Sum-Product-Transform Networks: Exploiting Symmetries using Invertible Transformations

Tomáš Pevný, Václav Šmídl
Department of Computers
Czech Technical University
Prague, Czech republic

Martin Trapp
Graz University of Technology
Graz, Austria

Ondřej Poláček, Tomáš Oberhuber
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University
Prague, Czech republic

Abstract

In this work, we propose Sum-Product-Transform Networks (SPTN), an extension of sum-product networks that uses invertible transformations as additional internal nodes. The type and placement of transformations determine properties of the resulting SPTN with many interesting special cases. Importantly, SPTN with Gaussian leaves and affine transformations pose the same inference task tractable that can be computed efficiently in SPNs. We propose to store and optimize affine transformations in their SVD decompositions using an efficient parametrization of unitary matrices by a set of Givens rotations. Last but not least, we demonstrate that G-SPTNs pushes the state-of-the-art on the density estimation task on used datasets.

1. INTRODUCTION

Modeling and manipulating complex joint probability distributions are central goals in machine learning. Its importance derives from the fact that probabilistic models can be understood as multi-purpose tools, allowing them to solve many machine learning tasks using probabilistic inference. However, recent flexible and expressive techniques for density estimation, such as normalizing flows Rezende and Mohamed (2015); Kobyzev et al. (2019) and neural auto-regressive density estimators Uria et al. (2016), lag behind when it comes to performing inference tasks efficiently. Motivated by the absence of tractable probabilistic inference capabilities, recent work in probabilistic machine learning has put forth many instances of so-called Probabilistic Circuits (PCs), such as Sum-Product Networks (SPNs) Poon and Domingos (2011), Probabilistic Sentential Decision Diagrams (PSDDs) Kisa et al. (2014) and Cutset network Rahman et al. (2014). In contrast to auto-regressive and flow-based techniques, PCs guarantee that many inference tasks can be computed exactly in time linear in their representation size. The critical insights for PCs are that: i) high-dimensional probability distributions can be efficiently represented by composing convex combinations, factorizations, and tractable input distributions; and that ii) decomposability Darwiche (2003) simplifies many inference scenarios to tractable inference at the input distributions. Due to their favorable properties, PCs have

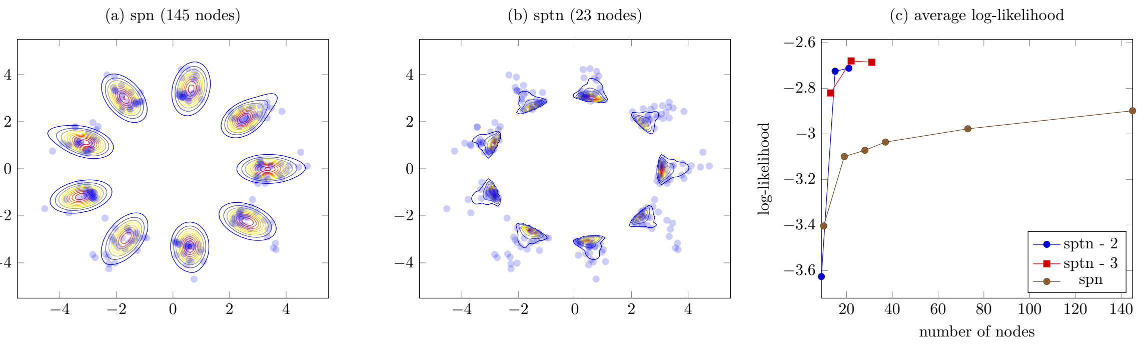


Figure 1: Figures (a-b), respectively, show the density function of an SPN and SPTN overlayed onto a subset of training data. SPTNs can fit the data more effectively, by exploiting transformations of the density function and result in a more compact representation, c.f. Figure (c).

been successfully applied for many complex machine learning tasks, e.g. image segmentation Rathke et al. (2017), semantic mapping Zheng et al. (2018), and image classification Peharz et al. (2019).

To model complex probability distributions, PCs leverage a hierarchy of convex combinations and factorizations, resulting in a compact representation of an exponentially large mixture distribution. However, by restricting to compositions of tractable input distributions using convex combinations and factorizations, PCs cannot exploit geometric properties, such as symmetries, in the density function and lack a compact representation in low-dimensional scenarios. Thus, potentially resulting in efficient representations of complex joint probability distributions in various scenarios.

