orbital freeway, each one counting the number of vehicles within a fixed time interval. It contains 1440 samples, each of which is a snapshot of the traffic flow. The task of this experiment is to predict the next snapshot ($|\mathbf{Y}| = 39$) given a historical one ($|\mathbf{X}| = 39$).

We trained CSPNs for $P(\mathbf{Y}|\mathbf{X})$ and SPNs for $P(\mathbf{Y}, \mathbf{X})$ with various depth. CSPNs are trained merely with structure learning (see Algorithm 1). Threshold α for each CSPN are respectively set as 0.1, 0.001 and 0.001, for each SPN 0.1, 0.001 and 0.00001. Minimum number of instances η for each CSPN are set according to the total number of training instance n_{train} by respectively $n_{\text{train}} + 1$, $\lceil n_{\text{train}} * 2/3 \rceil$, $\lceil n_{\text{train}}/2) \rceil$. η for each SPN are respectively $n_{\text{train}} + 1$, $\lceil n_{\text{train}} * 2/3 \rceil$, $\lceil n_{\text{train}}/7) \rceil$. These hyperparameters were chosen to induce same depth for CSPNs and SPNs. The gating nodes for CSPNs are parameterized by static weights in this experiment because standard sum nodes work well enough on this simple problem. The CSPNs use GLMs with exponential link function as Poisson univariate conditional leaves.

Predictions are made by running MPE inference. MPE configurations are computed via a popular approximate algorithm proposed for SPNs in the literature [5]: Replace sum nodes by max nodes. In the upward pass, a max node outputs the maximum weighted value among its children instead of their weighted sum. The downward pass then starts from the root and recursively selects the highest-valued child of a max node, and all children of a product node. For CSPNs, once **X** are observed, the parameters are fixed. And we can apply the classical MPE approximation.

Results are summarized in Fig. 6. We can see that CSPNs are always the most accurate model as their root mean squared error (RMSE) is always the lowest. As expected, deeper CSPNs have lower predictive error compared to shallow CSPNs. Moreover smaller CSPNs perform equally well or even better than SPNs, empirically confirming that CSPNs are more expressive than SPNs. This answers **(Q1, Q2)** affirmatively and also provides evidence for the convenience of directly modeling a conditional distribution.

5.2. Conditional density estimation

We now focus on conditional density estimation on 20 standard binary benchmark datasets. The number of variables of these datasets range from 16 to 1556. In order to estimate conditional density, features are split into evidences ($|\mathbf{X}|$)

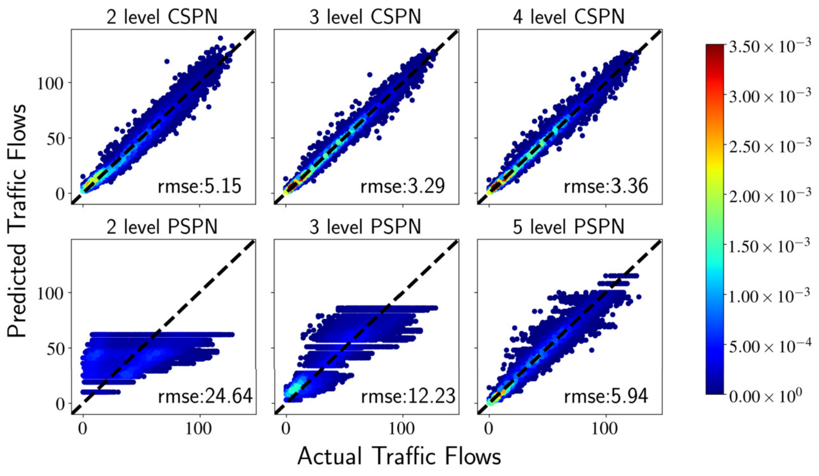


Fig. 6. CPSN can encode nontrivial conditional distributions: Traffic flow predictions of Poisson CSPNs (top) versus SPNs (bottom) for shallow (left) or deep models (center and right). CSPNs are consistently more accurate than corresponding SPNs and, as expected, deeper CSPNs outperform shallow ones (center and right).

Table 1
Average test conditional log-likelihood (CLL) of DACL, CCNs and CSPNs on 20 standard density estimation benchmarks. Best results are bold. As the numbers of wins for DACL, CCNs and CSPNs show, CSPNs are competitive to state-of-the-art.

