Deep Homogeneous Mixture Models: Representation, Separation, and Approximation

Priyank Jaini
Department of Computer Science & Waterloo AI Institute
University of Waterloo
pjaini@uwaterloo.ca

Pascal Poupart
University of Waterloo, Vector Institute & Waterloo AI Institute
ppoupart@uwaterloo.ca

Yaoliang Yu
Department of Computer Science & Waterloo AI Institute
University of Waterloo
yaoliang.yu@uwaterloo.ca

Abstract

At their core, many unsupervised learning models provide a compact representation of homogeneous density mixtures, but their similarities and differences are not always clearly understood. In this work, we formally establish the relationships among latent tree graphical models (including special cases such as hidden Markov models and tensorial mixture models), hierarchical tensor formats and sum-product networks. Based on this connection, we then give a unified treatment of exponential separation in *exact* representation size between deep mixture architectures and shallow ones. In contrast, for *approximate* representation, we show that the conditional gradient algorithm can approximate any homogeneous mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ "shallow" architectures, where the hidden constant may decrease (exponentially) with respect to the depth. Our experiments on both synthetic and real datasets confirm the benefits of depth in density estimation.

1 Introduction

Multivariate density estimation, a widely studied problem in statistics and machine learning [28], is becoming even more relevant nowadays due to the availability of huge amounts of unlabeled data in various applications. Many unsupervised and semi-supervised learning algorithms either implicitly (e.g. generative adversarial networks) or explicitly estimate (some functional of) the underlying density function. In this work, we study the problem of density estimation with an explicit representation through finite mixture models (FMMs) [19], which have endured thorough scientific scrutiny over decades. The popularity of FMMs is largely due to their simplicity, interpretability, and universality, in the sense that, given sufficiently many components (satisfying mild conditions), FMMs can approximate any distribution to an arbitrary level of accuracy [22].

Many familiar unsupervised models in machine learning, at their core, provide a compact representation of homogeneous density mixtures. This list includes (but is not limited to) hidden Markov models (HMM), the recently proposed tensorial mixture models (TMM) [26], latent tree graphical models (LTM)[21], hierarchical tensor formats (HTF) [13], and sum-product networks (SPN) [9; 24]. However, despite all being a certain form of FMM, the precise relationships among these models are not always well-understood. Our first contribution fills this gap: we prove (roughly) that

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

$\{HMM, TMM\} \subseteq LTM \subseteq HTF \subseteq SPN$. Moreover, converting from a lower to an upper class can be achieved in linear time and without any increase in size. Our results not only clarify the similarities and subtle differences between these widely-used models, but also pave the way for a unified treatment of many properties of such models, using tools from linear algebra.

We next investigate the consequence of converting a deep mixture model into a shallow one. We first prove that the (nonnegative) tensor rank exactly characterizes the minimum size of a shallow SPN (or LTM or HTF due to equivalence) that represents a given homogeneous mixture. Then, we show that a *generic* "deep" SPN (with depth at least 2) can be *exactly* represented by a shallow SPN only when the latter contains exponentially many product nodes. Our result extends significantly those in [7; 26; 10; 18; 8] in various aspects, but most saliently from the restrictive full binary tree [7; 26] to *any* rooted tree. As a consequence, our results imply that a generic HMM (whose underlying tree is "completely" unbalanced) cannot be exactly represented by any polynomially-sized shallow SPN, which, to our best knowledge, has not been shown before.

From a practical point of view, *exact* representations are an overkill: it suffices to *approximate* a given density mixture with reasonable accuracy. Our third contribution demonstrates that under the ℓ_∞ metric, we can approximate any homogeneous density mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ shallow SPNs. However, our proof requires the knowledge of the target density hence is not practical. Instead, borrowing a classic idea from [17] we show that minimizing the KL divergence using the conditional gradient algorithm can also approximate any homogeneous mixture within ϵ accuracy by combining $O(1/\epsilon^2)$ base SPNs, where the hidden constant decreases exponentially wrt the depth of the base SPNs. Each iteration of the conditional gradient algorithm amounts to learning a base SPN hence can be efficiently implemented. We conduct thorough experiments on both synthetic and real datasets and confirm the benefits of depth in density estimation.

We proceed as follows: In §2 we introduce homogeneous density mixtures. In §3 we articulate the relationships among various popular mixture models. §4 examines the exponential separation in *exact* representation size between deep and shallow models while §5 turns into *approximate* representations. We report our experiments in §6 and finally we conclude in §7. All proofs are deferred to Appendix C.

2 Density Estimation using Mixture Models

In this section, we introduce our main problem: how to estimate a multivariate density through an explicit, finite homogeneous mixture. To set up the stage, let $\mathbf{x} = (x_1, \dots, x_d)$, with $x_i \in \mathbb{X}_i$ where each $\mathbb{X}_i$ is a Borel (measurable) subset of the Euclidean space $\mathbb{E}_i$. We equip a Borel measure μ_i on $\mathbb{X}_i$. All our subsequent measure-theoretic definitions are w.r.t. the Borel σ-field of $\mathbb{X}_i$ and the measure μ_i. Let $\mathbb{X} = \mathbb{X}_1 \times \cdots \times \mathbb{X}_d$ and $\mu = \mu_1 \times \cdots \times \mu_d$ be the product space and product measure, respectively. For each $i \in [d] := \{1, \dots, d\}$, let $\mathcal{F}_i$ be a class of density functions (w.r.t. μ_i) of the variable x_i, and let $\mathcal{G}_i = \operatorname{conv}(\mathcal{F}_i)$ be its convex hull. The function class $\mathcal{F}_i$ is essentially our basis of densities for the variable x_i. Our setting here follows that in [18] and includes both continuous and discrete distributions.

We are interested in constructing a finite density mixture [19], using component densities from the basis class $\mathcal{F} = \bigcup_{i=1}^d \mathcal{F}_i$. We assume that our finite mixture f is "**homogeneous**," i.e.

$$f(\mathbf{x}) = \sum_{j_1=1}^{k_1} \sum_{j_2=1}^{k_2} \cdots \sum_{j_d=1}^{k_d} \mathcal{W}_{j_1, j_2, \dots, j_d} \prod_{i=1}^{d} f_{j_i}^i(x_i) = \langle \mathcal{W}, \vec{f}^1(x_1) \otimes \cdots \otimes \vec{f}^d(x_d) \rangle, \tag{1}$$

where $\vec{f}^i := (f_1^i, \dots, f_{k_i}^i) \in \mathcal{F}_i^{k_i}$, $\mathcal{W} \in \bigotimes_i \mathbb{R}_+^{k_i} \simeq \mathbb{R}_+^{k_1 \times \cdots \times k_d}$ is a d-order density tensor (nonnegative and sum to 1), and $\langle \cdot, \cdot \rangle$ is the standard inner product on the tensor product space. We refer to the excellent book [13] and Appendix A for some basic definitions about tensors. By dropping linearly dependent densities in each $\mathcal{F}_i$ we can assume w.l.o.g. the tensor representation $\mathcal{W}$ is *unique*.

There are a number of reasons for restricting to homogeneous mixtures: Firstly, this is the most common choice for estimating a multivariate density function [28]. Secondly, we can always apply the usual "homogenization" trick, i.e., by enlarging the function class $\mathcal{F}_i$ and appending the (improper) density 1 to each $\mathcal{F}_i$. Thirdly, homogeneous densities are "universal" if each class $\mathcal{F}_i$ is, c.f. Appendix A of [26]. In other words, any joint density can be approximated arbitrarily well by a homogeneous density, provided that each marginal class $\mathcal{F}_i$ can approximate any marginal density arbitrarily well and the size (i.e. k_i) tends to ∞. See Appendix F.1 for some empirical verifications, where we show that convex combinations of relatively few isotropic Gaussians can approximate mixtures of

Gaussians of full covariance matrices surprisingly well. Lastly, as we argue below, many known models in machine learning are simply compact representations of homogeneous mixtures.

3 Compact Representation of Homogeneous Mixtures

We now recall a few unsupervised learning models in machine learning and show that they have a compact representation of homogeneous mixtures at their core. We prove the precise relationship amongst them. Our results clarify the similarity and difference of these recent developments, and pave the way for a unified treatment of depth separation (Section 4) and model approximation (Section 5).

Sum-Product Networks (SPN) [9; 24; 18] An SPN $\mathfrak{T}$ is a rooted tree whose *leaves are density functions* $f_j^i(x_i)$ over each of the variables $x_1, \ldots, x_d$ and whose internal nodes are either a sum node or a product node. Each edge (u, v) emanating from a sum node u has an associated nonnegative weight w_{uv}. The value $\mathfrak{T}_v$ at a product node v is the product of the values of its children, $\prod_{u \in ch(v)} \mathfrak{T}_u$. The value $\mathfrak{T}_u$ at a sum node u is the weighted sum of the values of its children, $\sum_{v \in ch(u)} w_{uv} \mathfrak{T}_v$. The value of an SPN $\mathfrak{T}$ is the expression evaluated at the root node, which we denote as $\mathfrak{T}(\mathbf{x})$. The *scope* of a node v in an SPN is the set of all variables that appear in the leaves of the sub-SPN rooted at v. We only consider *decomposable* and *complete* SPNs, i.e., the children of each sum node must have the same scope and the children of each product node must have disjoint scopes. The main advantage of a decomposable and complete SPN over a generic graphical model is that joint, marginal and conditional queries can be answered by two network evaluations and hence, exact inference takes linear time with respect to the size of the network [9; 24; 18]. In comparison, inference in Bayesian Networks and Markov Networks may take exponential time in terms of the size of the network. W.l.o.g. we can rearrange an SPN to have alternating sum and product layers (see Theorem C.1).

The latent variable semantics [23] as well as SPNs representing a mixture model over its leaf densities [24] is well-known. It is also informally known that many tractable graphical models can be treated as SPNs, but precise characterizations are scarce (see [29] which relates SPNs with Bayesian Networks).

Self-similar SPNs (S^{3}PN) We call an SPN self-similar, if at *every* sum node, the sub-tree rooted at each of its (product node) children is the same, except the weights at corresponding sum nodes and the densities (but not the variables) at corresponding leaf nodes may differ. This special class of SPNs is exactly equivalent to some recently proposed unsupervised learning models, as we show below.

Hierarchical Tensor Format (HTF) [13] We showed in (1) that a homogeneous mixture can be identified with a tensor $\mathcal{W}$, whose explicit storage can, however, be quite challenging since its size is $\prod_{i=1}^{d} k_i$. HTF [13] aims at representing tensors compactly, hence can also be used for representing homogeneous mixtures. An HTF consists of a dimension-partition rooted tree (DPT) T, d vector spaces V_i with bases[1] $\mathcal{F}_i$ at the d leaf nodes, and at most $d-1$ internal nodes which are certain *subspaces* of the tensor product of vector spaces at *disjoint* children nodes. Note that the dimension of the tensor product $\mathsf{U} \otimes \mathsf{V}$ is the product of the dimensions of U and V. The key in HTF is to truncate each tensor product with a (much smaller) subspace, hence keeping the total storage manageable. Moreover, at each internal node v with k children nodes $\{v_i\}$, instead of storing its r bases directly, we store r coefficient tensors $\{w^{v,\gamma}_{j_1,\ldots,j_k} : \gamma \in [r]\}$ such that, recursively, the γ-th basis at node v is $\sum_{j_1} \cdots \sum_{j_k} w^{v,\gamma}_{j_1,\ldots,j_k} \mathbf{v}_{j_1} \otimes \cdots \otimes \mathbf{v}_{j_k}$, where $\{\mathbf{v}_{j_i}\}$ consists of the bases at the i-th child node v_i. To our best knowledge, HTFs have not been recognized as SPNs previously, although they have been used in a spectral method for latent variable models [27].

To turn an HTF into an SPN, more precisely an S^{3}PN, we start from the root of the dimension-partition tree T. For each internal node v with say r bases and say k children nodes $\{v_i\}$, each of which has r_i bases themselves, we create three layers in the corresponding S^{3}PN: in the first layer we have r sum nodes $\{\mathsf{S}^v_\gamma\}$, each of which is (fully) connected, with respective weights $w^{v,\gamma}_{j_1,\ldots,j_k}$, to the second layer of $\prod_{i=1}^{k} r_i$ product nodes $\{\mathsf{P}^v_{j_1,\ldots,j_k}\}$, and finally the third layer consists of $\sum_{i=1}^{k} r_i$ sum nodes $\{\mathsf{S}^{v_i}_{j_i}\}$. The product node $\mathsf{P}^v_{j_1,\ldots,j_k}$ is connected to k sum nodes $\{\mathsf{S}^{v_1}_{j_1}, \ldots, \mathsf{S}^{v_k}_{j_k}\}$. Note that the weights $w^{v,\gamma}_{j_1,\ldots,j_k}$ need not be positive or sum to 1 in HTF, although for representing a homogeneous mixture we can make this choice and we call this subclass HTF$_+$. Clearly, our construction is reversible hence we can turn an S^{3}PN into an equivalent HTF$_+$ as well. The construction takes linear time and there is

[1] More generally frames, in particular, the elements need not be linearly independent.