In this paper, we propose to extend PCs to additionally include invertible transformations. In particular, we introduce Sum-Product-Transformation Networks (SPTNs), which combine SPNs, i.e. complete and decomposable PCs, with an additional change of variables transformations. The resulting model class naturally combines tractable computations in normalizing flows with tractable computations in SPNs. SPTNs are an expressive and flexible probabilistic model that enables exploitation of the geometry, e.g. symmetries, while facilitating tractable inference scenarios, depending on the network structure.

To show the merits, consider the *flower* dataset illustrated in Figure 1, which consists of nine petal leaves located symmetrically around zero. Naturally, one would like this symmetry to be leveraged by model, e.g. using rotations around the origin, but this is currently impossible with SPNs. SPTNs makes this possible. Subfigure (a) shows the density of a fitted SPN model with 145 nodes and Subfigure (b) that of a fitted SPTN model with just 23 nodes (both models full covariance Gaussian leaves). The density function of SPTN model fits the petal leaves better. Subfigure (c) charts average log-likelihood on the testing set with respect to the number of nodes. That thanks to the ability to exploit symmetry, SPTNs achieve higher log-likelihood with almost order of magnitude fewer nodes. The experimental evaluation further confirms this.

Our main contributions can be summarised as follows: (i) we introduce an extension of probabilistic circuits (PCs) which interleaves common compositions in PCs with invertible transformations, resulting in a flexible tractable probabilistic model, which *unifies two paradigms: probabilistic circuits and flow models into a single framework*; (ii) we introduce a new affine flow and conjecture that for many interesting applications an affine transformation is sufficient; (ii) Our affine flow has a native parametrization in SVD decomposition, which allows *efficient inverse* and *efficient calculation* of determinant of the Jacobian and its gradients. (iii) we introduce a tractable subclass, G-SPTNs, consisting of sum and product nodes, Gaussian leaves, and only affine transformations. G-SPTNs support efficient marginalization and computation of conditionals.

2. BACKGROUND

Probabilistic circuits (PCs) are a large class of tractable probabilistic models that admit many probabilistic inference tasks in linear time (linear in their representation size).

Definition 1 (Probabilistic Circuit) *Given a set of random variables (RVs) $\mathbf{X}$, a Probabilistic Circuit (PC) is defined as tuple $(\mathcal{G}, \psi, \theta)$ consisting of a computational graph $\mathcal{G} = (V, E)$, which is a directed acyclic graph (DAG) containing sum, product and leaf nodes, a scope-function $\psi : V \to 2^{\mathbf{X}}$ and a set of parameters θ.*

In SPNs nodes are as follows:

- leaf node $\mathsf{L} \in V$ is a (tractable) distribution over its scope $\psi(\mathsf{N})$ parametrized by θ_{L};
- sum node is a weighted sum with non-negative weights of its children, i.e. $\mathsf{S}(x) = \sum_{\mathsf{N}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}}\mathsf{N}(x)$ with $w_{\mathsf{S},\mathsf{N}} \geq 0$, w.l.o.g. we assume $\sum_{\mathsf{N}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} = 1$;
- product node is a product of its children , i.e. $\mathsf{P}(x) = \prod_{\mathsf{N}\in\mathbf{ch}(\mathsf{P})} \mathsf{N}(x_{\psi(\mathsf{N})})$, where $\mathbf{ch}(\mathsf{N})$ returns the set of children of node N.

In general, we additionally expect the *scope-function* to fulfil the following properties: i) for all internal nodes $\mathsf{N} \in V$ we have $\psi(\mathsf{N}) = \bigcup_{\mathsf{N}'\in\mathbf{ch}(\mathsf{N})} \psi(\mathsf{N}')$ and ii) for each root node N, i.e. each node without parents, we have $\psi(\mathsf{N}) = \mathbf{X}$. To guarantee many inference scenarios to be tractable, we additionally require the scope-function to fulfil that, for each product node $\mathsf{P} \in V$ the scopes of the children of P are disjoint, i.e. $\bigcap_{\mathsf{N}'\in\mathbf{ch}(\mathsf{P})} \psi(\mathsf{N}') = \emptyset$ (*decomposability*). In this paper, we further assume that, for each sum node $\mathsf{S} \in V$ that $\psi(\mathsf{N}) = \psi(\mathsf{N}')\ \forall \mathsf{N}, \mathsf{N}' \in \mathbf{ch}(\mathsf{S})$ (*completeness/smoothness)*. Complete/smooth and decomposable PCs are often referred to as Sum-Product Networks (SPNs).

SPNs have recently gained increasing attention, due to their success in various applications, e.g. Stelzner et al. (2019); Peharz et al. (2019). Inspired by these successes, various flexible extensions of SPNs have recently been proposed, e.g. SPNs over variational autoencoders (VAEs) Tan and Peharz (2019), SPNs over Gaussian processes Trapp et al. (2020) and quotient nodes to represent conditional distributions within the SPN Sharir and Shashua (2018). However, to the best of our knowledge, SPNs and PCs have not been extended to incorporate invertible transformations as of yet.