		Nltcs	Msnbc	KDD	Plants	Audio	Jester	Netflix	Accidents	Retail	Pumsb.
50% evidence	DACL	-2.770	-2.918	**-0.998**	-4.655	-18.958	-24.830	-26.245	**-9.718**	**-4.825**	**-6.363**
	CCN	**-2.58**	**-2.18**	-1.19	**-4.53**	-18.67	-24.96	-26.03	-10.24	-4.88	-6.98
	CSPN	-2.795	-3.165	-1.023	-4.720	**-18.543**	**-24.543**	**-25.914**	-11.587	-5.600	-7.383
80% evidence	DACL	-1.255	-1.557	-0.386	-1.812	-7.337	-9.998	-10.482	**-3.493**	-1.687	-2.594
	CCN	**-0.99**	**-0.87**	**-0.36**	**-1.54**	-7.58	**-9.75**	**-10.22**	-3.61	-1.66	**-2.02**
	CSPN	-1.256	-1.684	-0.397	-1.683	**-7.110**	-9.830	-10.351	-4.045	**-1.654**	-2.618
		Dna	Kosarek	MSWeb	Book	EachMovie	WebKB	Reuters-52	20News	Bbc	Ad
50% evidence	DACL	-34.737	-5.053	-5.653	-16.801	-23.325	-72.072	-41.544	-76.063	-118.684	-4.893
	CCN	**-32.98**	**-4.76**	**-4.25**	-15.90	-24.85	-69.34	-36.56	-71.69	-114.52	**-3.41**
	CSPN	-38.243	-5.527	-6.686	**-10.653**	**-18.130**	**-18.542**	**-15.736**	**-35.900**	**-47.138**	-6.290
80% evidence	DACL	-12.116	-2.549	**-1.333**	-6.817	-9.403	-28.087	-17.143	-27.918	-44.811	-1.370
	CCN	-12.29	**-1.14**	-1.69	-6.48	-8.40	-25.76	-15.67	-27.72	-43.27	-1.18
	CSPN	**-11.895**	-2.397	-1.335	**-3.191**	**-4.579**	**-2.623**	**-3.878**	**-4.984**	**-2.996**	**-1.030**
50% evidence	**Wins**	DACL: 4	CCN: 7	CSPN: **9**							
80% evidence	**Wins**	DACL: 2	CCN: 8	CSPN: **10**							

and targets ($|\mathbf{Y}|$) with different proportions.[3] We compare to DACL [23] and CCNs [10] as they currently provide state-of-the-art conditional log-likelihoods (CLLs) on such data. To this end, we first perform structure learning on the train data split (stopping learning when no more than 10% of samples are available), followed by end-to-end learning on the train and validation data. Each leaf node is an univariate Bernoulli distribution whose parameter is induced by a GLM. The hyperparameter α and η are automatically chosen by a grid search within $\alpha \in \{0.1, 0.01, 0.001\}$ and η from 50 to $n_{\text{train}} + 1$ with spacing between values being 20. The parameters of the CSPNs induced by structure learning are then further optimized end-to-end, which is done with Tensorflow [55] and Adam [56] with its default parameters. Note that the sophisticated structure learning in DACL directly optimizes for the CLL at each iteration.

Results are reported in Table 1 (best in bold). Since we do not have access to per-instance CLLs on CCNs, t-test is not possible, but the results still provide a tendency for comparison. We can see that for both the 80%-evidence scenario and the 50%-evidence scenario, CSPNs win the most. In total, CSPNs perform on par with CCNs (wins are quite balanced) and outperform DACL. Besides, we note that CSPNs are faster to learn than DACL and that, in practice, no real hyperparameter tuning was necessary to achieve these scores, while DACL ones are the result of a fine grained grid search (see [23]). This answers **(Q2)** affirmatively and shows that CSPNs are comparable to state-of-the-art.

Table 2
Comparison to mean field models and mixture density networks: Average test conditional log-likelihood (CLL) and test accuracy of the mean field (MF) model, mixture density network (MDN), and neural conditional SPN (CSPN) on multilabel image classification tasks. The best results are marked in bold. As one can see, the additional representational power of CSPNs yields notable improvements.