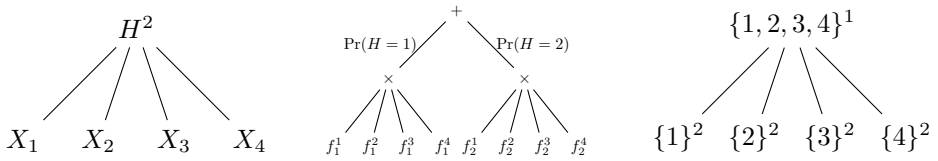


Figure 1: Left: A simple latent class model (special case of LTM). The superscript 2 indicates the number of values the hidden variable H can take. Middle: The equivalent S^{3}PN, where $f_j^i(x_i) = p(X_i = x_i|H = j)$ is from the density class $\mathcal{F}_i$. Right: The dimension-partition tree in an equivalent HTF$_+$. The superscript indicates the number of bases, which should be the same for sibling nodes.

no increase of representation size. See Figs.1,5 for simple illustrations[2]. In summary, HTF *is exactly* S^{3}PN *with arbitrary weights*.

Diagonal HTF (dHTF) [13] For later reference, let us call the subclass of HTFs whose coefficient tensors $w^{v,\gamma}_{j_1,\ldots,j_k}$ (that define bases recursively at internal nodes of the DPT, see above) are diagonal for all v and γ as dHTF, i.e., siblings in the DPT must have the same number of bases ($r_i \equiv r$) and $w^{v,\gamma}_{j_1,\ldots,j_k} \neq 0$ only when $j_1 = \ldots = j_k$. In neural network terminology, dHTFs are "locally connected." Compared to the fully connected HTF, dHTFs significantly reduce the representation size (at the expense of expressiveness, see Figure 7). For instance, the $\prod_{i=1}^k r_i = r^k$ product nodes in the above conversion from HTF to S^{3}PN are reduced to merely r product nodes.

Latent Tree Models (LTM) [21; 27; 5] An LTM is a rooted tree graphical model with observed variables X_i on the leaves and hidden variables H_j on the internal nodes. Note that we allow observed variables X_i to be either continuous or discrete but the hidden variables H_j can take only finitely many values. Using conditional independence, the joint density of observed variables is given as

$$f(x_1, \ldots, x_d) = \sum_{h_1} \cdots \sum_{h_t} \mathcal{W}(h_1, \ldots, h_t) \prod_{i=1}^d f^i_{h_{\pi_i}}(x_i), \tag{2}$$

where H_{π_i} is the parent of X_i. From (2) it is clear that an LTM is a homogeneous density mixture, whose tensor representation is given by the joint density $\mathcal{W}$ of the hidden variables. What is less known[3] is that LTMs are a special subclass of **self-similar** SPNs. It may appear that the size of S^{3}PN is larger than that of an equivalent LTM, but this is because S^{3}PN also encodes the conditional probability tables (CPT) into its structure whereas LTMs require other means to store CPTs. Note also that to evaluate an LTM, one usually needs to run a separately designed algorithm (such as message passing), while in S^{3}PN we evaluate the leaf densities and propagate in linear time to the root. In summary, LTM *is a subclass of* S^{3}PN *with CPTs encoded as edge weights and with inference simplified as network propagation*. More precisely, LTM is exactly dHTF$_+$, since conditioned on the parent, all children nodes must depend on the same realization. An algorithm for converting LTMs into equivalent S^{3}PNs, along with more examples (Figs. 1-6), can be found in Appendix B.1.

Tensorial Mixture Models (TMM) [26; 7; 6] TMM [26] is a recently proposed subclass of dHTF$_+$ where nodes on the same *level* of the dimension-partition tree must have the same number of bases. Clearly, TMM is a strict subclass of LTM since the latter only requires sibling nodes in the DPT to have the same number of bases. We note that TMM, as defined in [26], also assumes the DPT to be binary and balanced, i.e. each internal node has exactly two children, although this condition can be easily relaxed. See Figure 2 and its reduced form in Appendix B.3 for a simple example. Further, in Appendix B.4, we give an example of an LTM that is not a TMM.

Hidden Markov Models (HMM) [3; 25] HMM is a strict subclass of LTM. [14] recently observed that HMM is equivalent to the tensor-train format, a special subclass of dHTF$_+$ where the DPT is binary and completely "imbalanced." See Appendix B.5 for a simple example. In some sense, TMM and HMM are the two opposite extremes within dHTF$_+$ (or equivalently LTM).

Further, in Appendix B.6 we give an example of an S^{3}PN that is not an LTM, and in Appendix B.7, we give an example of an SPN that is not an S^{3}PN, leading to the following summary:

Theorem 3.1. *{TMM, HMM} $\subseteq$ LTM = dHTF$_+$ $\subseteq$ HTF$_+$ = S^{3}PN $\subseteq$ SPN, in the sense that we can convert in linear time from a lower representation class to an upper one, without any increase in size.*

[2] All of our illustrations of S^{3}PN in the main text are drawn with some redundant leaves, for the sake of making the self-similar property apparent. See Appendix B for the reduced (but equivalent) counterparts.

[3] As an evidence, we note that the recent survey [21] on LTMs did not mention SPNs at all.

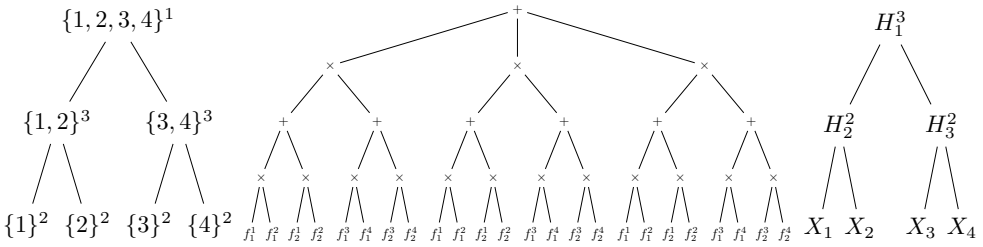


Figure 2: Left: A dimension-partition tree in HTF. The superscripts indicate the number of bases, which should remain constant on each level. Middle: The equivalent S^3PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent TMM. The superscripts indicate the number of values each hidden variable can take (again, remaining constant on each level).

It is important to point out one subtlety here: any (complete and decomposable) SPN, if expanded at the root, is a homogeneous mixture (c.f. (1)). Hence, any SPN is even equivalent to an LCM (i.e. an LTM with one hidden variable taking many values, like in Figure 1), at the expense of potentially increasing the size (significantly). Thus, the containment in Theorem 3.1 should be understood under the premise of not increasing the representation size. It would be interesting to understand if the containment is strict if only polynomial increase in size is allowed. We provide more comparing examples in Appendix B for different models, and in the next section we discuss the (huge) size consequence from converting a certain upper representation class to some lower one.

4 Depth Separation

In the previous section, we established relationships among different representation schemes for homogeneous density mixtures. In this section, we prove an exponential separation in size when converting one representation to another and extend the results in [10; 18; 7; 26]. The key is to exploit the equivalence to HTF, which allows us to bound the model size using linear algebra.

We call a (complete and decomposable) SPN shallow if it has only one sum node, followed by a layer of product nodes. Using the equivalence in Section 3, we know a shallow SPN (trivially self-similar) is equivalent to an LCM (a latent tree model with one hidden node taking as many values as the number of product nodes), or an HTF_+ whose DPT has depth 1 (c.f. Figure 1). Recall that $\mathrm{rank}_+(\mathcal{W})$ denotes the nonnegative rank of a tensor and $\mathsf{nnz}(\mathcal{W})$ is the number of nonzeros (c.f. Appendix A). The leaf nodes in SPN (LTM) or the leaf bases in HTF are either from $\mathcal{F}$ (union of *linearly independent* component densities) or $\mathcal{G}$ (the convex hull), see the definitions in Section 2.

Our first result characterizes the model capacity of shallow SPNs (LCMs):

Theorem 4.1. *If a shallow SPN $\mathfrak{T}$, with leaf (input) nodes from $\mathcal{G}$, represents the density mixture $\mathcal{W}$, then $\mathfrak{T}$ has at least $\mathrm{rank}_+(\mathcal{W})$ many product nodes. Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using $\mathrm{rank}_+(\mathcal{W})$ product nodes and 1 sum node.*

In other words, the nonnegative rank characterizes the smallest size of shallow SPNs (LCMs) that represent the density mixture $\mathcal{W}$. Similarly, we can prove the following result when the leaf nodes are from $\mathcal{F}$ instead of the convex hull $\mathcal{G}$.

Theorem 4.2. *If a shallow SPN $\mathfrak{T}$, with leaf nodes from $\mathcal{F}$, represents the density mixture $\mathcal{W}$, then either $\mathfrak{T}$ has at least $\mathsf{nnz}(\mathcal{W})$ product nodes or $\mathrm{rank}_+(\mathcal{W}) = 1$. Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using $\mathsf{nnz}(\mathcal{W})$ product nodes and 1 sum node.*

Note that we always have $\mathrm{rank}(\mathcal{W}) \le \mathrm{rank}_+(\mathcal{W}) \le \mathsf{nnz}(\mathcal{W})$, thus the lower bound in Theorem 4.2 is stronger than that in Theorem 4.1. This is not surprising, because an SPN with leaf nodes from $\mathcal{G}$ is the same as an SPN with leaf nodes from $\mathcal{F}$ and with an additional layer of sum nodes appended at the bottom (to perform the convex hull operation). This difference already indicates that an additional layer of sum nodes at the bottom can strictly increase the expressive power of SPNs. This distinction between leaf nodes from $\mathcal{F}$ or from $\mathcal{G}$, to our best knowledge, has not been noted before.

The significance of Theorem 4.1 and Theorem 4.2 is that they give *exact* characterizations of the model size of shallow SPNs, and they pave the way for comparing more interesting models. For convenience, we state our next result in terms of LTMs, but the consequence for dHTFs or SPNs should be clear, thanks to the equivalence in Theorem 3.1.

Theorem 4.3. *Let an LTM T have d observed variables $\mathcal{X} = \{X_1, \ldots, X_d\}$ with parents H_i taking r_i values respectively. Assuming the CPTs of T are sampled from a continuous distribution, then almost surely, the tensor representation $\mathcal{W}$ for T has rank at least*

$$\max_{1 \le m \le d/2} \quad \max_{\{S_1, \ldots, S_m, \bar{S}_1, \ldots, \bar{S}_m\} \subseteq \mathcal{X}} \quad \prod_{i=1}^{m} \min\{r_i, \bar{r}_i, k_i, \bar{k}_i\}, \tag{3}$$

where k_i ($\bar{k}_i$) is the number of (linearly independent) component densities that S_i ($\bar{S}_i$) has, and S_i ($\bar{S}_i$) are non-siblings.

Corollary 4.4. *In addition to the setting in Theorem 4.3, if each observed variable X_i has b sibling observed variables and $r_i \equiv r \le k \equiv k_i$, then the tensor representation $\mathcal{W}$ has rank at least $r^{\lfloor d/b \rfloor}$.*

Corollary 4.5. *In addition to the setting in Theorem 4.3, if each observed variable X_i has no sibling observed variables and $r_i \equiv r \le k \equiv k_i$, then the tensor representation $\mathcal{W}$ has rank at least $r^{\lfloor d/2 \rfloor}$.*

Combining Corollary 4.4 with Theorem 4.2 we conclude that an LTM T with d observed variables X_i where every b of them share the same hidden parent node is equivalent to an LCM T' where the hidden node must take at least $r^{\lfloor d/b \rfloor}$ many values. Note that T has $\Theta(d/b)$ hidden variables, each of them taking r values, thus the total size of the CPTs of T is $\Theta(rd/b)$ while the total size of that of T' is $r^{\lfloor d/b \rfloor}$, an exponential blow-up. By combining Corollary 4.5 with Theorem 4.2 a similar conclusion can be made for converting an HMM into a LCM. Of course, interpretation using SPNs is also readily available: Almost all depth-L S^3PNs ($L \ge 2$) with weights sampled from a continuous distribution can be written as a shallow SPN with necessarily exponentially many product nodes.

To our best knowledge, [10] was the first to construct a polynomial that, while representable by a polynomially-sized depth-$\log d$ SPN, would require exponentially many product nodes if represented by a shallow SPN. However, the deep SPN given in [10, Figure 1] is not complete. Recently, [7] proved that the existence result of [10] is in fact *generic*. However, the results of [7] and subsequent work [26] are limited to full binary trees. In contrast, our general Theorem 4.3 holds for any tree, and we allow non-sibling nodes to take different number of values. As a result, we are able to handle HMMs, the opposite extreme of TMM. Another important point we want to emphasize is that the exponential separation from a shallow (i.e. depth-1) tree can be achieved by increasing the depth by merely 1, as opposed to the depth-$\log d$ constructions in [10; 26].

We end this section by making another observation about Theorem 4.3: It also allows us to compare the model size of LTMs T_1 and T_2 where say T_1, after removing its root R, is a subtree of T_2. Indeed, in this case we need only define the children nodes of R as "observed" variables. Then, T_1 becomes an LCM and T_2 serves as T in Theorem 4.3, with observed variables as the children nodes of R. This essentially extends [7, Theorem 3] from a full binary tree to any tree and allowing non-sibling nodes to take different number of values.

5 Approximate Representation

In the previous section, we proved that homogeneous mixtures representable by "deep" architectures (such as SPN or LTM) of polynomial size cannot be *exactly* represented by a shallow one with sub-exponential size. In this section, we address a more intricate and relevant question: What if we are only interested in an *approximate* representation?