3. SUM-PRODUCT- TRANSFORMATION NETWORKS

Sum-Product-Transformation Networks (SPTNs) naturally combine SPNs with normalizing flows. by extending them with nodes representing a change of variables formulas.

Definition 2 (Sum-Product-Transformation Network) *A Sum-Product-Transformation Network (SPTN) over a set of RV $\mathbf{X}$ is an extension of PCs which is recursively defined as:*

- *An arbitrary (tractable) input distribution is an SPTN (leaf node), i.e.* $\mathsf{L}(x) = p(x \mid \theta_{\mathsf{L}})$.
- *A product of SPTNs is an SPTN (product node), i.e.* $\mathsf{P}(x) = \prod_{\mathsf{N}\in\mathbf{ch}(\mathsf{P})} \mathsf{N}(x_{\psi(\mathsf{N})})$.
- *A convex combination of SPTNs is an SPTN (sum node), i.e.* $\mathsf{S}(x) = \sum_{\mathsf{N}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}}\mathsf{N}(x)$ *with* $w_{\mathsf{S},\mathsf{N}} \geq 0$.

- *An invertible transformation of an SPTN is an SPTN (transformation node), i.e.* $\mathsf{T}(\mathsf{N}(x)) = \mathsf{N}(g(x)) \det |J_g(x)|$ *where* $g(x)$ *is a bijection and* $J_g(x)$ *denotes the Jacobian of the transformation.*

In the course of this paper, we will generally assume SPTNs to be complete/smooth and decomposable. Note that those properties are akin to completeness and decomposability in SPNs, as transformation nodes (T) have only a single child and, thus, $\psi(\mathsf{T}) = \psi(\mathsf{N}) \forall \mathsf{N} \in \mathbf{ch}(\mathsf{T})$.

3.1 REALIZATION OF TRANSFORMATION NODES

To calculate the density of a transformed random variable $z = f(x)$ constraints $f(x)$ to be invertible (bijection) and for practical reasons the determinant of the Jacobian of $f(x)$ has to be efficiently calculated. Recently introduced normalizing flows Rezende and Mohamed (2015) (see Papamakarios et al. (2019) for an overview) achieve these by imposing a special structure on f or by relying on properties of ODE equations. Although these approaches can be used in the proposed SPTN, the tractability of marginalization would be lost. We, therefore, extend this family by introducing a variant of dense layers in feed-forward networks, which allows efficient inversion, computation of the Jacobian, and in a special case does not destroy the tractability.

Recall that feed-forward neural networks implement a function $f(x) = \phi(\mathbf{W}x + b)$, where $\mathbf{W}$ is a weight matrix, b is a bias term, and $\phi(x)$ is a (non-)linear transformation. We further require $\mathbf{W} \in \mathbb{R}^{d,d}$ to have full rank, as it has to be invertible, and $b \in \mathbb{R}^d$. Furthermore, singular value decomposition (SVD) tells that $\mathbf{W}$ can be expressed as $\mathbf{W} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{D}$ is a diagonal matrix. SVD decomposition allows for a convenient calculation of the inverse of f as $f^{-1}(z) = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^\top(\phi^{-1}(z) - b)$, and also simplifies calculation of Jacobian as $\log\left(\left|\frac{\partial f}{\partial x}\right|\right) = \sum_{i=1}^d \log|d_{ii}| + \sum_{i=1}^d \log\left|\frac{\partial \phi_i}{\partial o_i}\right|$, where $o = \mathbf{U}\mathbf{D}\mathbf{V}^\top x + b$. Unfortunately, the SVD decomposition is generally expensive to calculate to be used directly.

We propose to keep $\mathbf{W}$ in SVD decomposition and optimize directly in it, which eliminates the expensive decomposition. This is possible by parametrizing group of unitary matrices $\mathcal{U}$ by $\theta \in \mathbf{\Theta}$ such that (i) $\mathbf{U}(\theta) \in \mathbf{\Theta}$ is a unitary square matrix for arbitrary $\theta \in \mathbf{\Theta}$ and (ii) for every unitary matrix $\mathbf{U}'$ there exists $\theta'\mathbf{\Theta}$ such that $\mathbf{U}' = \mathbf{U}(\theta')$; and (iii) a gradient $\frac{\partial \mathbf{U}(\theta)}{\partial \theta}$ exists and can be computed efficiently. Discussion of two possible parametrizations are in the next section.