	CLL			Accuracy		
	MF	MDN	CSPN	MF	MDN	CSPN
MNIST	-0.70	-0.61	**-0.54**	74.1%	76.4%	**78.4%**
Fashion	-0.95	-0.73	**-0.70**	73.4%	73.7%	**75.5%**
CelebA	-12.1	-11.6	**-10.8**	86.6%	85.3%	**87.8%**

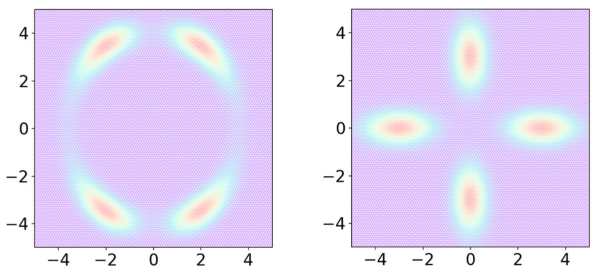


Fig. 7. Two two-dimensional prior distributions used in our experiments.

5.3. Neural CSPNs with random structures

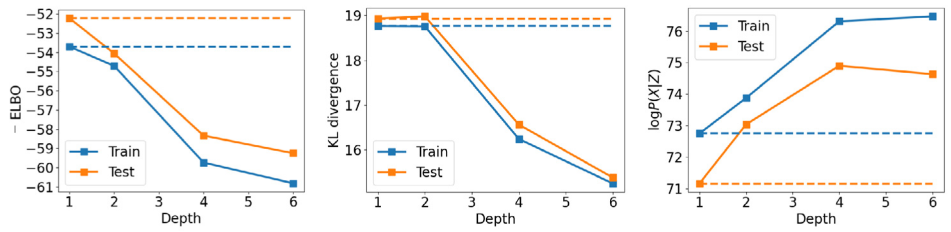
To demonstrate the efficacy of this approach, we evaluate it on several multilabel image classification tasks. The goal of each task is to predict the joint conditional distribution of binary labels Y given an image X. We experiment on the CelebA dataset, which features images of faces annotated with 40 binary attributes. In addition, we constructed multilabel versions of the MNIST and Fashion-MNIST datasets, by adding additional labels indicating symmetry, size, etc. to the existing class labels, yielding 16 binary labels total.

We compare our model to two different common ways of parameterizing conditional distributions using neural networks. The first is the mean field approximation, whereby the output of a neural network is interpreted as logits of independent univariate Bernoulli distributions, assuming that the labels Y are conditionally independent given X. Second, we compare to mixture density networks with 10 mixture components, each itself a mean field distribution. For each of these models, including the CSPN, we use the same standard convolutional neural network architecture up to the last two layers. Those final two layers are customized to the different desired output formats: For the mean field and MDN models, all parameters are predicted using two dense layers. For the CSPN, we use a dense layer followed by a 1d-convolution, in order to obtain the increased number of SPN parameters without using drastically more neural network weights. In this experiment, the structure of CSPNs are randomly initialized rather than learnt by structure learning. Therefore hyperparameters η and α are not considered. In particular, RAT-SPNs [57] are constructed based on a random region graph, which is an abstract representation of the network structure. To hyperparameters used to construct the random region graph is set as following: the number of sum nodes is 8 and the number of input distributions is 4. The resulting conditional log-likelihoods as well as accuracies are given in Table 2. To compute accuracies, we obtained MPE estimates from the models using the standard max-product approximation. On the MNIST and Fashion dataset, estimates were counted as accurate only if all 16 labels were correct, on the CelebA dataset, we report the average accuracy across all 40 labels. The results indicate that the mean field approximation is inappropriate on the considered datasets, as allowing the inclusion of conditional dependencies resulted in a pronounced increase in both likelihood and accuracy. The improved model capacity of the CSPN compared to the MDN yielded a further performance increase. On CelebA, CSPN outperforms a number of sophisticated neural network architectures, despite being based on a standard convnet with only about 400k parameters [58]. This provides an affirmative answer to **(Q3)**.