To formulate the problem, let g and h be two homogeneous mixtures with tensor representation $\mathcal{W}$ and $\mathcal{Z}$, respectively. We consider the distance $\mathrm{dist}(g, h) := \|\mathcal{W} - \mathcal{Z}\|$ for some norm $\|\cdot\|$ specified later. Using the characterization in Theorem 4.1 we formulate our approximation problem as follows. Let Δ be a perturbation tensor with $\|\Delta\| \le \epsilon$. What is the minimum value for $\mathrm{rank}_+(\mathcal{W} + \Delta)$, i.e. the size of a shallow SPN? This motivates the following definition adapted from [1]:

$$\epsilon\text{-}\mathrm{rank}_+(\mathcal{W}) = \min\Big\{ \mathrm{rank}_+(\mathcal{W} + \Delta) : \|\Delta\| \le \epsilon \Big\} = \min\Big\{ \mathrm{rank}_+(\mathcal{Z}) : \|\mathcal{Z} - \mathcal{W}\| \le \epsilon \Big\}. \tag{4}$$

In other words, ϵ-rank_+ is precisely the minimum size of a shallow SPN (LCM) that approximates a specified mixture $\mathcal{W}$ with accuracy ϵ. We can similarly define ϵ-rank, where we replace the nonnegative rank with the usual rank in (4). Note that the notion of ϵ-rank depends on the norm $\|\cdot\|$.

ℓ_∞-norm Let the norm in the definition (4) be the usual ℓ_∞ norm, and we signify this choice with the notation ϵ-rank^∞. In this setting, we can prove the following nearly-tight bound on the ϵ-rank.

Theorem 5.1. *Fix $\epsilon > 0$ and tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$. Then, for some (small) constant $c > 0$,*

$$\epsilon\text{-rank}^{\infty}(\mathcal{W}) \leq \tfrac{c\|\mathcal{W}\|_{\mathrm{tr}}}{\epsilon^2}, \tag{5}$$

where $\|\mathcal{W}\|_{\mathrm{tr}}$ is the tensor trace norm. A similar result holds for ϵ-$\mathrm{rank}_{+}^{\infty}(\mathcal{W})$. The dependence on ϵ is tight up to a log factor.

Note that the representative tensor $\mathcal{W}$ for a homogeneous density mixture f is nonnegative and sums to 1, in which case $\|\mathcal{W}\|_{\mathrm{tr}} \leq \|\mathcal{W}\|_1 = 1$. Thus, very surprisingly, Theorem 5.1 confirms that any deep SPN (or any LTM or HTF$_+$) can be approximated by some shallow SPN with accuracy ϵ under the ℓ_∞ metric and with at most c/ϵ^2 many product nodes. Of course, this does not contradict with the impossibility results in [7] and [18], because the accuracy ϵ there is exponentially small.

Theorem 5.1 remains mostly of theoretical interest, though, because (i) a straightforward application of Theorem 5.1 leads to a disappointing bound on the total variational distance between the two homogeneous mixtures f and g, due to scaling by the big constant $\prod_i k_i$; (ii) in practical applications we do not have access to $\mathcal{W}$ so the constructive algorithm in our proof does not apply.

KL divergence In contrast to the above ℓ_∞ approximation, we now give an efficient algorithm to approximate a homogeneous density mixture h, using a classic idea of [17]. We propose to estimate h by minimizing the KL divergence over the convex hull[4] of a hypothesis class H:

$$\min_{\mathcal{W}_g \in \mathrm{conv}(\mathsf{H})} \mathsf{KL}(h\|g), \tag{6}$$

where $\mathsf{KL}(h\|g) := \int h(\mathbf{x}) \log \frac{h(\mathbf{x})}{g(\mathbf{x})} \mathrm{d}\mu(\mathbf{x})$, and $\mathcal{W}_g$ is the representative tensor for the mixture g. Following [17], we apply the conditional gradient algorithm [12] to solve (6): Given g_{t-1}, we find

$$(\eta_t, f_t) \leftarrow \arg \min_{\eta \in [0,1], \mathcal{W}_f \in \mathsf{H}} \mathsf{KL}(h\|(1-\eta)g_{t-1} + \eta f), \qquad g_t \leftarrow (1-\eta_t)g_{t-1} + \eta_t f_t. \tag{7}$$

One can also simply set $\eta_t = \frac{2}{2+t}$, as is common in practice. Note that (7) can be approximately solved based on an iid sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ hence is practical:

$$\max_{\eta \in [0,1], \mathcal{W}_f \in \mathsf{H}} \sum_{i=1}^{n} \log[(1-\eta)g_{t-1}(\mathbf{x}_i) + \eta f(\mathbf{x}_i)]. \tag{8}$$

Using basically the same argument as in [17], the above algorithm enjoys the following guarantee:

$$\mathsf{KL}(h\|g_t) \leq c_h \delta / t, \tag{9}$$

where $\delta = \sup\{\log \frac{\langle \mathcal{W}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle}{\langle \mathcal{Z}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle} : \mathcal{W}, \mathcal{Z} \in \mathsf{H}, \mathbf{x} \in \mathbb{X}\}$, and

$$c_h = \min\{p \geq 0 : \mathcal{W}_h = \textstyle\sum_{i=1}^{p} \lambda_i \mathcal{W}_i, \mathcal{W}_i \in \mathsf{H}, \boldsymbol{\lambda} \geq 0, \mathbf{1}^\top \boldsymbol{\lambda} = 1\} \tag{10}$$

is essentially the rank of the mixture h (with tensor representation $\mathcal{W}_h$) w.r.t. the class H.

The important conclusion we draw from the above bound (9) is as follows: First, the constant c_h is no larger than $\prod_i k_i$ if H is any of the classes in Theorem 3.1 (since we only consider *finite* homogeneous mixtures h). Second, if the target density h is a small number of combinations of densities in H, then c_h is small and we can approximate h using the algorithm (7) efficiently. Third, c_h can be vastly different for different hypothesis classes H, as shown in Section 4. For instance, if h is a generic TMM and H is the shallow class LCM, then c_h is exponential in d, whereas if H is the class TMM, then c_h can be as small as 1. There is a trade-off though, since solving (8) for a simpler class (such as LCM) is easier than a deeper one (such as TMM). We will verify this trade-off in our experiments.

6 Experiments

We perform experiments on both synthetic and real world data to reinforce our theoretical findings. Firstly, we present experiments on synthetic data to demonstrate the expressive power of an SPN and the algorithm proposed in (7)-(8) which we call SPN-CG. Next, we present two sets of experiments on real world datasets and present results for image classification under missing data.

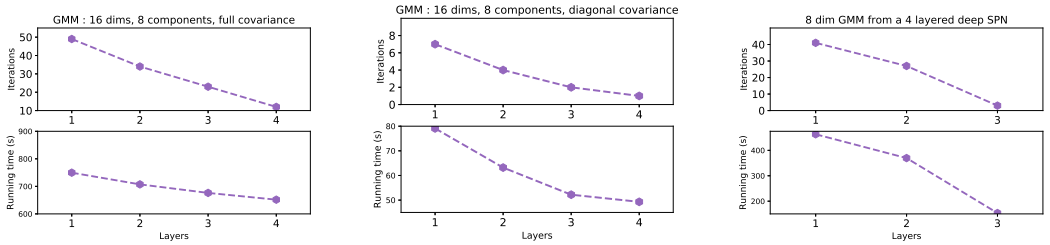


Figure 3: Depth efficiency and performance of SPN-CG

Synthetic data Firstly, in appendix F.1 we confirm that a Gaussian mixture model (GMM) with full covariance matrices can be well approximated by a homogeneous mixture model represented by an SPN learned using SPN-CG. Secondly, we generate 20,000 samples from a 16 dimensional GMM under three different settings - (i) 8 component GMM with full covariance matrices, (ii) 8 component GMM with diagonal covariance matrices and, (iii) GMMs represented by a deep SPN with 4 layers - and estimate each using SPN-CG. We consider layers, $L \in \{1, 2, 3, 4\}$ where $L = 1$ corresponds to a shallow network and $L = 4$ corresponds to a network in TMM (a full binary tree). For each L, at every iteration of SPN-CG we add a network with L layers. In Figure 3, we plot the number of iterations and the total running time until convergence w.r.t. the depth for each setting described above. We make the following observations: As the depth (layer) increases, the number of iterations decreases sharply, since adding a deeper network effectively is the same as adding exponentially many shallower networks (confirming Section 4). Moreover, although learning a deeper network in each iteration is more expensive than learning a shallower network, the sharp decrease in iterations full compensates this overhead and leads to a much reduced total running time. The advantage in using deeper networks is more pronounced when the data is indeed generated from a deep model.

Image Classification under Missing Data by Marginalization A natural setting to test the effectiveness of generative models like deep SPNs is for classification in the regime of missing data. Generative models can cope with missing data naturally through marginalizing the missing values, effectively learning all possible completions for classification. As stated earlier, SPNs are attractive because inference, marginalization and evaluating conditionals is tractable and amounts to one pass through the network. This is in stark contrast with discriminative models that often rely on either data imputation techniques (which result in sub-optimal classification) or by assuming the distribution of missing values is same during train and test time; an assumption that is often not valid in practice.

We perform experiments on MNIST [15] for digit classification and small NORB [16] for 3D object recognition under the MAR (missing at random) regime as described in [26] (Section 3). We experiment with two missing distributions- (i) an i.i.d. mask with a fixed probability of missing each pixel, and (ii) a mask obtained by the union of rectangles of a certain size, each positioned uniformly at random in the image. Concretely, let $P(\mathrm{X}, \mathrm{Y})$ be the joint distribution over the images ($\mathrm{X} \in \mathbb{R}^d$) and labels $\mathrm{Y} \in [M]$. Further, let Z be a random binary vector conditioned on $\mathrm{X} = \mathbf{x}$ with distribution $Q(\mathrm{Z}|\mathrm{X} = \mathbf{x})$. To generate images with missing pixels, we sample $\mathbf{z} \in \{0, 1\}^d$ and consider the vector $\mathbf{x} \odot \mathbf{z}$. A pixel $x_i, i \in [d]$ is considered missing if $z_i = 0$ in which case the corresponding coordinate in $\mathbf{x} \odot \mathbf{z}$ holds $*$ and it holds x_i if $z_i = 1$. In the MAR setting that we consider for our experiments, $Q(\mathrm{Z} = \mathbf{z}|\mathrm{X} = \mathbf{x})$ is a function of both $\mathbf{z}$ and $\mathbf{x}$ but is independent of changes to x_i if $z_i = 0$ i.e. Z is independent of missing pixels. As described in [26], the optimal classification rule in the MAR regime is $h^*(\mathbf{x} \odot \mathbf{z}) = P(\mathrm{Y} = y|w(\mathbf{x}, \mathbf{z}))$ where $w(\mathbf{x}, \mathbf{z})$ is the realization when X coincides with $\mathbf{x}$ on coordinates i for which $z_i = 1$.

Our major goal with these experiments is to test our algorithm SPN-CG for high-dimensional real world settings and show the efficacy of learning SPNs by increasing their expressiveness iteratively. Therefore, we directly adapt the experiments as presented in [26]. Specifically, we adapt the code of HT-TMM for our SPN-CG by following the details in [26]. In each iteration of our algorithm, we add an SPN structure exactly similar to HT-TMM. Therefore, the first iteration of our algorithm (i.e. SPN-CG1) amounts to a structure similar to HT-TMM while additional iterations increase the network capacity. For each iteration, we train the network using an AdamSGD variant with a base learning rate of 0.03 and momentum parameters $\beta_1 = \beta_2 = 0.9$. For each added network structure, we train the model for 22,000 iterations for MNIST and 40,000 for NORB.

[4]This is similar in spirit to [20; 2] which learn mixture of trees, but the algorithms are quite different.

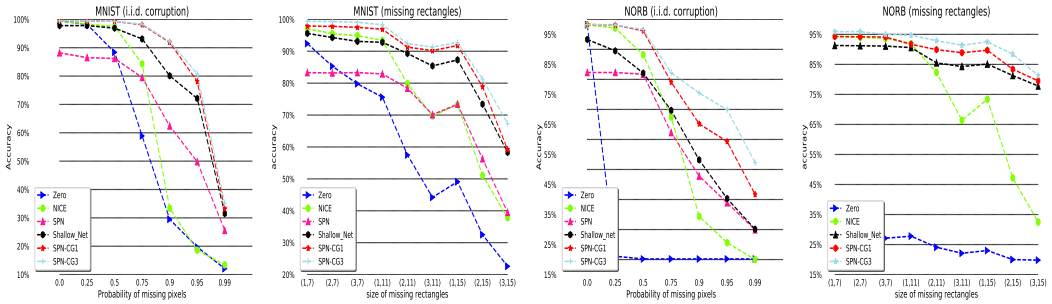


Figure 4: Performance of SPN-CG for missing data on MNIST and NORB

Due to space limit Figure 4 only presents results comparing our model with (i) data imputation techniques that complete missing pixels with zeros or NICE [11], a generative model suited for inpainting, and finally using a ConvNet for prediction, (ii) an SPN with structure learned using data as proposed in [24] augmented with a class variable to maximize joint probability, and (iii) shallow networks to demonstrate the benefits of depth. A more comprehensive figure showing comparisons with several other algorithms is given in appendix F.2, along with details.