The type of Transformation nodes and their placements in the computational graph has an impact on the tractability of the resulting model. Since they realize just transformation of variables, they *can be shared* within the model, which allows a compact representations (recall the *Flower* dataset in Introduction). We now discuss a few important special cases:

Affine Gaussian SPTN (G-SPTN): SPTNs with Gaussian leaves and arbitrarily placed affine transformations can be transformed into an exponentially large mixture of Gaussians, c.f. Theorem 3. This has an important consequence as marginalization is now analytically tractable, which arises from the fact that affine-transformed Gaussian distributions remain Gaussian.

Flow models: Any SPTN consisting only of transformation and product nodes is a flow Papamakarios et al. (2019). Note, however, that marginalization and computation of moments are generally not tractable.

SPN with Flexible Leaves: An SPTN with transformations only just above the leaf nodes extends the set of possible leaf node distributions. Since the transformation is deferred only to leaves, in the univariate case, tractability is generally preserved, as in Tan and Peharz (2019).

SPTN allows exploiting tractability in certain parts of the model while sacrificing it in favor of complex transformations in others, which provides flexibility to adjust models according to needs.

Theorem 3 *Inference tasks that are tractable in SPNs are also tractable in SPTN with affine transformation nodes and Gaussian distribution at the leaves (called G-SPTN).*

Proof Let the SPTN be composed of sum, product, affine transformation, and Gaussian leaf nodes. Further, let us assume that all $\mu_\cdot$ are vectors and all $\mathbf{\Sigma}_\cdot$ are matrices of appropriate dimensions.

Then, (i) *An affine transformation of a Gaussian distributed vector is Gaussian.* Specifically, let $x \sim \mathcal{N}(\mu_x, \mathbf{\Sigma}_x)$. Then $y \sim \mathcal{N}(\mu_y, \mathbf{\Sigma}_y)$ with $\mu_y = \mathbf{W}\mu_x + b$ and $\mathbf{\Sigma}_y = W\mathbf{\Sigma}_x W^\top$.

(ii) *The product distribution of Gaussian distributed vectors is Gaussian.* Let $x_1 \sim \mathcal{N}(\mu_1, \mathbf{\Sigma}_1)$ and $x_2 \sim \mathcal{N}(\mu_2, \mathbf{\Sigma}_2)$. Then,

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma}_1 & 0 \\ 0 & \mathbf{\Sigma}_2 \end{bmatrix} \right). \tag{1}$$

(iii) *The density function of any SPN can be represented by an exponentially large mixture.* As shown in Zhao et al. (2016); Trapp et al. (2019), any SPN can be represented by an exponentially large mixture distribution over so-called induced trees. The same applies to SPTNs as transformation nodes have only a single child, i.e., if a transformation node is included in the induced tree its child and the respective edge will also be included.

By applying (iii), we can express SPTN as a mixture of induced trees. Since every induced tree has Gaussian distributions on leaves and its inner nodes are only transformation and product nodes, by recursive application of (i) and (ii) it can be equivalently represented by a single Gaussian distributions. Thus G-SPTN can be expressed as a mixture of Gaussians, which are tractable. ■

Corollary 4 *Marginal and conditional distributions of G-SPTN have the same analytical properties as SPNs using Gaussian distributions at the leaves with a block-diagonal covariance structure.*

NODE SHARING SPTNs allow reducing the number of parameters via node sharing. Since the introduced transformation is just a new type of node, it can be shared in the computational graph just like the sum and product nodes. This is illustrated in schematics, such as in Figure 2.

4. PARAMETERIZING UNITARY MATRICES

We now discuss methods to parametrize groups of unitary matrices and then discuss the pros and cons. [1]

1. Both methods are implemented in a publicly available package `https://github.com/pevnak/Unitary.jl`.

4.1 GIVENS PARAMETRIZATION

The first parametrization relies on a set of Givens rotations. Let us assume a Givens rotation $\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$ in $\mathbb{R}^{2\times 2}$, parametrized by $\theta \in \mathbb{R}$. This matrix is unitary for every value of θ and vice-versa for every unitary matrix $\mathbf{U} \in \mathbb{R}^{2,2}$ with positive determinant there exists θ such that $\mathbf{U} = \mathbf{U}(\theta)$. As shown by Polcari (2014), for any $d = 2^k$, $k > 1$ a group of unitary matrices in the space $\mathbb{R}^{d,d}$ is parametrized by a set of Givens transformations. We generalize this to matrices of arbitrary dimension $d > 1$ as follows.