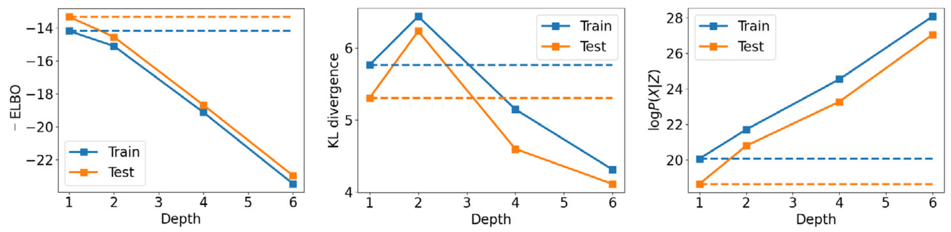
5.4. Auto-regressive image generation

We investigate ABCSPNs on Olivetti faces by splitting each image into 64 blocks of equal size. Then we trained a CSPN on Gaussian domain for each block conditioned on all the blocks above and to the left of it and on the image class and formulate the distribution of the images as the product of all the CSPNs. In Fig. 1 (right), new faces are sampled from an ABCSPN after conditioning on a set of class images that is the mixing of two original classes in the Olivetti dataset. That

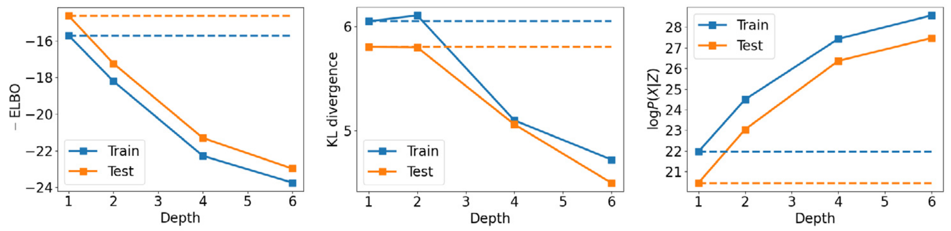
[3] We adopted the data splits of Rooshenas and Lowd [23].



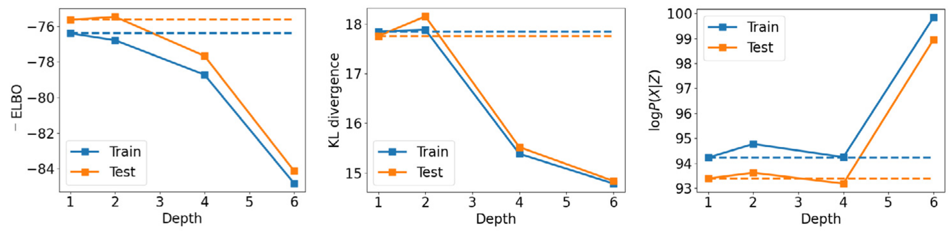
(a) Dataset: MNIST, the prior: isotropic multivariate Gaussian.



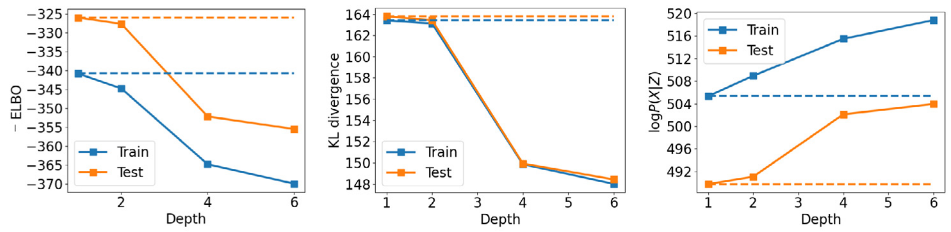
(b) On MNIST. Prior: a two-dimensional non-Gaussian distribution, see Fig. 7 (left).



(c) On MNIST. Prior: a two-dimensional non-Gaussian distribution, see Fig. 7 (right).



(d) On Fashion. Prior: isotropic multivariate Gaussian.