SPN-CG1 and SPN-CG3 in Figure 4 stand for one and three iterations of our algorithm respectively. The results show that SPN-CG performs well in all regimes of missing data for both MNIST and NORB. Furthermore, other generative models including SPN with structure learning perform comparably only when a few pixels are missing but perform very poorly as compared to SPN-CG when larger amounts of data is missing. Our results here complement those in [26] where these experiments were first reported with state of the art results.

7 Conclusion

We have formally established the relationships among some popular unsupervised learning models, such as latent tree graphical models, hierarchical tensor formats and sum-product networks, based on which we further provided a unified treatment of exponential separation in *exact* representation size between deep architectures and shallow ones. Surprisingly, for *approximate* representation, the conditional gradient algorithm can approximate any homogeneous mixture within accuracy ϵ by combining $O(1/\epsilon^2)$ shallow models, where the hidden constant may decrease exponentially wrt the depth. Experiments on both synthetic and real datasets confirmed our theoretical findings.

Acknowledgement

The authors gratefully acknowledge support from the NSERC discovery program.

References

[1] Noga Alon. Perturbed identity matrices have high rank: Proof and applications. *Combinatorics, Probability and Computing*, 18(1-2):3–15, 2009.

[2] Animashree Anandkumar, Daniel Hsu, Furong Huang, and Sham M. Kakade. Learning mixtures of tree graphical models. In *Advances in Neural Information Processing Systems*, 2012.

[3] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

[4] Richard Caron and Tim Traynor. The zero set of a polynomial. Technical report, 2005.

[5] Myung Jin Choi, Vincent Y. F. Tan, Animashree Anandkumar, and Alan S.Willsky. Learning latent tree graphical models. *Journal of Machine Learning Research*, 12:1771–1812, 2011.

[6] Nadav Cohen, Or Sharir, Yoav Levine, Ronen Tamari, David Yakira, and Amnon Shashua. Analysis and design of convolutional networks via hierarchical tensor decompositions, 2017. arXiv:1705.02302v4.

[7] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[8] Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *ICML*, 2016.

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

[10] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

[11] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[12] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[13] Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Springer, 2012.

[14] M. Ishteva. Tensors and latent variable models. In *The 12th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 49–âĂŞ55, 2015.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE, 2004.

[17] Jonathan Q Li and Andrew R Barron. Mixture density estimation. In *Advances in neural information processing systems*, pages 279–285, 2000.

[18] James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014.

[19] Geoffrey McLachlan and David Peel. *Finite mixture models*. John Wiley & Sons, 2004.

[20] Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

[21] Raphaël Mourad, Christine Sinoquet, Nevin L. Zhang, Tengfei Liu, and Philippe Leray. A survey on latent tree models and applications. *Journal of Artificial Intelligence Research*, 47:157–203, 2013.

[22] Hien D Nguyen and Geoffrey J McLachlan. On approximations via convolution-defined mixture models. *arXiv preprint arXiv:1611.03974*, 2016.

[23] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2017.

[24] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Uncertainty in Artificial Intelligence*. UAI, 2011.

[25] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[26] Or Sharir, Ronen Tamari, Nadav Cohen, and Amnon Shashua. Tensorial mixture models, 2018. arXiv:1610.04167v5.

[27] Le Song, Haesun Park, Mariya Ishteva, Ankur Parikh, and Eric Xing. Hierarchical tensor decomposition of latent tree graphical models. In *ICML*, 2013.

[28] Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer, 2009.

[29] Han Zhao, Mazen Melibari, and Pascal Poupart. On the relationship between sum-product networks and bayesian networks. In *International Conference on Machine Learning*, pages 116–124, 2015.

Supplementary Material

Deep Homogeneous Mixture Models: Representation, Separation, and Approximation

A Tensor Background

For any natural number d, we denote $[d] := \{1, \dots, d\}$. Let $\mathsf{V}_i, i \in [d]$, be k-dimensional vector spaces over the real field $\mathbb{R}$, then the tensor product $\mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$ is the canonical vector space that linearizes multilinear maps over the product space $\mathsf{V}_1 \times \cdots \times \mathsf{V}_d$. Perhaps the simplest way to construct the tensor product is to first formally define rank-1 tensors as:

$$\{\mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_d : \ \mathbf{v}_i \in \mathsf{V}_i, \ i \in [d]\}, \tag{11}$$

and then take the linear span of rank-1 tensors. For each $\mathscr{T} \in \mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$, we define its rank as

$$\mathrm{rank}(\mathscr{T}) := \min\{r : \mathscr{T} = \sum_{\gamma=1}^{r} \mathbf{v}_1^\gamma \otimes \cdots \otimes \mathbf{v}_d^\gamma, \ \mathbf{v}_i^\gamma \in \mathsf{V}_i, \ i \in [d], \ \gamma \in [r]\}. \tag{12}$$

Sometimes we further restrict each factor $\mathbf{v}_i^\gamma$ to some subset $\mathsf{U}_i \subseteq \mathsf{V}_i$, leading to a "larger" notion of rank. For instance, when $\mathsf{V}_i \equiv \mathbb{R}^k$, the above definition is called the CP-rank and if we take $\mathsf{U}_i = \mathbb{R}_+^k$, then we get the refined notion of nonnegative rank, denoted as rank_+. Obviously, $\mathrm{rank}_+ \geq \mathrm{rank}$ (whenever the former is defined).

Usually we can identify a d-order tensor $\mathscr{T} \in \mathsf{V}_1 \otimes \cdots \otimes \mathsf{V}_d$ with a multi-dimensional array

$$\mathcal{T} = [\mathcal{T}_{i_1,\dots,i_d}]_{i_j \in [k_i], j \in [d]} \in \bigotimes_i \mathbb{R}^{k_i} \simeq \mathbb{R}^{k_1 \times \cdots \times k_d}, \tag{13}$$

once some bases have been chosen for each V_i. We can extend an inner product to the tensor product space: provided that some inner product $\langle \cdot, \cdot \rangle_i$ has been specified on each V_i, we first define the inner product for rank-1 tensors:

$$\langle \mathbf{u}_1 \otimes \cdots \otimes \mathbf{u}_d, \mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_d \rangle := \prod_{i=1}^{d} \langle \mathbf{u}_i, \mathbf{v}_i \rangle_i, \tag{14}$$

and then extend multi-linearly.

We give an explicit description of TMM [26] here. For simplicity, let us assume $d = b^L$ for some integers b and L. Then, every d-order tensor $\mathscr{T}$ can be represented recursively as

$$\phi_\gamma^{\ell,t} = \sum_{j=1}^{r_{\ell-1}} w_j^{\ell,t,\gamma} \bigotimes_{s=1}^{b} \phi_j^{\ell-1,b(t-1)+s}, \ \ \ell \in [L-1], \ \gamma \in [r_\ell], \ t \in [b^{L-\ell}], \tag{15}$$

$$\mathscr{T} = \phi_1^{L,1} = \sum_{j=1}^{r_{L-1}} w_j^{L,1,1} \bigotimes_{s=1}^{b} \phi_j^{L-1,s}, \tag{16}$$

where $\phi_\gamma^{0,i} \in \mathsf{V}_i$ for all $\gamma \in [r_0]$. Note that the tensor $\mathscr{T}$ is completely determined by

$$\{\phi_\gamma^{0,i} : \gamma \in [r_0], i \in [d]\} \bigcup \{\mathbf{w}^{\ell,t,\gamma} \in \mathbb{R}^{r_{\ell-1}} : \ell \in [L-1], \gamma \in [r_\ell], t \in [b^{L-\ell}]\} \cup \{\mathbf{w}^{L,1,1} \in \mathbb{R}^{r_{L-1}}\}, \tag{17}$$

where the former are the base vectors at the bottom level and the latter are the coefficient vectors at each intermediate level. Note that the representation (15)-(16) is not 1-1 (hence some redundancy). Let TMM_r^b (with default $\mathsf{TMM}_r := \mathsf{TMM}_r^2$) be the class of tensors that can be represented as in (15)-(16).

A simple counting argument reveals that the coefficient tensors in TMM_r^b have $\frac{d-b}{b-1}r^2 + r$ entries. It is clear that $\mathsf{TMM}_r^b \subseteq \mathsf{TMM}_{r+1}^b$, and TMM_1^b is exactly the set of rank-1 tensors. As shown in [13], every tensor of rank r can be represented in TMM_r^b. Similarly, every tensor of nonnegative rank r can be represented in TMM_r^b, with all base vectors and coefficient vectors in (17) nonnegative. Moreover, we can normalize the base vectors $\phi_\gamma^{0,i}$ so that they have unit ℓ_1 norm.

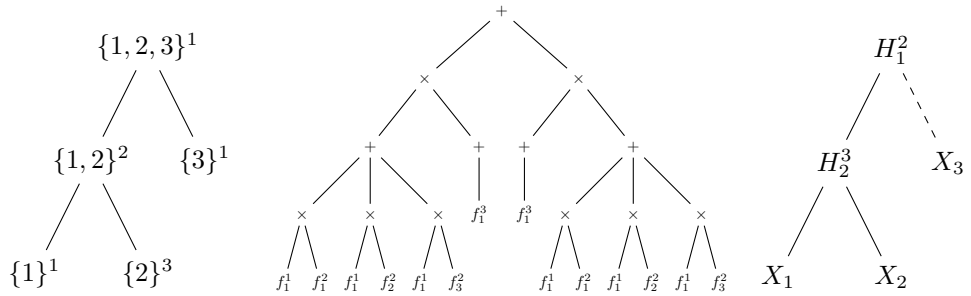


Figure 5: Left: A dimension-partition tree in HTF$_+$. The superscripts indicate the number of bases. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An "equivalent LTM." The superscripts indicate the number of values each hidden variable can take. The two densities of X_3 are equal, i.e. $f_1^3 = f_2^3$ (hence X_3 does not actually depend on H_1).

B More results on comparing different models

This appendix section provides more details to compliment section 4. We provide additional details and examples to support the arguments that we made in section 4.

B.1 Converting an LTM to S^{3}PN

Given an LTM, we can build a corresponding S^{3}PN as follows: starting from the root of the LTM, for each hidden variable H that takes k possible values $\{1, \ldots, k\}$ and that has r children nodes $\{V_1, \ldots, V_r\}$, we create a sum node S_H with k children product nodes $\{\mathsf{P}_{H,1}, \ldots \mathsf{P}_{H,k}\}$, each of which has r children sum nodes $\{\mathsf{S}_{V_1}, \ldots, \mathsf{S}_{V_r}\}$. We set the weight from the sum node S_H to its i-th child product node $\mathsf{P}_{H,i}$ as $\Pr(H = i|\pi(H) = j)$, if S_H connects to the j-th child product node of the parent hidden variable $\pi(H)$ (for the root, the parent is empty). If the child V_t is a hidden variable, we continue the construction similarly, while if $V_t = X_i$ is an observed variable, then we replace the sum node S_{V_t} with the density $f_j^i(x_i)$, assuming S_{V_t} is connected to the j-th child product node of the parent hidden variable H. Algorithm 1 summarizes this construction, and Figure 1 illustrates the idea using a simple latent class model (LCM) [21].

In Algorithm 1, we describe a procedure to convert a latent tree model (LTM) as described in (2) to a self-similar SPN (S^{3}PN). In Figure 6 we give another example to illustrate Algorithm 1.

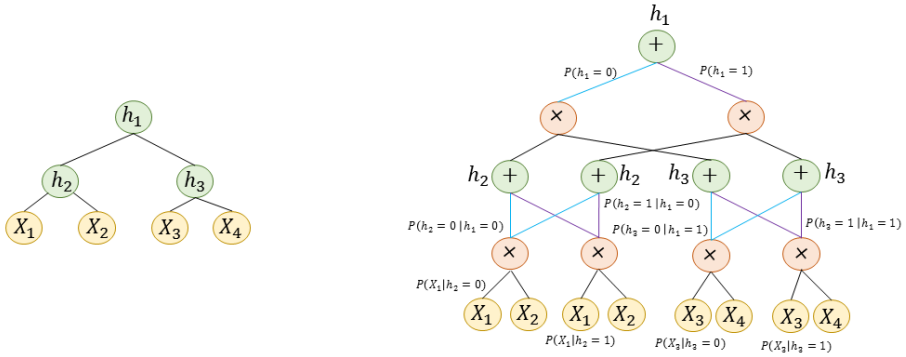


Figure 6: Left shows a latent tree model with three binary hidden variables $\mathbf{H} = \{h_1, h_2, h_3\}$ and four observed variables $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$. The second figure shows the equivalent SPN representing the latent tree. The blue edges imply that the hidden variable takes value 0 and the violet edges mean it takes value 1. Only a subset of leaf distributions are explicitly shown in the figure.

In Figure 6, we consider a latent tree graphical model forming a balanced binary tree with three binary hidden variables $\mathbf{H} = \{h_1, h_2, h_3\}$ and four observed variables $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$. The tree has 3 levels and is rooted at h_1. The algorithm proceeds by going through each level one at a time. In the first iteration, it encounters the root node h_1 and creates a corresponding sum node in the SPN. It then creates two (equal to all possible states of h_1) product nodes as children to this sum node.