Theorem 5 *Let* $\mathbf{U} \in \mathbf{R}^{d,d}$, $d > 1$ *be a unitary matrix and let* $\mathbf{G}^{r,s}(\theta)$ *denotes an almost diagonal matrix, with a Givens rotation on* r *and* s *columns.*[2] *Then there exist* $\{(\mathbf{G}^{r,s}(\theta_{r,s}))|1 \leq r < s \leq d, \theta_i \in \mathbf{R}\}$, *such that* $\mathbf{U} = \prod_{1<r<s}^{d,d} \mathbf{G}^{r,s}(\theta_{r,s})$.

Proof Proof is in the supplementary at `https://arxiv.org/abs/2005.01297` ■

The corollary of this theorem is that $\prod_{1<r<s}^{d,d} \mathbf{G}^{r,s}(\theta_{r,s})$, parametrizes a whole group of positive definite unitary matrices in $\mathbb{R}^{d,d}$ using $\frac{1}{2}d(d-1)$ parameters. The parametrization is not unique due to periodicity of goniometric functions and $\mathbf{U}(0)$ is equal to identity.

4.2 HOUSEHOLDER PARAMETERIZATION

The second parametrization relies on the representation of unitary matrix $\mathbf{U} \in \mathbb{R}^{d,d}$ as a product of at most d Householder transformations Urías (2010), i.e. $\mathbf{U} = \mathbf{P}_d\mathbf{P}_{d-1}\ldots\mathbf{P}_1$, where each $\mathbf{P}_i$ is defined by vector $\mathbf{y}_i$ as $\mathbf{P}_i = \mathbf{I} - t_i\mathbf{y}_i\mathbf{y}_i^\top$, for $t_i = 2/\left\|\mathbf{y_i}\right\|^2$. By using d reflections P_i we can effectively generate a whole group of unitary matrices. This construction over-parametrizes the group, as it uses d^2 parameters for a group with only $\frac{1}{2}d(d-1)$ degrees of freedom.

4.3 COMPUTATIONAL COMPLEXITY

The computational complexity of *Givens parametrization* is $2d(d-1)$ multiplications and $d(d-1)$ additions while that of *Householders* is $2d^2$ multiplications and the same number of additions. In both cases, the backpropagation is three times more expensive if intermediate results of sub-transformations are not stored but computed on the fly as has been proposed in Gomez et al. (2017).

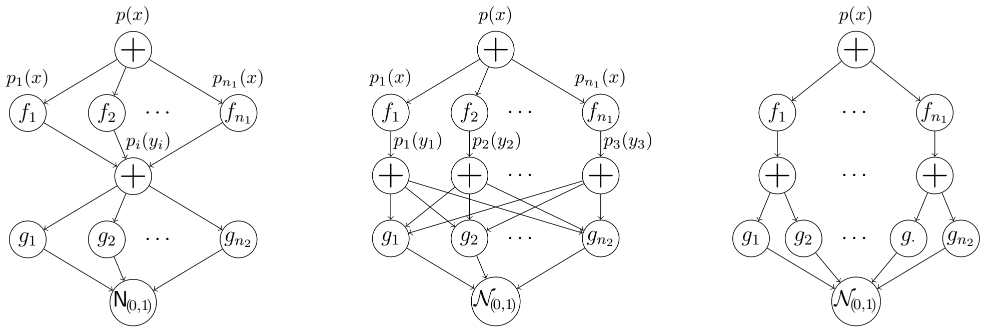
Because Givens parametrization has lower computational complexity and the identity coincides with all parameters being zero, we prefer it, yet both are compared in the experimental section.

5. RELATED WORK

The proposed approach combines mixture models, probabilistic circuits, flow models, and representation of unitary matrices. Although each of these topics has a rich literature, the proposed combination is unique, and only the most relevant works combining the transformation of variables and mixtures/PCs are reviewed below.

Sum-Product Networks The works by Tan and Peharz (2019) and Trapp et al. (2020) can be understood as a flexible extension of SPNs that uses transformation in leave nodes. In particular,

2. For example $\mathbf{G}^{1,3}(\theta)$ in $\mathbb{R}^{4,4}$ has the form $\mathbf{G}^{1,3}(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.



(a) Sharing sum and transformation nodes (b) Sharing only transformation nodes (c) No sharing

Figure 2: Different modes of sharing nodes (parameters) in SPTNs. The model in Subfigure (a) shares sum nodes $\oplus$ and transformation nodes $\{g_i\}_{i=1}^{n_2}$, which means that all $\{f_i\}_{i=1}^{n_1}$ have the same child node $\oplus$ and $\{g_i\}_{i=1}^{n_2}$ thereof; that in Subfigure (b) shares only transformation nodes $\{g_i\}_{i=1}^{n_2}$; and finally that in Subfigure (c) does not share any node except leaf, which does not have any parameters.