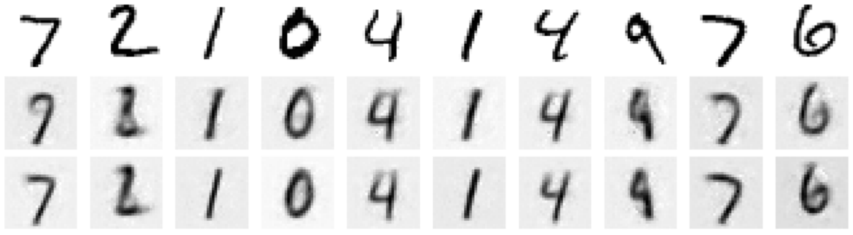


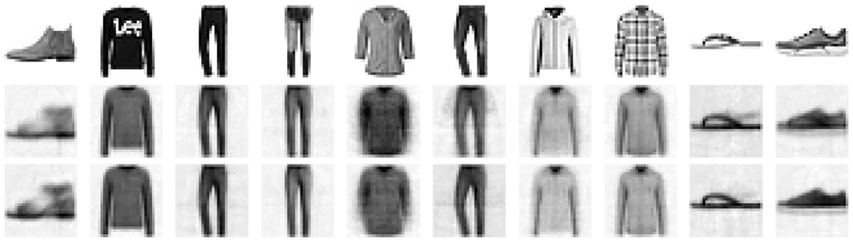
(e) On CelebA. Prior: isotropic multivariate Gaussian.

Fig. 8. The change of ELBO, KL divergence and likelihood when the encoder SPN increases its depth. Dashed lines highlight the value of a vanilla VAE.

is, by conditioning on two classes c_1, c_2 it generates samples that resemble both individuals belonging to those classes by the distribution $P(\cdot|\text{one-hot}(c_1) + \text{one-hot}(c_2))$, even though the ABCSPN never saw that class combination before during training. As one can see, these samples from ABCSPNs look very plausible.

ABCSPNs achieve this while reducing the number of independence tests among pixels required by CSPNs: from quadratic over all pixels in an image down to quadratic in the block size. This shows ABCSPNs are able to learn meaningful and accurate models over the image manifold, providing an affirmative answer to **(Q4)**.


(a) On MNIST.


(b) On Fashion.

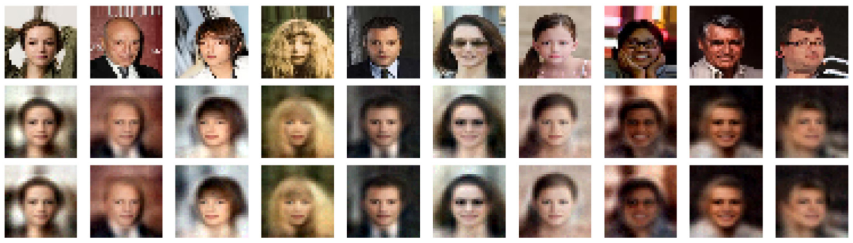

(c) On CelebA.

Fig. 9. The original test images (top row) and reconstruction images using vanilla VAEs (second row), and CSPNs (bottom row).

5.5. *CSPNs for variational inference*

To investigate effectiveness of SPVAE, we use CSPNs of different depth as encoders while keeping their input neural network fixed and compare their ELBO, KL divergence and likelihood with each other (ELBO = likelihood - KL divergence). We use standard convolutional neural network to extract parameters for SPNs, and for each dataset we fix this network up to the last two layers (the last two layers depend on the number of required parameters). We fix the decoder as a naive CSPN, i.e. CSPN with only one layer. Note that a VAE with naive CSPNs as both encoder and decoder makes the mean field assumption on the posterior distribution and is equivalent to the VAE proposed by Kingma and Welling [11]. We experiment on MNIST, Fashion and the CelebA dataset. The VAE on MNIST with isotropic multivariate Gaussian as prior distribution has 10 latent variables, and 2 variables for the two-dimensional priors. The VAE on the CelebA dataset has 128 latent variables and only isotropic multivariate Gaussian priors are used. In this experiment, all CSPNs are initialized with random structure, and the parameters are optimized end-to-end by maximizing the likelihood. The hyperparameters for constructing the random SPN structure are as follows: For all the datasets, number of sum nodes is 3 and number of input distributions is 1 for 2-layered, 4-layered and 6-layered SPNs. For 1-layered SPN, number of sum nodes and number of input distributions are both 1.

