Machine Learning Journal

Visualizing and understanding Sum-Product Networks

Antonio Vergari · Nicola Di Mauro · Floriana Esposito

August 2018

Abstract Sum-Product Networks (SPNs) are deep tractable probabilistic models by which several kinds of inference queries can be answered exactly and in a tractable time. They have been largely used as black box density estimators, assessed by comparing their likelihood scores on different tasks. In this paper we explore and exploit the inner representations learned by SPNs. By taking a closer look at the inner workings of SPNs, we aim to better understand what and how meaningful the representations they learn are, as in a classic Representation Learning framework. We firstly propose an interpretation of SPNs as Multi-Layer Perceptrons, we then devise several criteria to extract representations from SPNs and finally we empirically evaluate them in several (semi-)supervised tasks showing they are competitive against classical feature extractors like RBMs, DBNs and deep probabilistic autoencoders, like MADEs and VAEs.

1 Introduction

Density estimation is the unsupervised task of learning an estimator for a joint probability distribution over a set of random variables (RVs) from a set of samples. Such an estimator can be used to do *inference*—computing the probability of queries over those RVs. Many machine learning (ML) problems can be reframed as different kinds of probabilistic inference tasks, e.g., classifying a target RV can be solved by Most Probable Explanation (MPE) inference [20]. Density estimation can be thought as one of the most general tasks in ML.

Sum-Product Networks (SPNs) [31] are tractable density estimators compiling a joint probability distribution into a deep architecture. While for classical density estimators such as Probabilistic Graphical Models (PGMs) [20], like Markov Networks (MNs) and Bayesian Networks (BNs), performing exact inference is generally

Antonio Vergari
Dept. of Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen, Germany

Nicola Di Mauro · Floriana Esposito
Dept. of Computer Science, University of Bari, Bari, Italy
E-mail: antonio.vergari@tuebingen.mpg.de, nicola.dimauro@uniba.it, floriana.esposito@uniba.it

unfeasible—it is exponential in the model treewidth—for SPNs many kinds of queries like marginal and conditional probabilities are computable in linear time in the size of the network [31]. This is achievable by the presence of structural constraints like *decomposability* and *completeness* [31, 29], regarding the *scope* of the nodes—the RVs appearing in the distributions modeled by those nodes. SPNs have been successfully employed in several applications, such as computer vision [13, 27], speech recognition [28], natural language processing [4, 26] and activity recognition [2]. The task of learning an SPN has been tackled both in the weight [31, 33, 43] and structure learning scenarios [14, 34, 10, 32, 17].

Up to now, however, SPNs have been evaluated only as *black box* inference machines, i.e., only their output—the answer to a probabilistic query—is actually exploited in the task considered. In this paper we aim to *uncover the inner workings* of these probabilistic models by i) extending them towards Representation Learning (RL) [3] and ii) bridging themselves closer to deep neural models. By leveraging the learned inner representations of a model, RL approaches aim to disentangle and uncover different explanatory factors behind the data [3]. Usually, one employs these *embeddings* as features in predictive tasks later, e.g., Restricted Boltzmann Machines (RBMs) [37] have been employed as feature extractors after being unsupervisedly trained [6, 24], or representations from neural autoencoders—trained to reconstruct the data—are classically used as highly predictive features [39, 16].

In this work, we investigate how to extract and exploit the representations learned by SPNs when trained unsupervisedly as density estimators. Specifically, we try to answer the following questions: **Q1**) What are the inner representations learned by SPNs?; **Q2**) How can representations at different levels of abstraction be extracted from an SPN? How and why are RL approaches for classical deep neural models unsuitable for SPNs?; **Q3**) Are SPN representations competitive with those extracted from other neural models for predictive tasks?