Tan and Peharz (2019) proposed to combine SPNs with variational autoencoders (VAE) on leaves, while Trapp et al. (2020) proposed to extend SPNs with Gaussian processes at leaves. However, both approaches do not exploit invertible transformations as internal nodes and are therefore conceptually different from our proposal.

Mixture models The use of mixture models on the latent layer of variational autoencoder Dilokthanakul et al. (2016) may be understood as a transformation node as the root followed by a summing node. Mixtures of flow models have been recently suggested in Papamakarios et al. (2019) using a shallow structure but without any experimental evidence. Further, optimal transport has been used within Gaussian mixture models in Chen et al. (2018).

Unitary matrices Unitary matrices have been proposed for autoencoders Tomczak and Welling (2016), convolution layers Putzky and Welling (2019), and for recurrent neural networks Arjovsky et al. (2016), to the best of our knowledge, they have not been applied to invertible flows and/or combined with SVD.

6. EXPERIMENTS

We have compared G-SPTNs to the prior art, specifically to SPNs, GMMs, and Masked Autoregressive Flows Papamakarios et al. (2017) (MAF), on the task of density estimation, which is the usual benchmark problem for these models Poon and Domingos (2011); Peharz et al. (2019). Experiments were carried out on 21 real-valued problems designed for anomaly detection Pevný (2016). They were originally derived from the UCI database for evaluation of anomaly detectors, such that the complexity of the problem is maximized. All experiments were repeated five times with different random division of data from the "normal" / majority class into training (64%), validation (16%), and testing sets (20%).

6.1 ESTIMATING PARAMETERS

Since SPTNs are a strict superset of SPNs, which are a strict superset of GMMs, we have used the same method to estimate the parameters of all models. In particular, we used stochastic gradient descend to maximize the log-likelihood as done in Peharz et al. (2019). Since parameters of transformation nodes in SPTNs are differentiable, we can apply automatic differentiation to learn all parameters of SPTNs. Specifically, we have used Adam Kingma and Ba (2014) with a batchsize of 100 in all experiments.

6.2 COMPARED MODELS

Since advanced structure learning in SPTN is not yet available, we have used a random sampling of architectures to learn both SPNs and SPTNs, akin to Peharz et al. (2019); Rashwan et al. (2016). The very same references also showed that randomly generated architectures are competitive to those found by structure learning algorithms. The best structure was selected on the validation set. Unless said otherwise, the experimental setting is as described below.

Gaussian Mixture Model The only free architectural parameter is the number of components. We trained mixture models with $n \in \{2, 4, 8, 16, \ldots, 512\}$ components and full covariance Gaussian distributions implemented using an affine transformation before each leaf node with $\mathsf{N}(0, \mathbf{I})$.

Sum-Product network In case of SPNs, we have varied number of children of each sum node, $n \in \{2, 4, 8, 16, \ldots, 128\}$, the number of partitions under each product node, $b \in \{1, 2, 4, 8, 16, 32\}$, and the number of layers, $l \in \{1, 2, 3, 4, 5\}$, where by layer we mean a combination of a sum and product node. Similarly to GMMs, full covariance Gaussian distributions were implemented using an affine transformation before each leaf node with $\mathsf{N}(0, \mathbf{I})$.

Affine Gaussian Sum-Product-Transform network To decrease the degrees of freedom in architecture search for SPTNs, we omitted product nodes in our architecture search. The sampled architectures had $l \in \{1, 2, 3\}$, layers (a layer is the combination of a sum node followed by an affine transformation node) and the number of children under each sum nodes, $n \in \{2, 4, 8, 16\}$. We also distinguished between architectures with {no sharing, sharing of transformation nodes, sharing of sum, and transformation nodes} as outlined in Figure 2. Leaf nodes were fixed to $\mathsf{N}(0, \mathbf{I})$.

Masked auto-regressive flows In the case of MAFs, we performed a similar random search for the architecture as for G-SPTNs. We randomly sampled the number of masked auto-regressive layers Germain et al. (2015) $l \in \{5, 10, 20\}$, the number of layers in these layers $m \in \{1, 2, 3, 4\}$, and the number of neurons in each layer $k \in \{10, 20, 40, 80\}$. The non-linearity was fixed to $\tanh$, as was used in the reference implementation `https://github.com/gpapamak/maf`. Parameters of MAFs have been learned as described above, by maximizing the log-likelihood Papamakarios et al. (2017). Similarly to all the above models, we performed 10,000 optimization steps.