Fig. 8a presents a summary of ELBO, KL divergence and likelihood for SPVAE using SPNs with 1, 2, 4 and 6 layers. The prior distribution is 10-dimensional isotropic multivariate Gaussian. The values were obtained on MNIST and on both training set and the held-out set. Fig. 8b and 8c present the same summary but on highly non-Gaussian two-dimensional prior distributions, see Fig. 7 for the prior distributions. When the latent variables are only two dimensional, the SPN in SPVAE can have at most 2 layers, so in this case we increase the capacity of the two-layered SPN with more mixtures. Both figures show that neural CSPNs help to improve variational inference on the latent variables by providing tighter lower bound, specifically both KL divergence and likelihood improved. The same conclusion can also be confirmed on Fashion and the CelebA dataset, see Fig. 8d and 8e.

Fig. 9a presents some randomly chosen held-out images of MNIST (top row), corresponding reconstructed images using vanilla VAE (second row) and SPVAE (bottom row). One can see that the output images are sharper and resemble more of the input images using SPVAE. The same conclusion can also be confirmed on Fashion and the CelebA dataset, see Fig. 9b and 9c.

This series of experiments demonstrate the effectiveness of SPVAEs and provide an affirmative answer to **(Q5)**.

To summarize, the above experiments can answer the research questions from **(Q1)** to **(Q5)** affirmatively. In particular, the representational power of CSPNs, the effectiveness of our learning routine, and the benefit of CSPNs in imposing structures in generative models are confirmed.

6. Conclusions

We extended the concept of sum-product networks (SPNs) towards conditional distributions by introducing conditional SPNs (CSPNs). Conceptually, they combine simpler models in a hierarchical fashion in order to create a deep representation that can model multivariate and mixed conditional distributions while maintaining tractability. They can be used to impose structure on deep probabilistic models and, in turn, significantly boost their power as demonstrated by our experimental results. They can also be used as variational inference networks to provide efficient approximate posterior inference, especially when the mean field assumption is invalid.

Much remains to be explored, including other learning methods for CSPNs, design principles for CSPN+SPN architectures, combining the (C)SPN stack with the deep neural learning stack in general, more work on extensions to sequential and autoregressive domains, and further applications. In particular, one should explore using CSPNs not only on the latent variables, but also on the parameters of the generative model, to yield a full variational Bayesian approach.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors thanks Arseny Skryagin for valuable discussion on differentiable sample for probabilistic circuits. The authors acknowledge the support of the German Science Foundation (DFG) project "Argumentative Machine Learning" (CAML, KE 1686/3-1) of the SPP 1999 "Robust Argumentation Machines" (RATIO). KK also acknowledges the support of the Rhine-Main Universities Network for "Deep Continuous-Discrete Machine Learning" (DeCoDeML), and TL the support of the DFG under project B4 'Analysis and Communication for Dynamic Traffic Prognosis' of the Collaborative Research Centre SFB 876. This project has also received funding from the European Union's Horizon 2020 research and innovation programme under the Marie SkłodowskaCurie Grant Agreement No. 797223.