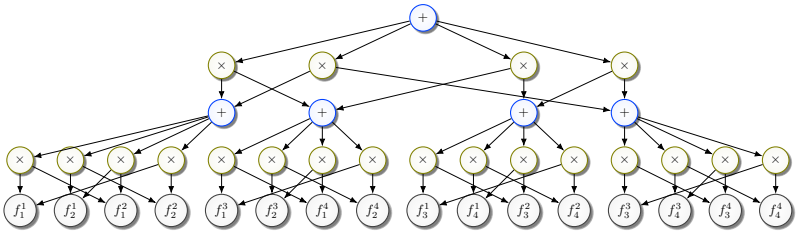
The edge on the left (blue in the figure) denotes the edge when $h_1 = 0$ and has weight $\Pr(h_1 = 0)$ and the edge on the right (violet) denotes edge when $h_1 = 1$ and has weight $\Pr(h_1 = 1)$. In the next iteration, the algorithm proceeds to level 2 which has two hidden variables h_2 and h_3. The algorithm processes these one at a time. First, it takes h_2 and creates two sum nodes corresponding to h_2, one child each for each product node in the previous layer. Next, for two product nodes are created and an edge is created between each of these product nodes and each of the sum node created before corresponding to h_2. The same procedure is then repeated for h_3. Finally, for each observed variable, a leaf distribution $\Pr(X|\pi_X)$ is induced.

Algorithm 1 Converting an LTM into an S^{3}PN

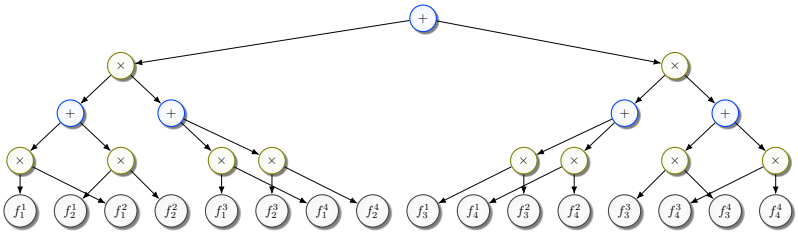
Input : A latent tree model with L levels and $(\mathbf{X}, \mathbf{H})$
Output : An equivalent S^{3}PN
for $l \leftarrow L$ to 1 **do**
  $\mathbf{H}_l :=$ {all nodes in current level from left to right order}
  while $\mathbf{H}_l \neq \emptyset$ **do**
    $h = \text{Pop}(\mathbf{H}_l)$
    if $h \in \mathbf{X}$ **then**
      for $j \leftarrow 1$ to $|\pi_h|$ **do**
        create a leaf v_j with distribution $Pr(h|\pi_h = j)$
        add an edge (v_j, p_j^{l-1})
      end for
    else
      if $\pi_h \neq \emptyset$ **then**
        create $|\pi_h|$ sum nodes i.e. $S_h^l := \{s_1^l, s_2^l, \cdots s_{|\pi_h|}^l\}$
        for $j \leftarrow 1$ to $|\pi_h|$ **do**
          add an edge (s_j^l, p_j^{l-1})
        end for
        create $|h|$ product nodes i.e. $P_h^l := \{p_1^l, p_2^l, \cdots, p_h^l\}$
        for $i \leftarrow 1$ to $|\pi_h|$ **do**
          for $j \leftarrow 1$ to $|h|$ **do**
            create an edge (s_i^l, p_j^l) with weight $w_{i,j} = Pr(h = j|\pi_h = i)$
          end for
        end for
      else
        create one sum node s_h
        create h product nodes i.e. $P_h := \{p_1, p_2, \cdots, p_h\}$
        for $i \leftarrow 1$ to $|h|$ **do**
          add an edge (s_h, p_i) with weight $Pr(h = i)$
        end for
      end if
    end if
  end while
end for

B.2 dHTF $\subsetneq$ HTF

Figure 7 shows the difference between HTF and dHTF. As is clear from the figure, dHTF allows for only local connections and can be thought of as having pointwise multiplication between bases densities. HTF is more general and can allow for cross connections.



(a) HTF



(b) dHTF

Figure 7: Top : A general HTF representation. The network has cross connections and calculates all possible multiplications. Bottom : A dHTF with same bases functions. The representation allows for local connections.

B.3 Example for TMM as an LTM and S^{3}PN

In fig. 8, we give a representation for fig. 2 without redundancy. The figure shows that a TMM can be represented by an LTM and hence an S^{3}PN.

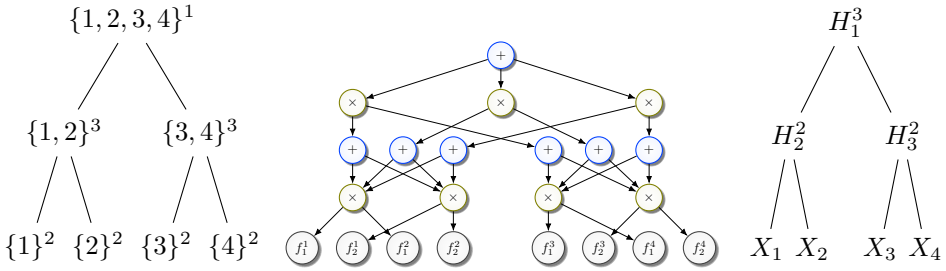


Figure 8: Left: A dimension-partition tree in HTF. The superscripts indicate the number of bases, which should remain constant on each level. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent TMM. The superscripts indicate the number of values each hidden variable can take (again, remaining constant on each level).

B.4 Example for TMM $\subsetneq$ LTM

In fig. 9 we give an S^{3}PN that is equivalent to an LTM but not a TMM. It is evident from the figure that the LTM consists of hidden variables at the same level with different number of possible states. This arrangement, however, is not allowed in TMM.

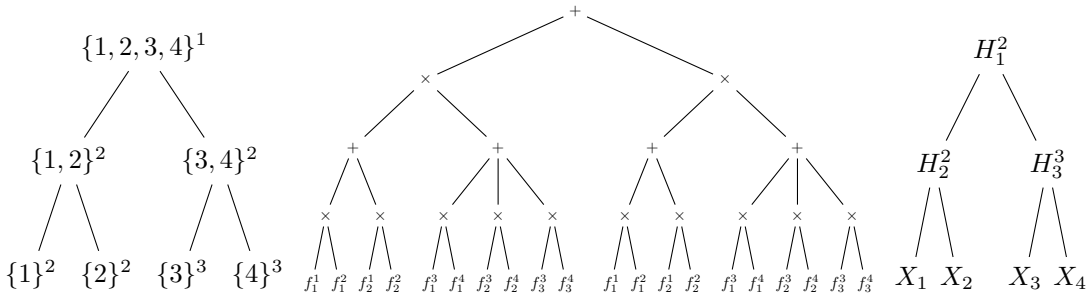


Figure 9: Left: A dimension-partition tree in HTF$_+$. The superscripts indicate the number of bases. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent LTM. The superscripts indicate the number of values each hidden variable can take.

A compact representation of fig. 9 without redundancy is given in fig. 10.

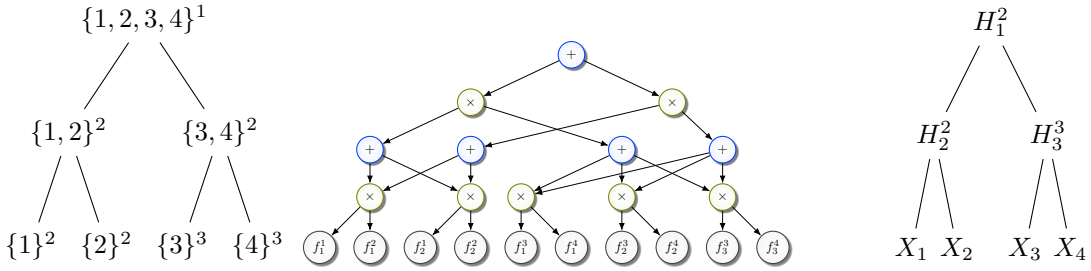


Figure 10: Left: A dimension-partition tree in HTF$_+$. The superscripts indicate the number of bases. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent LTM. The superscripts indicate the number of values each hidden variable can take.

B.5 TTM/HMM as LTM and S^{3}PN

In fig. 11, we give an example of a Tensor Train Model (TTM) which is know to be equivalent to an HMM. We show an equivalent representation of the TTM/HMM into an LTM and therefore an S^{3}PN.

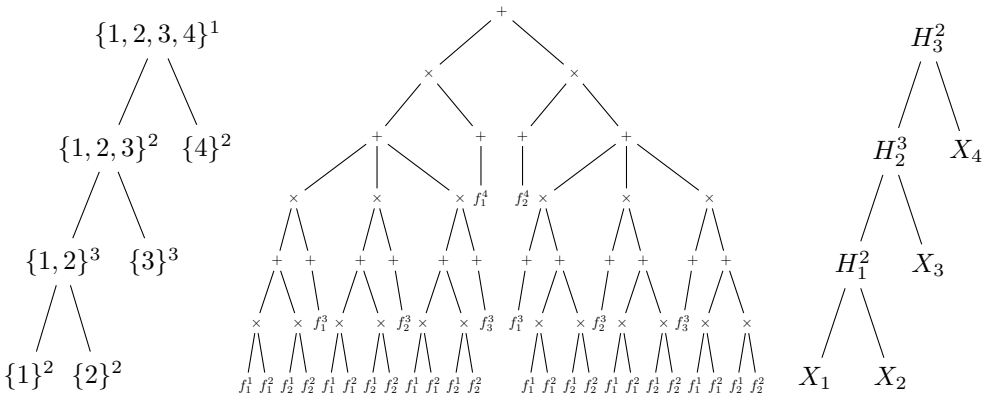


Figure 11: Left: A dimension-partition tree in tensor-train. The superscripts indicate the number of bases, which should remain constant for siblings. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An equivalent HMM. The superscripts indicate the number of values each hidden variable can take.

Figure 12 shows a simpler example to convert a TTM/HMM to an LTM and S^{3}PN with no redundancy.

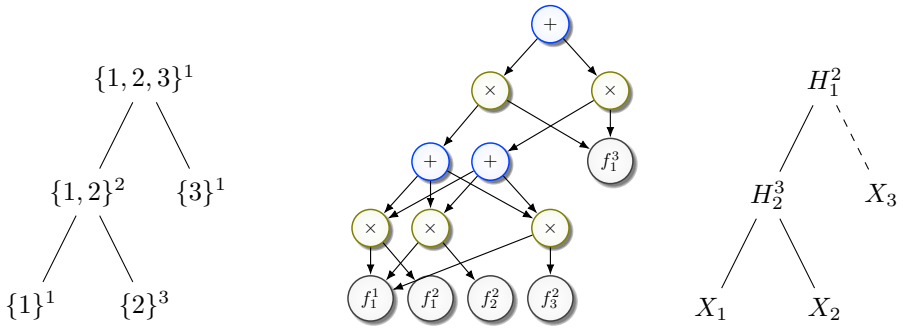


Figure 12: Left: A dimension-partition tree in HTF$_+$. The superscripts indicate the number of bases. Middle: The equivalent S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: An "equivalent LTM." The superscripts indicate the number of values each hidden variable can take. The two densities of X_3 are equal, i.e. $f_1^3 = f_2^3$ (hence X_3 does not actually depend on H_1).

B.6 Example for LTM $\subsetneq$ S^{3}PN

In **??** we give an example of an S^{3}PN whose resulting latent model has cycles and hence cannot be represented as an LTM without increasing the size of the Latent tree exponentially w.r.t. to the size of the latent model.

B.7 Example for S^{3}PN $\subsetneq$ SPN

In fig. 13, we give an example of an SPN that is not equivalent to S^{3}PN. It is evident from the example that the subtrees rooted at the first sum node have different variable partitions and hence the resulting SPN is not sn S^{3}PN. The figure on the right shows that converting this SPN to an S^{3}PN will result in an increase in the size of the network.

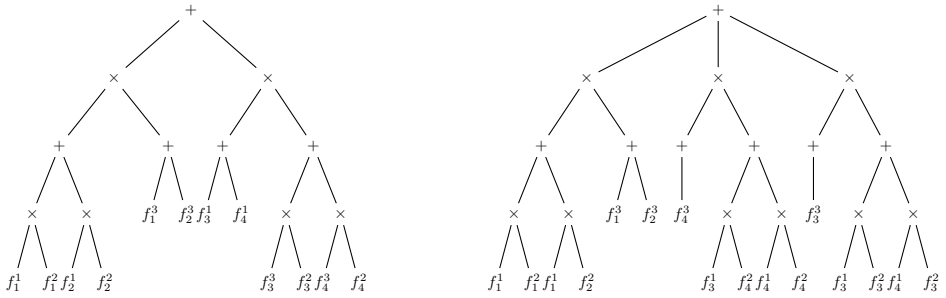


Figure 13: Left: An SPN but which is not an S^{3}PN. The leaf f_j^i is the j-th basis of vector space V_i. Right: The equivalent S^{3}PN requires an increase in the size of the network.