To do so, we make the following contributions. First, we propose a natural interpretation of SPNs as sparse, labeled and generative Multi Layer Perceptrons (MLPs) that are arranged in a graph (**Q1**). Then, we try to better understand SPN representations by devising sampling routines in order to visually inspect their generated samples. Moreover, we visualize what each neuron has learned by providing a probabilistic formulation of visualizing samples maximizing the neuron activations in MLPs [11, 40] (**Q1**). Additionally, since extracting representations at different levels of abstraction in a layer-wise fashion—as usually done in MLPs [3]—is inadequate in SPNs, due to the aforementioned structural constraints, we devise several alternative criteria to build embeddings, like arranging nodes by type or scope length or aggregating them by scope (**Q2**). Finally, we demonstrate that the SPNs embeddings are competitive with other classical feature extractors such as RBMs, their deep counterparts Deep Belief Networks (DBNs) [35], and deep probabilistic autoencoders [15, 19] when evaluated on several (semi-)supervised tasks (**Q3**).

By answering the aforementioned questions we both provide practitioners with several routines to effectively exploit any learned SPN as a feature extractor, and suggest when and why to prefer one routine over another. At the same time, we hope to attract those in the deep learning community that are not familiar with such models by highlighting the differences—and advantages—of SPNs w.r.t. classical neural models for RL. Ultimately, we argue that SPNs are not only expressive and tractable probabilistic models but also provide rich part-based representations. They

naturally provide this without the need of being retrained and without requiring to manually specify an architecture beforehand or imposing an embedding size a priori, thus classifying as promising candidates for RL.

2 Related Work

The theoretical properties of SPNs have been thoroughly investigated, while their node interactions and practical interpretability have received little or no attention. For instance, [9] investigates the representational power of SPNs through a theoretical analysis that compares deep vs. shallow architectures. Martens and Medabalimi [25] demonstrate how expressive efficiency in SPNs correlates to their depth. In [34], SPNs are demonstrated to be estimators equivalent to Arithmetic Circuits over discrete finite domains. In [29], it was shown that consistency—a less strict constraint than decomposability—does not lead to exponentially more compact networks. In [42] it is demonstrated how SPNs are equivalent to bipartite BNs with Algebric Decision Diagrams modeling their conditional probability tables.

Visualizations provide important tools to assess a model from a qualitatively perspective, and have proven to be complementary to quantitative analysis. The most common and simplest technique for (not only deep) generative model is to visualize sampled instances [22, 15]. Recently, the need to better understand the successes of deep models more in depth lead to studies focused on particular architectures, for instance Convolutional Neural Networks in [41] and Recurrent Neural Networks, even more recently, in [18]. In this paper, we follow the work in [11] to visualize the feature learned by each neuron from an arbitrary layer as the input instance maximizing its activation. Extensions of [11] explored how to impose natural image priors to visualize features from deep models learned on image data: e.g., in [40]. In [36], on the other hand, the optimization problem is recast as finding the best image maximizing a class score and computing a saliency map for a query image sample, given a class. With MPE inference with SPNs we can efficiently solve an optimization problem similar to [11], effectively showing that the learned features are *part-based* representations.

Representation Learning (RL) [3] works have extensively studied how to extract useful features in unsupervised, semi-supervised and supervised settings from both deep and shallow models. RBMs have been extensively employed as robust feature extractors in several studies, both as generative and discriminative models [21, 24]. They also inspired successful autoregressive models like the Neural Autoregressive Distribution Estimator (NADE) [22]. For all these neural density estimators the structure is fixed a priori or after a hyperparameter selection for the number of hidden layers and hidden nodes per layer. With SPNs, efficient structure learning is possible. Moreover, the extracted representations can be assessed against the learned structure and *vice versa*, due to their recursive definition. Masked Autoencoder Distribution Estimators (MADEs) have been introduced in [15] as the autoencoder variant of NADEs. Empirically, they have been proven to be highly competitive in terms of likelihood scores while providing tractable complete evidence inference. The autoregressive property in MADEs binds inner neurons to be connected only to other neurons whose direct input respects the order dependencies among RVs, making them sparsely connected. Differently from SPNs, where each node outputs a probability, MADEs only do this in the last layer. Similarly to MADEs, variational

autoencoders (VAEs) [19] are generative autoencoders, but differently from MADEs they are tailored towards compressing and learning untangled representations of the data through a variational approach to Bayesian inference. While VAEs have recently gained momentum as generative models, their inference capabilities, contrary to SPNs, are limited and restricted to Monte Carlo estimates relying on the generated samples.

W.r.t. all the above mentioned neural models, one can learn one SPN structure from data and obtain a highly versatile probabilistic model capable of performing a wide variety of inference queries efficiently and at the same time providing very informative feature representations, as we will see in the following sections.