References

[1] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[2] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, in: Proceedings of International Conference on Learning Representations, 2014.
[3] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, A. Courville, Y. Bengio, Generative adversarial nets, in: Proceedings of Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[4] A. Darwiche, A differential approach to inference in Bayesian networks, J. ACM (2003).
[5] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: Proceedings of Uncertainty in Artificial Intelligence, 2011, pp. 689–690.
[6] M. Johnson, D.K. Duvenaud, A. Wiltschko, R.P. Adams, S.R. Datta, Composing graphical models with neural networks for structured representations and fast inference, in: Proceedings of Advances in Neural Information Processing Systems, 2016, pp. 2946–2954.
[7] P.L. Tan, R. Peharz, Hierarchical decompositional mixtures of variational autoencoders, in: Proceedings of International Conference on Machine Learning, in: Proc. Mach. Learn. Res., vol. 97, 2019, pp. 6115–6124.
[8] Y. Shen, A. Goyanka, A. Darwiche, A. Choi, Structured bayesian networks: from inference to learning with routes, in: Proceedings of AAAI Conference on Artificial Intelligence, vol. 33 (01), 2019, pp. 7957–7965.
[9] Y. Shen, A. Choi, A. Darwiche, Conditional PSDDs: modeling and learning with modular knowledge, in: Proceedings of AAAI Conference on Artificial Intelligence, 2018, pp. 6433–6442.
[10] T. Rahman, S. Jin, V. Gogate, Cutset bayesian networks: a new representation for learning Rao-Blackwellised graphical models, in: Proceedings of International Joint Conference on Artificial Intelligence, 2019, pp. 5751–5757.
[11] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint, arXiv:1312.6114, 2013.
[12] X. Shao, A. Molina, A. Vergari, K. Stelzner, R. Perharz, T. Liebig, K. Kersting, Conditional sum-product networks: imposing structure on deep probabilistic architectures, in: Proceedings of the International Conference on Probabilistic Graphical Models, 2020, pp. 401–412.
[13] R. Peharz, R. Gens, F. Pernkopf, P. Domingos, On the latent variable interpretation in sum-product networks, IEEE TPAMI 39 (10) (2017).
[14] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proceedings of Uncertainty in Artificial Intelligence, 1996, pp. 115–123.
[15] R. Peharz, S. Tschiatschek, F. Pernkopf, P. Domingos, On theoretical properties of sum-product networks, in: Proceedings of International Conference on Artificial Intelligence and Statistics, 2015, pp. 744–752.
[16] Y. Shen, A. Choi, A. Darwiche, Conditional PSDDs: modeling and learning with modular knowledge, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018, pp. 6433–6442.
[17] A. Shih, S. Ermon, Probabilistic circuits for variational inference in discrete graphical models, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 4635–4646.
[18] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning, Adaptive Computation and Machine Learning, MIT Press, 2006.
[19] J.D. Lafferty, A. McCallum, F.C.N. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: Proceedings of International Conference on Machine Learning, 2001, pp. 282–289.
[20] M. Trapp, R. Peharz, C. Rasmussen, F. Pernkopf, Learning deep mixtures of gaussian process experts using sum-product networks, in: Proceedings of International Conference on Machine Learning, 2018.