C Proofs

Theorem C.1. *Any SPN can be rearranged to have alternating layers of sum and product nodes without any change in the size of the resultant standard SPN from the original SPN.*

Proof. It is straightforward to show that consecutive combination of either sum nodes or product nodes can be merged/collapsed into one layer of the corresponding nodes. This can be seen as follows: consider a sum node v that has m sum nodes as children and denote the set as $ch(v) := \{v_i\}_{i=1}^m$. Then, the expression f_v evaluated at v is

$$f_v(\mathbf{x}) = \sum_{i=1}^{m} \alpha_{v_i} f_{v_i}(\mathbf{x}) \tag{18}$$

However, since each $v_i \; \forall i \in [m]$ is also a sum node; denote the children of v_i by the set $ch(v_i) := \{\hat{v}_{i,j}\}_{j=1}^{t_i}$ for each $i \in [m]$. Thus,

$$f_{v_i}(\mathbf{x}) = \sum_{j=1}^{t_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{19}$$

Therefore, $f_v(\mathbf{x})$ can be now be re-written as

$$f_v(\mathbf{x}) = \sum_{i=1}^{m} \alpha_{v_i} \sum_{j=1}^{t_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{20}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{t_i} \alpha_{v_i} \beta_{\hat{v}_{i,j}} f_{\hat{v}_{i,j}} \tag{21}$$

$$\tag{22}$$

Define a $1-1$ mapping between the tuple (i,j) $i \in [m],\ j \in [t_i]$ and $[K]$ where $K = \sum_{i=1}^{m} t_i$ such that $k = j + \sum_{l=1}^{i-1} t_l,\ k \in [K]$. Then, we can re-write the above as

$$f_v(\mathbf{x}) = \sum_{k=1}^{K} \gamma_k f_{\hat{v}_K} \tag{23}$$

where $\gamma_k = \alpha_{v_i}\beta_{\hat{v}_{i,j}}$ and $f_{\hat{v}_k} = f_{\hat{v}_{i,j}}$. This shows that two consecutive layers of sum node can be collapsed into one layer of sum layer while preserving the same size of the network. Similarly, it can be shown for consecutive layers of product nodes.

Now, we give the procedure to convert any SPN into an SPN with alternating layers of sums and products. Perform a top-down pass starting at the root node (W.l.o.g. assume the root node is a sum node). For every children of the root node, if it is a sum node, merge the node into the root node. This ensures that after this step the top layer and the next layer are alternating (including leaf nodes). Proceeding similarly for every node in the network ensures the final network has alternating layers throughout. This completes the proof. □

Theorem 4.1. *If a shallow SPN $\mathfrak{T}$, with leaf (input) nodes from $\mathcal{G}$, represents the density mixture $\mathcal{W}$, then $\mathfrak{T}$ has at least* $\mathrm{rank}_+(\mathcal{W})$ *many product nodes. Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using* $\mathrm{rank}_+(\mathcal{W})$ *product nodes and 1 sum node.*

Proof. Suppose the shallow SPN $\mathfrak{T}$ represents the (homogeneous) mixture density $\mathcal{W}$. If the hidden layer is all sum nodes, then the output node must be a product node. The claim trivially holds in this case. If the hidden layer is r product nodes, then the output node is a sum node, with weight z_γ to the γ-th product node. The output of the SPN $\mathfrak{T}$, when expanded at the root, is in the following form:

$$\mathfrak{T}(\mathbf{x}) = \sum_{\gamma=1}^{r} z_\gamma \prod_{i=1}^{d} g_i^{\gamma}(x_i) = \sum_{\gamma=1}^{r} z_\gamma \langle \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle \tag{24}$$

$$= \langle \sum_{\gamma=1}^{r} z_\gamma \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle \tag{25}$$

$$= \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{26}$$

Thus, $\mathcal{W} = \sum_{\gamma=1}^{r} z_\gamma \cdot \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}$, i.e., $\mathrm{rank}_+(\mathcal{W}) \le r$.

Conversely, let $r = \mathrm{rank}_+(\mathcal{W})$ with the decomposition $\mathcal{W} = \sum_{\gamma=1}^{r} \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}$. Note that each $\mathbf{w}_i^{(\gamma)}$ is nonzero, as otherwise we would be able to reduce the rank. We construct a shallow SPN $\mathfrak{T}$ to represent $\mathcal{W}$ as follows: On the first layer we have r product nodes, with the γ-th one computing $\prod_{i=1}^{d} g_i^{(\gamma)}(x_i)$, where

$$g_i^{(\gamma)}(x_i) = \sum_{j=1}^{k} \bar{w}_{ij}^{(\gamma)} f_{i,j}(x_i), \quad \bar{w}_{ij}^{(\gamma)} = \frac{w_{ij}^{(\gamma)}}{\|\mathbf{w}_i^{(\gamma)}\|_1}. \tag{27}$$

Note that $\|\mathbf{w}_i^{(\gamma)}\|_1 := \sum_{j=1}^{k} w_{ij}^{(\gamma)} > 0$ hence the above is well-defined. Then, we add a sum node on top of all product nodes, with weight $\|\mathbf{w}^{(\gamma)}\|_1 := \prod_{i=1}^{d} \|\mathbf{w}_i^{(\gamma)}\|_1 > 0$ for the γ-th product node. The output of the constructed shallow SPN is:

$$f(\mathbf{x}) = \sum_{\gamma=1}^{r} \|\mathbf{w}^{(\gamma)}\|_1 \prod_{i=1}^{d} g_i^{(\gamma)}(x_i) = \sum_{\gamma=1}^{r} \prod_{i=1}^{d} \sum_{j=1}^{k} w_{ij}^{(\gamma)} f_{i,j}(x_i) \tag{28}$$

$$= \sum_{\gamma=1}^{r} \langle \mathbf{w}_1^{(\gamma)} \otimes \cdots \otimes \mathbf{w}_d^{(\gamma)}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle = \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{29}$$

The proof is now complete. □

Theorem 4.2. *If a shallow SPN $\mathfrak{T}$, with leaf nodes from $\mathcal{F}$, represents the density mixture $\mathcal{W}$, then either $\mathfrak{T}$ has at least* $\mathsf{nnz}(\mathcal{W})$ *product nodes or* $\mathrm{rank}_+(\mathcal{W}) = 1$. *Conversely, there always exists a shallow SPN that represents $\mathcal{W}$ using* $\mathsf{nnz}(\mathcal{W})$ *product nodes and 1 sum node.*

Proof. Suppose $\mathfrak{T}$ has a hidden layer of sum nodes. Because $\mathfrak{T}$ is standard, the product output is then a mixture density of the following form:

$$\mathfrak{T}(\mathbf{x}) = \prod_{i=1}^{d} \sum_{j=1}^{k} w_{ij} f_{i,j}(x_i) = \langle \mathbf{w}_1 \otimes \cdots \otimes \mathbf{w}_d, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle = \langle \mathcal{W}, \vec{f}_1 \otimes \cdots \otimes \vec{f}_d \rangle. \tag{30}$$

Thus, $\mathcal{W} = \mathbf{w}_1 \otimes \cdots \otimes \mathbf{w}_d$ has nonnegative rank 1. On the other hand, if $\mathfrak{T}$ has a hidden layer of product nodes, then the output of the standard SPN $\mathfrak{T}$, when expanded at the root sum node, is in the following form:

$$\mathfrak{T}(\mathbf{x}) = \sum_{j_1=1}^{k} \cdots \sum_{j_d=1}^{k} z_{j_1,\ldots,j_d} \prod_{i=1}^{d} f_{i,j_i}(x_i) = \langle \mathcal{W}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d) \rangle. \tag{31}$$

Thus, $\mathcal{W} = \mathcal{Z}$, in particular $\mathsf{nnz}(\mathcal{W}) = \mathsf{nnz}(\mathcal{Z})$, but the latter is exactly the number of product nodes in $\mathfrak{T}$.

The converse part follows by reversing the above argument. □

Theorem 4.3. *Let an LTM T have d observed variables $\mathcal{X} = \{X_1, \ldots, X_d\}$ with parents H_i taking r_i values respectively. Assuming the CPTs of T are sampled from a continuous distribution, then almost surely, the tensor representation $\mathcal{W}$ for T has rank at least*

$$\max_{1 \le m \le d/2} \max_{\{S_1,\ldots,S_m,\bar{S}_1,\ldots,\bar{S}_m\} \subseteq \mathcal{X}} \prod_{i=1}^{m} \min\{r_i, \bar{r}_i, k_i, \bar{k}_i\}, \tag{3}$$

where k_i ($\bar{k}_i$) is the number of (linearly independent) component densities that S_i ($\bar{S}_i$) has, and S_i ($\bar{S}_i$) are non-siblings.

Proof. We present our proof using the equivalence between LTM and dHTF.

Recall that an HTF consists of a dimension-partition tree (DPT) T whose leaf nodes $\{i\}, i \in [d]$ represent d vector spaces with bases $\{\mathbf{v}_1^i, \ldots, \mathbf{v}_{r_i}^i\}$ respectively. At each internal node β with b_β children nodes $\beta_1, \ldots, \beta_{b_\beta}$, we have r_β coefficient tensors $\mathbf{w}^{\beta,\ell[\beta]} \in \mathbb{R}^{r_{\beta_1} \times \cdots \times r_{\beta_{b_\beta}}}, \ell[\beta] \in [r_\beta]$, and r_α denotes the number of bases at node α. Any tensor $\mathcal{W}$ living in the space at the root D of T can thus be represented using r_D coefficients $\{c_{\ell[D]} : \ell[D] \in [r_D]\}$ in the following way (c.f. Eq (11.26) of [13] for the special case when the DPT is binary):

$$\mathcal{W} = \sum_{\substack{\ell[i]=1 \\ i \in [d]}}^{r_i} \left[\sum_{\substack{\ell[\alpha]=1 \\ \alpha \in \mathsf{T} \setminus \mathsf{L}}}^{r_\alpha} c_{\ell[D]} \prod_{\beta \in \mathsf{T} \setminus \mathsf{L}} w^{\beta,\ell[\beta]}_{\ell[\beta_1],\ldots,\ell[\beta_{b_\beta}]} \right] \bigotimes_{i=1}^{d} \mathbf{v}^i_{\ell[i]}. \tag{32}$$

For a dHTF, the coefficient tensors $w^{\beta,\ell[\beta]}_{\ell[\beta_1],\ldots,\ell[\beta_{b_\beta}]}$ are diagonal, so in the summation above we can only consider sibling nodes once as a group. The key observation is that the right-hand side of (32) is a sum of many rank-1 tensors, hence $\mathcal{W}$ is likely to have a large rank.

Let $\{S_i, \bar{S}_i : i = 1, \ldots, m\} \subseteq \mathcal{X} := \{X_1, \ldots, X_d\}$, where S_i's are non-siblings and $\bar{S}_i$'s are also non-siblings. Set $t_i = \min\{r_i \bar{r}_i, k_i, \bar{k}_i\}$. For each S_i, set its parent say H_i's (diagonal) coefficient tensor as follows:

$$w^{H_i,\ell[H_i]}_{\ell[S_i]} = \begin{cases} 1, & \text{if } \ell[H_i] = \ell[S_i] = \ell[\bar{S}_i] \le t_i \\ 0, & \text{otherwise} \end{cases}. \tag{33}$$

Similarly for each $\bar{S}_i$. For any remaining internal node β, set its (diagonal) coefficient tensor as:

$$w^{\beta,\ell[\beta]}_{\ell[\beta_1]} = \begin{cases} 1, & \text{if } \ell[\beta] = 1 \\ 0, & \text{otherwise} \end{cases}. \tag{34}$$

Under the above specification, we have

$$\mathcal{W} \propto \sum_{j_1=1}^{t_1} \cdots \sum_{j_m=1}^{t_m} \underbrace{\left[\otimes_{i=1}^{m} \mathbf{v}^{S_i}_{j_i}\right]}_{\mathbf{a}_{j_1,\ldots,j_m}} \bigotimes \underbrace{\left[\otimes_{i=1}^{m} \mathbf{v}^{\bar{S}_i}_{j_i}\right]}_{\mathbf{b}_{j_1,\ldots,j_m}}. \tag{35}$$

Since $\{\mathbf{a}_{j_1,\ldots,j_m}\}$ and $\{\mathbf{b}_{j_1,\ldots,j_m}\}$ are linearly independent, respectively, through matricization we know $\operatorname{rank}(\mathcal{W}) \geq \prod_i t_i$. This shows that there exist coefficient tensors w so that $\operatorname{rank}(\mathcal{W}) \geq \prod_i t_i$.