3 Sum-Product Networks

We denote RVs by upper-case letters, e.g., X, and ordered sets of RVs by their bold variants, e.g., $\mathbf{X}$. We denote a *sample* for $\mathbf{X}$ as $\mathbf{x} \sim \mathbf{X}$, and a single value from it as x_j. We define a set of m samples—a *dataset*—as $\{\mathbf{x}^i\}_{i=1}^m$. Let $\mathbf{Q} \subseteq \mathbf{X}$, then $\mathbf{x}_{|\mathbf{Q}}$ denotes the *marginal* sample $\mathbf{q} \sim \mathbf{Q}$, i.e., the restriction of $\mathbf{x}$ to $\mathbf{Q}$.

A *Sum-Product Network* (SPN) S over RVs $\mathbf{X}$ is a probabilistic model defined via a rooted directed acyclic graph (DAG). Let $\mathbf{S}$ be the set of all nodes in S and $\mathsf{ch}(n)$ denote the set of children of a node $n \in \mathbf{S}$. The DAG structure recursively defines a distribution S_n for each node $n \in \mathbf{S}$. To a *leaf* node n, i.e., $\mathsf{ch}(n) = \emptyset$, is associated a computationally tractable distribution $\phi_n \triangleq S_n$ over $\mathsf{sc}(n) \subseteq \mathbf{X}$, where $\mathsf{sc}(n)$ denotes the *scope* of n. [1]

An inner node n is either a *sum* or *product* node and its scope is recursively defined as $\mathsf{sc}(n) = \bigcup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$. A sum node n outputs a non-negative weighted sum over its children: $S_n = \sum_{c \in \mathsf{ch}(c)} w_{nc} S_c$. A product node n outputs a product over its children: $S_n = \prod_{c \in \mathsf{ch}(c)} S_c$. The *distribution* encoded by an SPN S is the normalized output of its root, and it depends both on the structure of S and its parameters—the set of sum-weights and the leaf distributions parameters—denoted as $\mathbf{w}$. Let $\mathbf{S}^{\oplus}$ (resp. $\mathbf{S}^{\otimes}$) be the set of all sum (resp. product) nodes in S. An example of SPNs is shown in Figure 1, where the direction of the model edges are graphically omitted to avoid clutter.

In order to allow for efficient inference, an SPN S is required to be *complete*, i.e., $\forall n \in \mathbf{S}^{\oplus}$, $\forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$, and *decomposable*, i.e., $\forall n \in \mathbf{S}^{\otimes}, \forall c_1, c_2 \in \mathsf{ch}(n), c_1 \neq c_2 : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$ [31, 29]. Moreover, we assume SPNs to be *locally normalized* [29]: $\forall n \in \mathbf{S}^{\oplus}, \sum_{c \in \mathsf{ch}(n)} w_{nc} = 1$. W.l.o.g., we also assume SPNs to have alternate node types, which we call *alternate* SPNs, i.e., each product (resp. sum) node can have as child a sum (resp. product) node [38].

Computing the *exact* probability of complete evidence $\mathbf{x} \sim \mathbf{X}$ consists of a single bottom-up evaluation of S: each leaf n evaluates $\phi_n(\mathbf{x}_{|\mathsf{sc}(n)})$ and subsequently, each inner node computes the probability $S_n(\mathbf{x}_{|\mathsf{sc}(n)})$—or short-hand $S_n(\mathbf{x})$—before passing it to its parent, till the root. This computation is guaranteed to be tractable as long as the network size—$|S|$, the number of edges in it—is polynomial in $|\mathbf{X}|$.

Even *exact marginal inference* can be computed in linear time w.r.t. $|S|$ in a complete and decomposable SPN S [31, 29]: to compute the query $p(\mathbf{Q} = \mathbf{q}), \mathbf{Q} \subset \mathbf{X}$,

[1] For discrete (resp. continuous) RVs, ϕ_n represents a probability mass function (resp. density function). We will generically refer to both as *probability distribution functions* (pdfs).

one evaluates ϕ_n for each leaf n by marginalizing RVs in $\mathsf{sc}(n)$ not in $\mathbf{Q}$, then propagating the outputs bottom-up, as before. Consequently, also exact conditionals are computable in linear time, since $p(\mathbf{Q}|\mathbf{E}) = p(\mathbf{Q}, \mathbf{E})/p(\mathbf{E})$, for $\mathbf{Q}, \mathbf{E} \subset \mathbf{X}$.