[21] Y. Liang, G. Van den Broeck, Learning logistic circuits, in: Proceedings of AAAI Conference on Artificial Intelligence, 2019, pp. 4277–4286.
[22] R. Gens, P. Domingos, Discriminative learning of sum-product networks, in: Proceedings of Advances in Neural Information Processing Systems, 2012, pp. 3248–3256.
[23] A. Rooshenas, D. Lowd, Discriminative structure learning of arithmetic circuits, in: Proceedings of International Conference on Artificial Intelligence and Statistics, 2016, pp. 1506–1514.
[24] O. Sharir, A. Shashua, Sum-product-quotient networks, in: Proceedings of International Conference on Artificial Intelligence and Statistics, 2017.
[25] R. Ranganath, D. Tran, D. Blei, Hierarchical variational models, in: Proceedings of International Conference on Machine Learning, 2016, pp. 324–333.
[26] D.J. Rezende, S. Mohamed, Variational inference with normalizing flows, in: Proceedings of International Conference on Machine Learning, 2015, pp. 1530–1538.
[27] C.M. Bishop, Mixture density networks, Tech. Rep. NCRG/4288, Aston University, Birmingham, UK, 1994.
[28] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the em algorithm, Neural Comput. 6 (2) (1994) 181–214.
[29] C.M. Bishop, et al., Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[30] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, J. Dean, Outrageously large neural networks: the sparsely-gated mixture-of-experts layer, in: Proceedings of International Conference on Learning Representations, 2017.
[31] A. Vergari, N. Di Mauro, F. Esposito, Simplifying, regularizing and strengthening spn structure learning, in: Proceedings of ECML/PKDD, 2015, pp. 343–358.
[32] A. Choi, A. Darwiche, On the relative expressiveness of bayesian and neural networks, in: Proceedings of the International Conference on Probabilistic Graphical Models, in: PMLR, vol. 72, 2018, pp. 157–168.
[33] A. Choi, A. Darwiche, On relaxing determinism in arithmetic circuits, in: Proceedings of International Conference on Machine Learning, 2017, pp. 825–833.
[34] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[35] H.D. Nguyen, L.R. Lloyd-Jones, G.J. McLachlan, A universal approximation theorem for mixture-of-experts models, Neural Comput. 28 (12) (2016) 2585–2593.
[36] R. Gens, P. Domingos, Learning the structure of sum-product networks, in: Proceedings of International Conference on Machine Learning, 2013, pp. 873–880.
[37] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, K. Kersting, Mixed sum-product networks: a deep architecture for hybrid domains, in: Proceedings of AAAI Conference on Artificial Intelligence, 2018.
[38] P. McCullagh, Generalized linear models, Eur. J. Oper. Res. 16 (3) (1984).
[39] E.V. Strobl, K. Zhang, S. Visweswaran, Approximate kernel-based conditional independence tests for fast non-parametric causal discovery, J. Causal Inference 7 (1) (2019).
[40] B. Lindsay, R. Pilla, P. Basak, Moment-based approximations of distributions using mixtures: theory and applications, Ann. Inst. Stat. Math. (2000).
[41] X. He, T. Gumbsch, D. Roqueiro, K. Borgwardt, Kernel conditional clustering, in: 2017 IEEE International Conference on Data Mining, ICDM, IEEE, 2017, pp. 157–166.
[42] S. Dasgupta, Y. Freund, Random projection trees and low dimensional manifolds, in: Proceedings of Symp. Theory of Computing, 2008, pp. 537–546.
[43] P.J. Green, Iteratively Reweighted Least Squares for Maximum Likelihood Estimation, and Some Robust and Resistant Alternatives, 1984, JSTOR.
[44] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, Z. Ghahramani, Random sum-product networks: a simple and effective approach to probabilistic deep learning, in: Proceedings of Uncertainty in Artificial Intelligence, 2019, pp. 334–344.
[45] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, et al., Conditional image generation with pixelcnn decoders, in: Proceedings of Advances in Neural Information Processing Systems, 2016, pp. 4790–4798.
[46] A.V. Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, in: M.F. Balcan, K.Q. Weinberger (Eds.), Proceedings of the 33rd International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 48, PMLR, New York, New York, USA, 2016, pp. 1747–1756, https://proceedings.mlr.press/v48/oord16.html.
[47] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, L.K. Saul, An introduction to variational methods for graphical models, Mach. Learn. 37 (2) (1999) 183–233.
[48] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: E.P. Xing, T. Jebara (Eds.), Proceedings of the 31st International Conference on Machine Learning, Bejing, China, in: Proc. Mach. Learn. Res., vol. 32, 2014, pp. 1278–1286, https://proceedings.mlr.press/v32/rezende14.html.
[49] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: Proceedings of International Conference on Machine Learning, 2014, pp. 1278–1286.
[50] E. Jang, S. Gu, B. Poole, Categorical reparametrization with Gumbel-softmax, in: Proceedings International Conference on Learning Representations, 2017, OpenReviews.net, https://openreview.net/pdf?id=rkE3y85ee.
[51] D. Rezende, S. Mohamed, Variational inference with normalizing flows, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, Lille, France, in: Proceedings of Machine Learning Research, vol. 37, 2015, pp. 1530–1538, https://proceedings.mlr.press/v37/rezende15.html.
[52] D. Lowd, P. Domingos, Approximate inference by compilation to arithmetic circuits, in: Proceedings of Advances in Neural Information Processing Systems, vol. 23, 2010, pp. 1477–1485.
[53] A. Molina, S. Natarajan, K. Kersting, Poisson sum-product networks: a deep architecture for tractable multivariate Poisson distributions, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (1), Feb. 2017, https://ojs.aaai.org/index.php/AAAI/article/view/10844.
[54] C. Ide, F. Hadiji, L. Habel, A. Molina, T. Zaksek, M. Schreckenberg, K. Kersting, C. Wietfeld, Lte connectivity and vehicular traffic prediction based on machine learning approaches, in: Proceedings of Vehicular Technology Conference, IEEE, 2015, pp. 1–5.
[55] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, software available from tensorflow.org, https://www.tensorflow.org/, 2015.
[56] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations Conference Track Proceedings, 2015, http://arxiv.org/abs/1412.6980.
[57] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, Z. Ghahramani, Random sum-product networks: a simple and effective approach to probabilistic deep learning, in: R.P. Adams, V. Gogate (Eds.), Proceedings of the 35th Uncertainty in Artificial Intelligence Conference, in: Proc. Mach. Learn. Res., vol. 115, 2020, pp. 334–344, https://proceedings.mlr.press/v115/peharz20a.html.
[58] M. Ehrlich, T.J. Shields, T. Almaev, M.R. Amer, Facial attributes classification using multi-task representation learning, in: Proceedings of the CVPR Workshops, 2016, pp. 47–55.