SPTNs, SPNs, and GMMs were implemented using the same library available at `https://github.com/pevnak/SumProductTransform.jl` designed to implement arbitrarily DAGs containing sum, product, transformation nodes, and leaves represented by their density functions. All algorithms were trained for 10,000 iterations. In the case of (G)-SPTN and MAF, we restricted the random search to 100 architectures or 3 days of total CPU time per problem and repetition of the problem.

6.3 EXPERIMENTAL RESULTS

Table 1 shows the average test log-likelihood of G-SPTNs, SPNs, GMMs, and MAFs calculated as $\frac{1}{n}\sum_{i=1}^{n}\log p(x_i)$. Reported values are macro averages over five repetitions of the experiment. On 15 out of 21 datasets, G-SPTNs obtain the highest log-likelihood, and in some cases like miniboone, statlog-segment, and cardiotocography the difference is significant. Contrary, the difference of SPTN to the best model in `Waveforms`, `Pendigits` (less than 0.1) and `Wine` is negligible with the only significant difference being only on `wall-following-robot`. We conjecture this to be caused by the omission of product nodes in our architecture search. The poor performance of MAFs is caused by over-fitting, which can be seen from a high log-likelihood on training data (Table 4 in supplementary).[3]

dataset	G-SPTN	SPN	GMM	MAF
breast-cancer-wisconsin	**-0.07**	-20.55	-6.05	-1874.29
cardiotocography	**45.91**	11.06	10.95	-598.63
magic-telescope	-4.12	-5.78	-4.58	**-3.44**
pendigits	-1.16	-6.51	-2.3	**1.21**
pima-indians	**-7.35**	-8.18	-8.7	-68.81
wall-following-robot	-12.59	**-4.45**	-7.9	-21.08
waveform-1	-23.87	**-23.85**	-23.9	-29.56
waveform-2	-23.91	**-23.85**	-23.89	-25.19
yeast	**8.22**	-0.62	-3.17	0.28
ecoli	**0.66**	-3.21	-3.79	-3.93
ionosphere	**-11.75**	-22.15	-12.69	-3457.46
iris	**-1.79**	-2.28	-1.87	-53.97
miniboone	**162.46**	73.75	43.53	-965573.45
page-blocks	**12.46**	2.58	3.75	5.67
parkinsons	**-3.55**	-19.68	-10.13	-2931.57
sonar	**-74.8**	-74.88	-84.88	-18991.33
statlog-satimage	**4.6**	-9.65	2.52	4.1
statlog-segment	**34.39**	9.63	11.07	-191.06
statlog-vehicle	**-2.76**	-11.73	-5.38	-106.13
synthetic-control-chart	**-39.51**	-43.92	-40.21	-9433.77
wine	-13.61	**-13.39**	-13.92	-3074.69
rank	**1.38**	2.57	2.62	3.43

Table 1: Average log-likelihood of the best models (higher is better) on the test set. Best models were selected according to the performance on the validation set. The best model is in bold blue. The average rank is calculated according to the ranking of each model on each problem (lower is better).

Influence of parametrization of Unitary matrices Although both Givens and Householder parametrizations generate the whole group of Unitary matrices, they might influence learning, for example, due to overparameterization in Householder or more natural representation of *identity* in Givens. We have therefore executed the above experiments with G-SPTN with unitary matrices in affine Transformation nodes realized by both parametrizations. There are certainly differences between them, but across datasets, they perform the same, since Givens was better on 10 problems while Householder on 11. Complete results are in Table 2 in the supplementary.

Influence of node sharing Since SPTN allows flexible sharing of nodes within the network, we have compared no sharing, sharing transformation nodes, and sharing sum and transformation nodes outlined in Figure 2. Networks sharing sum and transformation nodes were generally inferior being best on six problems, whereas two other modes of sharing were best on nine problems. We conjecture that sharing only transformation nodes greatly improves flexibility for a small increase in the number of parameters. Complete results are in Table 3 in the supplementary.

Influence of (non)-linearity Since Transformation nodes in SPTN permit non-linear functions after the affine transformation, we were curious to see if non-linear transformations improve the fit (the average likelihood). We have therefore compared SPTN with linear, leaky-relu Maas et al. (2013), and selu Klambauer et al. (2017) transformations applied element-wise after affine transformation in transformation nodes. According to average log-likelihood on the testing set, G-SPTN with linear functions was the best on 19 out of 21 problems, which implies affine transformations (G-SPTNs) are sufficient for these problems. Complete results are in Table 1 in the supplementary.