Exact MPE inference is hard in general SPNs [30, 7]. However, reasonable approximations for MPE solutions can be found in linear time in general SPNs [31, 30]. Given an SPN S over RVs $\mathbf{X}$, to find an MPE assignment $\mathbf{q}^* = \text{argmax}_{\mathbf{q} \sim \mathbf{Q}}\, p(\mathbf{E}, \mathbf{q})$ for some RVs $\mathbf{E}, \mathbf{Q} \subset \mathbf{X}, \mathbf{E} \cap \mathbf{Q} = \emptyset, \mathbf{E} \cup \mathbf{Q} = \mathbf{X}$, S is transformed into a Max-Product Network (MPN) M, by replacing each sum node n with a *max node* computing $\max_{c \in \mathsf{ch}(n)} w_{nc} M_c(\mathbf{x})$ and each leaf distribution ϕ_n with a maximizing distribution ϕ_n^M [30]. In a first bottom-up step, one computes $M(\mathbf{x}_{|\mathbf{E}})$. A top-down step traces back the MPE solution for RVs $\mathbf{Q}$. Starting from the root and following only the max output child of a max node and all the children of a product node, an *induced tree* is grown. Taking the argmax over its leaves retrieves the MPE solution [31].

The structure of SPNs can be effectively learned from data by leveraging the *probabilistic semantics* of sum nodes as *mixture models* over their child distributions and product nodes being *factorizations of independent components* [31, 29]. In particular, a categorical latent RV H_n, having values in $\{1, \ldots, |\mathsf{ch}(n)|\}$, can be associated to each sum node n. Network weights w_{nk} can be interpreted as the probabilities of choosing the k-th child branch from sum node n, having taken the path from the root up to n. Several constraint-based algorithms exploit this perspective and perform variants of hierarchical co-clustering [14, 34, 10, 38]. To introduce a decomposable product node, RVs are clustered by some statistical independence test, while complete sum nodes are introduced by clustering samples in sub-populations. The first learner adopting such a schema is LearnSPN [14], which greedily induces *tree-shaped* SPNs by recursively splitting the data matrix top down along its axis. For each call on a submatrix, column splits add child nodes to product nodes, while those on rows extend sum node. RVs are checked for independency by means of a G-test and a product node is inserted in the network if the test is passed with threshold ρ. A sum node n is inserted over k child nodes if a clustering step over the rows produced k different clusters. The weights w_{nc} are directly estimated as the proportions of samples falling into each cluster c. In this way, no weight learning step is needed after the network is fully grown. The learning process stops when the number of samples in a partition falls under a threshold μ. Then leaves are introduced as univariate distributions whose parameters are smoothed with a coefficient α. As they are considered to be independent one from another, a product node is put on top of them.

Here we adopt such a structure learning approach because i) it is simple and yet effective [14, 38]; ii) it does not require designing or fixing a priori a network structure; iii) it allows us to automatically determine the size of the representations we extract from SPNs in Section 7; iv) finally, by performing hierarchical co-clustering, LearnSPN acts as a *recursive data crawler*, providing the rich part-based representations we visualize in Section 6.

While latent RVs associated to sum nodes suggest a natural way to exploit SPNs as *generative* models, to the best of our knowledge, they have not been employed in the literature to sample. We use a simple sampling scheme for SPNs, effectively adopting it to visually inspect what a network has learned in Section 7. To generate one sample $\mathbf{x}$ from the pdf over $\mathbf{X}$ encoded by an SPN S, one traverses