To extend our conclusion from a special realization above to the generic case, let us note that the determinant of any submatrix of any matricization of $\mathcal{W}$ is a polynomial function of the coefficient tensors w. We have shown above this polynomial is not always zero, but then it follows immediately that the zero set of this polynomial has measure zero [4], i.e., for a generic realization of the coefficient tensors, we have $\operatorname{rank}(\mathcal{W}) \geq \prod_i t_i$. □

We constructed an explicit homogeneous mixture in the above proof whose rank is exponential. A similar construction, in the discrete setting, is given below [18]:

Example C.2. *Let $x_i \in \{0,1\} \forall i$, $k = 2$ and $d = 2m$. Choose for all $\forall i$ the basis (unnormalized) densities $f_{i,1}(x_i) = 1(x_i = 1)$ and $f_{i,2}(x_i) = 1(x_i = 0)$ (wrt some non-degenerate counting measure on $\{0,1\}$). Consider the (unnormalized) multivariate density F on $\{0,1\}^d$ (wrt the product counting measure):*

$$F(\mathbf{x}) = \begin{cases} 1, & \text{if } x_i = x_{i+m} \text{ for all } 1 \leq i \leq m \\ 0, & \text{otherwise} \end{cases}. \tag{36}$$

Clearly, F is a density mixture with input nodes from $\mathcal{F}$ and the associated tensor $\mathcal{W}$ satisfies $\operatorname{rank}_+(\mathcal{W}) > 1$. Hence, a standard shallow SPN needs at least $\mathsf{nnz}(\mathcal{W}) = 2^{d/2}$ product nodes to represent F, with input nodes from $\mathcal{F}$. The density mixture F is the so-called EQUAL *function in [18], whose Theorem 24 follows immediately from our Theorem 4.2 since $\mathsf{nnz}(\mathcal{W}) \geq \operatorname{rank}(W)$ for any matricization W of $\mathcal{W}$.*

We note that F is also a density mixture with input nodes from $\mathcal{G}$ (elements of which are themselves mixtures of $f_{i,1}$ and $f_{i,2}$), and $\operatorname{rank}_+(\mathcal{W}) \geq \operatorname{rank}(W) = \mathsf{nnz}(\mathcal{W}) = 2^{d/2} \geq \operatorname{rank}_+(\mathcal{W})$. Thus, any standard shallow SPN with input nodes from $\mathcal{G}$ still requires $2^{d/2}$ product nodes to represent the EQUAL *function. In other words, an SPN with an input layer from $\mathcal{F}$, a layer of sum nodes, a layer of product nodes, and a single sum node as output, would still require $2^{d/2}$ product nodes in order to represent the* EQUAL *function. This distinction between input nodes from $\mathcal{F}$ and input nodes from $\mathcal{G}$, to our best knowledge, has not been noted before.*

Theorem 5.1. *Fix $\epsilon > 0$ and tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$. Then, for some (small) constant $c > 0$,*

$$\epsilon\text{-rank}^{\infty}(\mathcal{W}) \leq \tfrac{c\|\mathcal{W}\|_{\mathrm{tr}}}{\epsilon^2}, \tag{5}$$

where $\|\mathcal{W}\|_{\mathrm{tr}}$ is the tensor trace norm. A similar result holds for $\epsilon\text{-rank}^{\infty}_{+}(\mathcal{W})$. The dependence on ϵ is tight up to a log factor.

Proof. Note that the ℓ_∞ norm is dominated by the ℓ_2 norm, so $\epsilon\text{-rank}^2 \geq \epsilon\text{-rank}^{\infty}$. Thus, given $\mathcal{W}$, we consider the approximation problem:

$$\min_{\|\mathcal{Z}\|_{\mathrm{tr}} \leq \|\mathcal{W}\|_{\mathrm{tr}}} \|\mathcal{Z} - \mathcal{W}\|_2^2. \tag{37}$$

Obviously, the minimum is 0. Moreover, if we run the generalized conditional gradient of [12] with initialization $\mathcal{Z}_0 = \mathbf{0}$, then after t iterations, we have

$$\|\mathcal{Z}_t - \mathcal{W}\|_2^2 \leq \frac{c\|\mathcal{W}\|_{\mathrm{tr}}}{t}, \ \operatorname{rank}(\mathcal{Z}_t) \leq t, \tag{38}$$

where c is some small universal constant. Here we are exploiting the property that each iteration of the conditional gradient algorithm only increments the rank by at most 1. Setting $c\|\mathcal{W}\|_{\mathrm{tr}}/t = \epsilon^2$ gives us

$$\|\mathcal{Z}_t - \mathcal{W}\|_\infty \leq \|\mathcal{Z}_t - \mathcal{W}\|_2 \leq \epsilon, \ \operatorname{rank}(\mathcal{Z}_t) \leq \frac{c\|\mathcal{W}\|_{\mathrm{tr}}}{\epsilon^2}, \tag{39}$$

whence follows $\epsilon\text{-rank}^{\infty}(\mathcal{W}) \leq O(\|\mathcal{W}\|_{\mathrm{tr}}/\epsilon^2)$.

The proof for the nonnegative rank is completely similar. □

The inverse-square dependence on ϵ in Theorem 5.1 is almost tight, as shown below:

Theorem C.3 ([1, Theorem 2.1]). *Let W be a $k^{d/2} \times k^{d/2}$ matrix with $|w_{i,i}| = 1\ \forall i$ and $|w_{i,j}| \leq \epsilon\ \forall i \neq j$, where $k^{-d/4} \leq \epsilon \leq \frac{1}{2}$. Then, for some absolute positive constant c,*

$$\operatorname{rank}(W) \geq \frac{cd \log k}{\epsilon^2 \log(\frac{1}{\epsilon})} \tag{40}$$

The above theorem, through un-matricization, clearly implies that there exist tensors $\mathcal{W}$ with $\epsilon\text{-rank}^{\infty}(\mathcal{W})$ lower bounded by $\frac{cd \log k}{\epsilon^2 \log(\frac{1}{\epsilon})}$, when ϵ is not too small.

A few remarks with regard to Theorem 5.1 are in order. We note first that our proof actually gives the same upper bound for the epsilon-rank under any ℓ_p norm where $p \in [2, \infty]$. Using the norm inequality $\|\mathcal{W}\|_1 \leq k^{d/2}\|\mathcal{W}\|_2$, we then immediately have from Theorem 5.1 that

$$\epsilon\text{-rank}^{1}(\mathcal{W}) \leq \frac{c\|\mathcal{W}\|_{\text{tr}} k^d}{\epsilon^2}. \tag{41}$$

Note however that there is still a factor of $k^{d/4}$ gap between the upper and lower bounds in Theorem D.2. It might be possible to optimize the lower bound in Theorem D.2 through different matricizations.

There are at least two issues with Theorem 5.1. First, if we use it to naively bound the L_1 norm difference of the underlying densities, i.e.,

$$\|g - h\|_1 = \int |\langle \mathcal{W} - \mathcal{Z}, \vec{f}_1(x_1) \otimes \cdots \otimes \vec{f}_d(x_d)\rangle| \mathrm{d}\mu(\mathbf{x}) \leq k^d \|\mathcal{W} - \mathcal{Z}\|_{\infty}, \tag{42}$$

the big constant k^d would pop out in the worse case. More disturbingly, in a practical application, we usually do not have access to $\mathcal{W}$ (which is too large to maintain directly anyways), hence it is not clear how to attain the bound in Theorem 5.1 algorithmically.

D ℓ_1 norm based ϵ-rank

In this section, we fix the norm $\|\cdot\|$ to be the usual ℓ_1 norm in definition (4), and we use the notation $\epsilon\text{-rank}^{1}_{+}$ or $\epsilon\text{-rank}^{1}$ to signify this convention. Our next result provides a new lower bound on the ϵ-rank, based on matricization.

Theorem D.1. *Fix $\epsilon > 0$ and let $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$. Then,*

$$\epsilon\text{-rank}^{1}_{+}(\mathcal{W}) \geq \epsilon\text{-rank}^{1}(\mathcal{W}) \geq \min\{i \geq 0 : \epsilon \geq \|W\|_{\text{tr}} - \sum_{j=1}^{i} \sigma_i(W)\}, \tag{43}$$

where W is any matricization of the tensor $\mathcal{W}$, $\|\cdot\|_{\text{tr}}$ is the matrix trace norm (i.e. sum of singular values), and $\sigma_i(W)$ denotes the i-th largest singular value of W.

Proof. Since the nonnegative rank is lower bounded by the rank, which is in turn lower bounded by the rank of any matricization, we clearly have

$$\epsilon\text{-rank}^{1}_{+}(\mathcal{W}) \geq \epsilon\text{-rank}^{1}(\mathcal{W}) \geq \epsilon\text{-rank}^{1}(W), \tag{44}$$

where W is an arbitrary matricization of $\mathcal{W}$ (note that matricization does not change the ℓ_1 norm). Moreover, for matrices, $\|\cdot\|_{\infty} \leq \|\cdot\|_{\text{sp}}$ (i.e., maximum singular value) hence $\|\cdot\|_1 \geq \|\cdot\|_{\text{tr}}$, thanks to duality. Therefore,

$$\epsilon\text{-rank}^{1}(W) = \min_{\|\Delta\|_1 \leq \epsilon} \operatorname{rank}(W + \Delta) \geq \min_{\|\Delta\|_{\text{tr}} \leq \epsilon} \operatorname{rank}(W + \Delta) = \min_{\|W - Z\|_{\text{tr}} \leq \epsilon} \operatorname{rank}(Z). \tag{45}$$

Using say [47, Theorem 1], we know that at the minimum we can choose Z to have the same singular vectors as W. It is clear then that Z should match the singular values of W, from the biggest to smallest, until the trace norm difference between Z and W falls under ϵ. □

The only prior work to Theorem D.1 that we are aware of is [18, Lemma 28], which only deals with the identity matrix $W = I$ and is loose by a factor of 2. [18] went on to construct a density function based on the EQUAL function (where $W = I$, c.f. Example C.2) that cannot be approximated by a polynomially-sized standard shallow SPN, up to some exponentially small ϵ. This existence result is then strengthened by [7], who showed that under the HR model, for almost every random tensor $\mathcal{W}$, there exists some ϵ (potentially depending on $\mathcal{W}$), such that any polynomially-sized standard shallow SPN cannot approximate $\mathcal{W}$ under ϵ. Based on Theorem D.1, we now present a complementary result.

Theorem D.2. *Fix any $\epsilon > 0$, and sample each entry of the tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ independently and identically from a standard Gaussian distribution, then with high probability,*

$$\epsilon\text{-rank}^1(\mathcal{W}) \geq O(k^{d/2} - \epsilon k^{d/4}). \tag{46}$$

Proof. Consider the reshaped matrix $W \in \mathbb{R}^{k^{d/2} \times k^{d/2}}$ of the tensor $\mathcal{W}$. Clearly, each entry of W is again an iid sample from the standard Gaussian distribution. As shown in [42], the smallest singular value of W is $\Theta(k^{-d/4})$, with high probability. Let $r = \epsilon\text{-rank}^1(\mathcal{W})$, then according to Theorem D.1, we have

$$\epsilon \geq \sum_{j=r+1}^{k^{d/2}} \sigma_i(W) \geq (k^{d/2} - r)k^{-d/4}. \tag{47}$$

Rearranging we obtain the claimed lower bound. □

The failure probability in Theorem D.2 depends on d only mildly: up to a small constant it approaches 0 at the rate $c^{k^{d/2}}$ for some constant $0 < c < 1$. Moreover, the standard Gaussian distribution can be replaced with any subgaussian distribution, or more generally any distribution with a bounded 4-th moment. To see the significance of Theorem D.2, let us note first that we can fix ϵ beforehand so there is no dependence on the tensor $\mathcal{W}$. Secondly, Theorem D.2 implies that with high probability, for any mixture density f, even if we contend with a constant approximation accuracy $\epsilon = O(1)$, a standard shallow SPN $\mathfrak{T}$ would still need $O(k^{d/2})$ many product nodes.

E More Related Works

The first attempts at rigorously analyzing the effect of depth in a network was in relation to the computational complexity of boolean circuits. A classical result is due to [43] who showed that for every integer I, there are boolean functions computed by a circuit with alternating *AND* and *OR* gates of polynomial size and depth I; but if the depth is reduced to $I - 1$, an exponential sized circuit would be required. A similar result was proven later by [37]. Another body of work in similar spirit was by [46; 30] proving that solving the k-bit parity task by a circuit of depth 2 requires exponentially many gates. A more recent result is due to [34] proving that bounded-depth boolean circuits cannot distinguish some non-uniform input distributions from the uniform distribution. This work by [34] solved a longstanding conjecture in the field.

Classical results for analyzing the expressiveness of neural networks involved results on universal approximation by [35; 31; 38], and by [32] who studied the networks VC dimension. Although, these early results provided general theoretical insights, they were restricted to shallow networks. Recently, several studies have been undertaken to understand the effect of depth on the expressive capacity of a deep network [41; 36; 45; 40; 39]. Most of these works provide separation results between the class of functions that can be efficiently represented by a deep network and those by shallow networks. However, one major limitation of these works is they consider pathological hand-coded network weights that exhibit these extremal properties by design. It is not evident if these class of networks and the hypothesis function class they encode resemble networks and functions used in practice. Therefore, fundamental questions about the expressive power of depth for neural networks used in practice is still not well understood.

Directly relevant to our contributions in this manuscript are recent works in analysing the effect of depths in *Arithmetic Circuits* [9], *Convolutional Arithmetic Circuits* [7] and particularly in *Sum-Product Networks* [24]. The first theoretical results for depth efficiency of SPNs was by [10]. They

constructed two families of functions - $\mathcal{F}$ which formed a full binary tree - and - $\mathcal{G}$ which consisted of n nodes in every layer with each node being connected to $n-1$ nodes in the previous layer - using an SPN with alternating *sum* and *product* layers. Their results establish that any n-dimensional function $f \in \mathcal{F}$ can be computed by an SPN of polynomial size but would require a shallow SPN $\Omega(2^{\sqrt{n}})$ hidden units to exactly represent f. They further show that for each $d \geq 4$ there exists a function $g_d \in \mathcal{G}$ that can be computed by an SPN of depth d and size $\mathcal{O}(nd)$ but would require $\Omega(n^d)$ sized shallow SPN to compute. However, this work has several limitations. Firstly, the separation results provided are restricted to depth 3 networks and networks in the family $\mathcal{G}$ and $\mathcal{F}$; it is not clear if a similar separation result holds for intermediate depths. Secondly, as the authors themselves state, it does not comment on any separation results when a deep SPN is only to be *approximated* by a shallow SPN. Thirdly, the specific families of functions $\mathcal{F}$ and $\mathcal{G}$ considered by [10] are not shown to be a relevant and universal hypothesis class that occurs in practice. Lastly, the SPNs considered in this work are not *valid* SPNs i.e. they do not encode a probability density function. Furthermore, the analysis is limited to only discrete variables with indicator leaf functions.