3. Full version is available at `https://arxiv.org/abs/2005.01297`.

7. CONCLUSION

In this paper, we suggest extending the compositions used in Probabilistic Circuits to additionally include invertible transformations. Within this new class, called Sum-Product-Transform Networks (SPTN), two frameworks, Probabilistic Circuits, and Flow models unite and each becomes a special case. Since models in SPTNs, in general, do not support efficient marginalization and conditioning, an important sub-class (called G-SPTN) for which these operations are efficient was identified. G-SPTN restricts transformations to be affine and leaf nodes to be Gaussian distributions. The affine transformations keep their projection matrices in SVD forms, which is facilitated by parametrizing groups of unitary matrices, which is treated in detail.

The proposed approach was experimentally compared to Sum-Product Networks (SPNs), Gaussian mixture models, and Masked autoregressive flows on a corpus of 21 publicly available problems. Because SPTNs unify flow models and SPNs, it should not be surprising that the results confirm their good modeling properties. But importantly, this good performance was achieved by G-SPTN, which still features efficient marginalization and conditioning.

Despite good experimental results, there remain several open problems some of which we plan to address in the future. Specifically, a major challenge in learning SPNs is structure learning, which has inspired many sophisticated techniques, e.g. Vergari et al. (2015); Peharz et al. (2019); Trapp et al. (2019). Learning structures for SPTNs is even more challenging and we hope that some of the existing techniques for SPNs can be extended to SPTNs in the future. Moreover, we want to explore more efficient parameter learning for SPTN, as done in the SPN literature, and conduct a more in-depth investigation of the capacities of SPTNs for anomaly detection.

Acknowledgments

This paper was supported by Czech Ministry of education under the GACR project 18-21409S and by OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765 ”Research Center for Informatics”.

References

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *In proceedings of the International Conference on Machine Learning (ICML)*, pages 1120–1128, 2016.

Y. Chen, T. T. Georgiou, and A. Tannenbaum. Optimal transport for gaussian mixture models. *IEEE Access*, 7:6269–6278, 2018.

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

N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. Made: Masked autoencoder for distribution estimation. In *In proceedings of the International Conference on Machine Learning (ICML)*, pages 881–889, 2015.

A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224, 2017.

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

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

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

I. Kobyzev, S. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods, 2019.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *In proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 2338–2347, 2017.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, X. Shao, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, page 124, 2019.

T. Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

J. Polcari. Butterfly decompositions for arbitrary unitary matrices - rev 2. Technical report, 02 2014.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *In proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 337–346, 2011.

P. Putzky and M. Welling. Invert to learn to invert. In *In proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 444–454, 2019.

T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceedings of the Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645, 2014.

A. Rashwan, H. Zhao, and P. Poupart. Online and distributed bayesian moment matching for parameter learning in sum-product networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics AISTATS*, volume 51, pages 1469–1477, 2016.

F. Rathke, M. Desana, and C. Schnörr. Locally adaptive probabilistic models for global segmentation of pathological OCT scans. In *In proceedings of Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 10433, pages 177–184, 2017.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *In proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 1530–1538, 2015.

O. Sharir and A. Shashua. Sum-product-quotient networks. In *In proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84, pages 529–537, 2018.

K. Stelzner, R. Peharz, and K. Kersting. Faster attend-infer-repeat with tractable probabilistic models. In *In Proceedings of the International Conference on Machine Learning (ICML)*, volume 97, pages 5966–5975, 2019.

P. L. Tan and R. Peharz. Hierarchical decompositional mixtures of variational autoencoders. In *In proceedings of the International Conference on Machine Learning (ICML)*, pages 6115–6124, 2019.

J. M. Tomczak and M. Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

M. Trapp, R. Peharz, H. Ge, F. Pernkopf, and Z. Ghahramani. Bayesian learning of sum-product networks. In *In proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 6344–6355, 2019.

M. Trapp, R. Peharz, F. Pernkopf, and C. E. Rasmussen. Deep structured mixtures of gaussian processes. In *In proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

B. Uria, M. Côté, K. Gregor, I. Murray, and H. Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17:205:1–205:37, 2016.

J. Urías. Householder factorizations of unitary matrices. *Journal of Mathematical Physics*, 51: 072204, 2010.

A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

H. Zhao, T. Adel, G. Gordon, and B. Amos. Collapsed variational inference for sum-product networks. In *International Conference on Machine Learning*, pages 1310–1318, 2016.

K. Zheng, A. Pronobis, and R. P. N. Rao. Learning graph-structured sum-product networks for probabilistic semantic maps. In *In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 4547–4555, 2018.