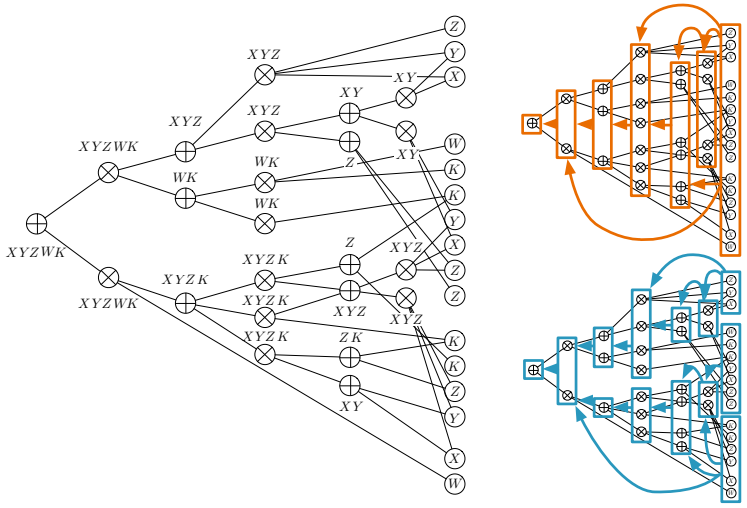


Fig. 1: *Layered representation of SPNs.* On the left, an example of a complete and decomposable SPN over RVs X, Y, Z, W. Leaves are represented as labelled circles and inner nodes have their scope associated. On the right, two possible layered representations featuring more (bottom) or less (top) sparse weight connections.

S top-down and induces a tree similarly to MPE inference: at each sum node n, the child c to follow is randomly chosen with probability proportional to w_{nc}. Product node children are followed all at once. To draw a sample $\mathbf{q}$ from the conditional distribution $p(\mathbf{Q}|\mathbf{e})$, one chooses the sum child branch c proportionally to $w_{nc}S_c(\mathbf{e})$, instead. Again, the leaves of the induced tree form a partition over all $\mathbf{X}$. A complete sample is generated by sampling from these leaf distributions.

4 Interpreting Sum-Product Networks as neural networks

Similarly to Arithmetic Circuits (ACs) [8, 34], SPNs are *computational graphs* in which the operations computed to evaluate a pdf are rearranged into an efficient data structure. Differently from ACs, the latent RVs semantics in SPNs allows for *direct* structure learning [5]. SPNs and ACs are *not* classical PGMs. Edges in SPNs deterministically determine the evaluation of the nodes in the DAG, which represent *computational units*, while in PGMs nodes represents RVs and edges the statistical dependencies among them.

As computational graphs, SPNs can be seen as feedforward deep neural networks (DNNs) in which hidden neurons can only compute sum and products and input neurons are pdfs. We argue that the peculiarity of SPNs as DNNs lies in them being a) *labelled*, b) *constrained* and c) retaining a *fully probabilistic* semantics. The scope function labels each network node by a set of RVs, enabling a *direct encoding* of the input space [3]. The DAG *constrained topology*, due to completeness and decomposability of scopes, determines *sparse* and *local* connections, similar to convolutional networks. Moreover, like RBMs, but differently from deep estimators like NADEs and MADEs [15], each neuron activation, i.e., $S_n(\mathbf{x})$, is still a valid

probability value by definition. These properties suggest each hidden neuron to act as *probabilistic part-based feature extractor*, which we investigate in Section 6.

We propose an interpretation of SPNs as sparse Multi Layer Perceptrons (MLPs) whose layers are arranged in a DAG. A classic *sequential* MLP consists of an input layer, a series of hidden layers and an output layer. A hidden layer of s neurons is a function of its input $\mathbf{x} \in \mathbb{R}^r$: $h(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, with σ being a nonlinear activation, e.g., ReLU [3], and $\mathbf{W} \in \mathbb{R}^{s \times r}$ a linear transformation with bias $\mathbf{b} \in \mathbb{R}^s$.

To reframe an SPN as an MLP one first has to *group nodes into layers* containing nodes of the same type. Each layer can receive input connections from multiple layers (including the input layer), and whose adjacent input and output layers are made up of nodes of a different type. Moreover, one layer can feed multiple layers with its output. These layers lend themselves to be arranged in a DAG based on their multiple input and output connections.