[18] extended the work in [10] by proving that there exist functions that can be efficiently computed by a depth d *valid* SPN but would require super-polynomially size for a depth $d-1$ SPN. In particular, they considered the EQUAL function on an array of Boolean variables $\mathbf{x} = (x_1, x_2, .., x_n)$ defined as follows : let $\mathcal{A} = \{1, 2, 3, .., n/2\}$ and $\mathcal{B} = (n/2+1, n/2+2, ..., n)$ be the index partition. Then, $\text{EQUAL} : \{0,1\}^n \to \{0,1\}$ where $\text{EQUAL}(\mathbf{x}) = 1$ when $\mathbf{x}_\mathcal{A} = \mathbf{x}_\mathcal{B}$ (i.e. the first half of the input is equal to the second half) and 0 otherwise. They proved that a *valid* shallow SPN would require $2^{\frac{n}{2}}$ units in the hidden layer to exactly represent $\text{EQUAL}(\mathbf{x})$ while an SPN of depth 4 would require $\mathcal{O}(n)$ size. Further, they also proved that a shallow SPN would still require $2^{n/2-2}$ nodes in the hidden layer to approximate $\text{EQUAL}(\mathbf{x})$[5]. However, [18] also restricted their analysis in the paper to only Boolean variables primarily because they used previous works from circuit complexity on arithmetic circuits to derive their results. Further, for the separation results, they constructed an example restricted to Boolean variables and indicator functions in the leaves; the proof does not generalize to *valid* SPNs with arbitrary density functions. Most importantly, they use very specific hand-crafted functions to prove separation results both for exact and approximate representation with no information on how frequently these functions occur in practice. Therefore, it might be the case that expect for a few hand-crafted pathological example, a shallow SPN can efficiently represent all functions derived from a deep SPN.

Recently, [7] proposed a deep network which they called *convolutional arithmetic circuits* that incorporates locality, sharing and pooling. They went on to show that this network corresponded to the Hierarchical Tucker Tensor decomposition [13]. Their main theoretical result showed that except for a negligible set of measure zero, all functions that can be represented by a deep convolutional arithmetic circuits of polynomial size, require an exponential sized shallow network to be realized exactly or approximated. The hypothesis class they considered was universal. However, a major limitation of their main result is that it is an existence result. That is, they say, for any deep convolutional arithmetic circuit, there exists an ϵ such that no shallow network of polynomial size can approximate the deep network within this ϵ distance. However, they do not provide any explicit relation with ϵ for the approximation. In other words, according to this analysis, this ϵ which requires an exponentially sized shallow network to approximate a deep network may be infinitesimally small. Therefore, a natural question to ask is : what is the ϵ-dependency of the size of a shallow network approximating a deep network within some ϵ distance?

F Detailed Experiments

We perform experiments on both synthetic and real world data to reinforce our theoretical findings. In appendix F.1, we present experiments on synthetic data to demonstrate the expressive power of an SPN and the algorithm proposed in (7)-(8) which we call SPN-CG. Next, we present two sets of experiments on real world datasets - in appendix F.2, we present results for image classification under missing data. In **??**, we compare the performance of SPN-CG to structure learning techniques for SPNs on seven real world datasets used previously as benchmarks.

[5] We direct the reader to [18] for further details on the exact definition of approximation used and the proof.

F.1 Synthetic data

In the first experiment, we generated 2000 samples from a 4 component and two dimensional GMM with full covariance matrices for each component. We estimate this GMM using SPN-CG. In each iteration, we add an additional SPN from $\mathsf{TMM}_{2,2}$ and learn its parameters and the mixing weights. We use SGD with weight decay to learn the parameters in each iteration. This experiment helps us to demonstrate that an SPN with univariate leaf distributions (thereby resulting in a mixture model with factored distributions) can estimate a GMM with full covariance matrix. Figure 14 shows the convergence of the model to the true negative log-likelihood on a held-out test set as a function of number of iterations.

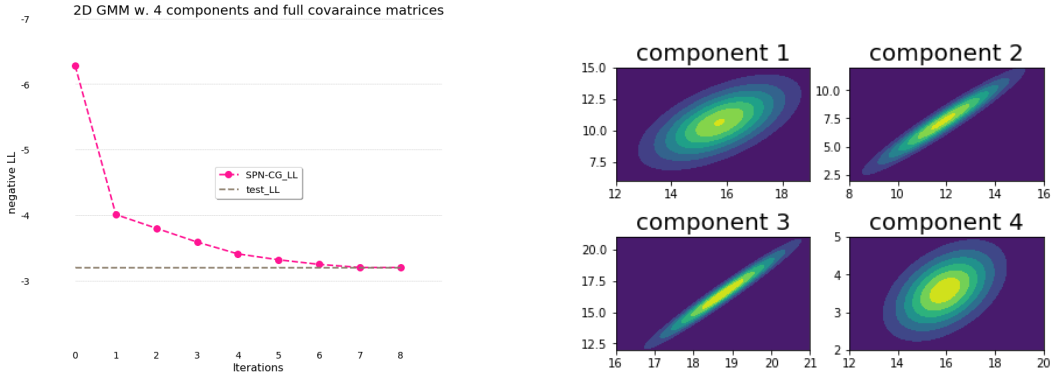


Figure 14: (Left) Convergence to true negative log-likelihood using SPN-CG (Right) Surface plots for covariance matrices of the components

F.2 Image Classification under Missing Data

In this section, we demonstrate the efficacy of generative models like deep mixture models learned using SPN-CG for image classification under missing data. We show that deep mixture models for which marginalization is tractable lend themselves naturally for problems under the missing data regime. We perform experiments on MNIST [15] for digit classification and small NORB [16] for 3D object recognition. We keep the same settings for the experiment as described in [26] i.e. we test on two settings of MAR missing distributions - (i) an i.i.d. mask with a fixed probability of missing each pixel, and (ii) a mask obtained by the union of rectangles of a certain size, each positioned uniformly at random in the image.

Our major aim with these experiments is to test our conditional gradient algorithm for high-dimensional real world settings. Therefore, we directly adapt the experiments as presented in [26]. Specifically, we adapt the code for the proposed HT-TMM for SPN-CG by following the details as given in [26]. In each iteration of our algorithm, we add an SPN structure exactly similar to HT-TMM. Therefore, the first iteration of our algorithm (i.e. SPN-CG1) amounts to a structure similar to HT-TMM while additional iterations increase the network capacity. For each iteration, we train the network by using AdamSGD variant of optimization for parameters with a base learning rate of 0.03 and $\beta_1 = \beta_2 = 0.9$. For each added network structure, we train the model for 22,000 iterations for MNIST and 40,000 for NORB.

We compare our results to the following methods :

1. **Data Imputation Methods** : Data imputation methods are a common technique to handle missing data using discriminative classifiers. The algorithm proceeds by completing missing values via some heuristic and passing the results to a classifier trained on uncorrupted data. In our approach, we use a ConvNet for prediction. We tested on the following data imputation methods in our manuscript :
    - Zero data imputation : completing all missing values with zeros.
    - Mean data imputation : completing all missing values with the mean value over the dataset

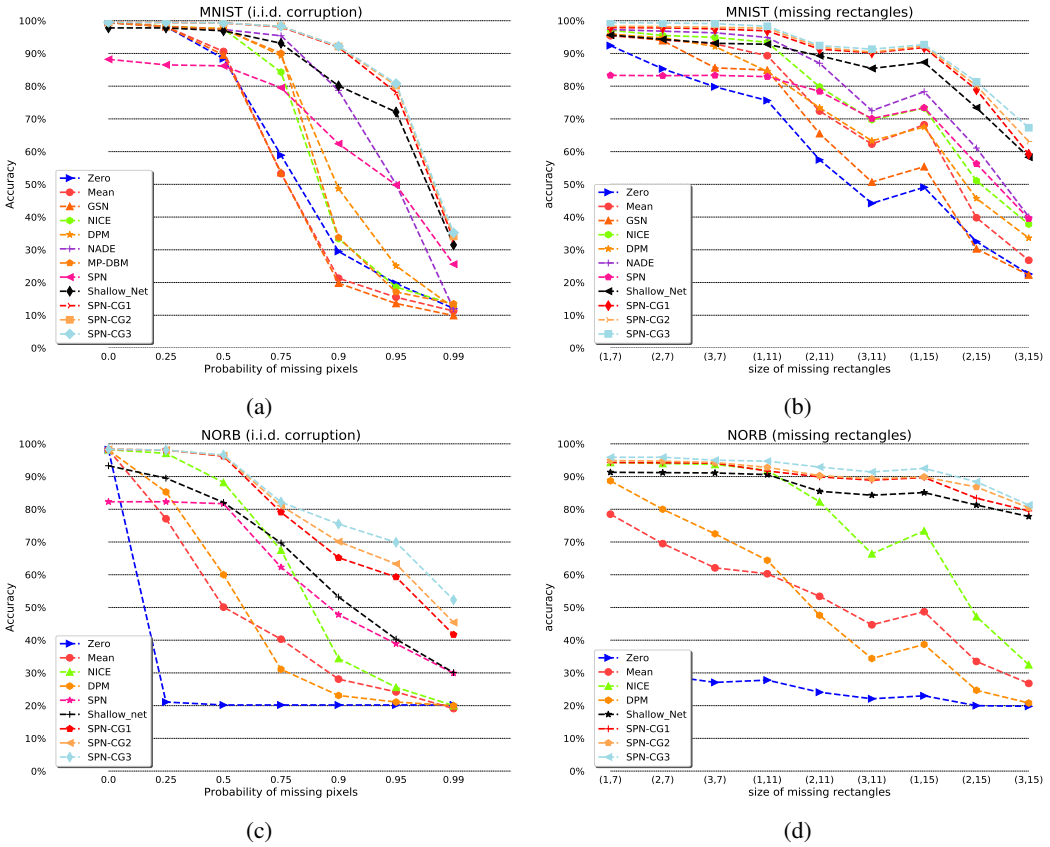


Figure 15: Performance of SPN-CG on missing data (a) MNIST data with i.i.d missing pixels (b) MNIST data with rectangles of missing pixels (c) NORB dataset with i.i.d. missing pixels (d) NORB dataset with rectangles of missing pixels

- Generative Stochastic Networks [33]
- Non-linear Independent Components Estimation (NICE) [11]
- Diffusion Probabilistic Models (DPM) [44]

2. **LearnSPN** [24]: We used the original code to learn the structure augmented with the class variable and learn the joint probability distribution using CCCP.

For all the algorithms except LearnSPN, we used the modified version of the code as suggested by [26]. Most of the code can be publicly accessed at : `https://github.com/HUJI-Deep/Generative-ConvACs`. We used the original code as suggested by the authors for LearnSPN. For our algorithm, due to time constraints, we could only perform three iterations for both NORB and MNIST dataset. We present the results for these three iterations denoted in the results as SPN-CG1, SPN-CG2 and, SPN-CG3 in this manuscript (see fig. 15a - fig. 15d). Our implementation for SPN-CG is available at : `https://git.uwaterloo.ca/l4mou/SPN`

The results show that SPN-CG performs well in the regime of missing data for both MNIST and NORB. Furthermore, other generative models including SPN with structure learning perform comparably only when a few pixels are missing but perform very poorly as compared to deep mixture models as larger amounts of data is missing suggesting that the structure of deep mixture models is advantageous. These experiments on MNIST and NORB help us conclude that deep mixture models learned using SPN-CG outperform other methods on image classification with missing pixels. Our results compliment the results in [26] where such experiments with state of the art results were presented.

Additional References

[30] Miklos Ajtai. $\sum_1^1$-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[31] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[32] Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

[33] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234, 2014.

[34] Mark Braverman. Poly-logarithmic independence fools bounded-depth boolean circuits. *Communications of the ACM*, 54(4):108–115, 2011.

[35] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[36] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

[37] Johan Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

[38] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[39] Holden Lee, Rong Ge, Tengyu Ma, Andrej Risteski, and Sanjeev Arora. On the ability of neural nets to express distributions. In *Conference on Learning Theory*, pages 1271–1296, 2017.

[40] James Martens, Arkadev Chattopadhya, Toni Pitassi, and Richard Zemel. On the representational efficiency of restricted boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2877–2885, 2013.

[41] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

[42] Mark Rudelson and Roman Vershynin. The least singular value of a random square matrix is $O(n^{-1/2})$. *C. R. Acad. Sci. Paris, Ser. I*, 345:893–896, 2008.

[43] Michael Sipser. Borel sets and circuit complexity. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 61–69. ACM, 1983.

[44] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

[45] Matus Telgarsky. Benefits of depth in neural networks. *COLT*, 2016.

[46] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 1–10. IEEE, 1985.

[47] Yao-Liang Yu and Dale Schuurmans. Rank/norm regularization with closed-form solutions: Application to subspace clustering. *arXiv preprint arXiv:1202.3772*, 2012.