The input layer still computes the pdfs of the leaf distributions. The output of each hidden layer, based on its type, can be computed as follows. Let $\mathbf{S}(\mathbf{x}) \in \mathbb{R}^s$ denote the output of a generic SPN hidden layer with s nodes: $\mathbf{S}(\mathbf{x}) = \langle S_1(\mathbf{x}), \ldots, S_s(\mathbf{x}) \rangle$. A sum layer receiving r input nodes would output $\mathbf{S}(\mathbf{x}) = \log(\mathbf{W}\mathbf{x})$ where $\mathbf{W} \in \mathbb{R}_+^{s \times r}$ is the weight matrix defining the sparse connections: $\mathbf{W}_{(ij)} = w_{ij}$ if there is an edge between nodes i and j, and 0 otherwise. For locally normalized SPNs, we want $\mathbf{W} \cdot \mathbf{1}_r = \mathbf{1}_s$. A product layer, instead, would compute $\mathbf{S}(\mathbf{x}) = \exp(\mathbf{P}\mathbf{x})$, with $\mathbf{P} \in \{0, 1\}^{s \times r}$ being a sparse connection matrix: $\mathbf{P}_{(ij)} = 1$ if there is an edge between nodes i and j, 0 otherwise. In this reparameterization *exp* and *log* functions act as non-linear functions σ and the signals between layers switch from the domains of probabilities to log-probabilities and vice versa. The absence of a bias term $\mathbf{b}$ is due to dealing with normalized probabilities.

Grouping all nodes at a certain depth in a single layer leads to sequential DAGs with *very sparse weight matrices*. On the other hand, grouping only sibling nodes in a layer increases the number of layers in the DAG arrangement. In general, grouping nodes into layers in the DAG is somehow *arbitrary*: one can always break them up or merge them together to reduce or enhance sparsity on the matrices $\mathbf{W}$ and $\mathbf{P}$. In Figure 1 the same tree-shaped SPN is rearranged into a more sequential architecture. The advantages such a reparameterization offers are: i) better understanding SPNs as DNNs, highlighting the role of nonlinearities in SPNs; ii) allowing for efficient GPU implementations; [2] iii) paving the way to structure learning as constrained optimization—learning the sparse $\mathbf{P}$ and $\mathbf{W}$ indeed determines the DAG of S; and iv) questioning what are the representations learned from an SPN and how to extract them from it like for classic MLPs.

5 Extracting Representations from SPNs

A new feature representation for a set of samples—usually called *embedding* when it is continuous and dense—is a transformation of such a set to a new geometric space. The main aim of Representation Learning (RL) approaches is to extract *meaningful* feature representations, such that they can better explain the latent factors underlying the data or be effectively *reused* in other predictive tasks [3]. In

[2] As done in our code, available at `https://github.com/arranger1044/spyn-repr`.

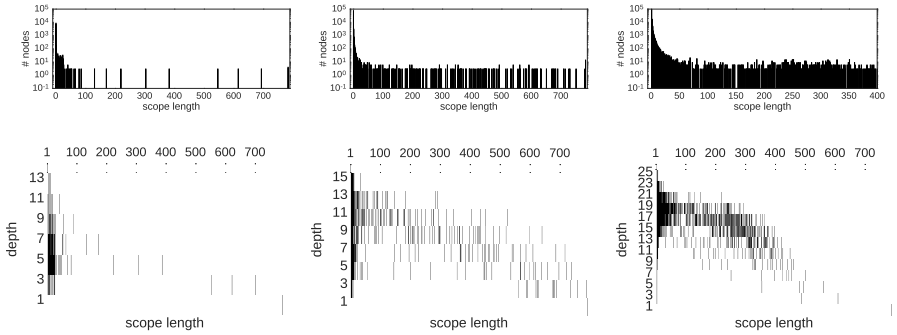
this section, we discuss how to employ SPNs for RL, following our interpretation of SPNs as peculiar DNNs, and how classical depth-based feature extraction criteria are unsatisfactory for SPNs. Furthermore, looking at SPNs under a RL lens can help better understanding them as probabilistic models, as well.

For deep architectures, it is common practice to employ the top hidden layer activations as the learned representations [3, 40]. The rationale behind this *layer-wise* extraction criterion is that such representations are arranged in a *hierarchy of abstractions* at different levels of granularity, correlated with the depth of a layer, with the top layers providing the most complex and meaningful features [11, 41, 40]. We are looking for an analogous and reasonable criterion to filter node activations in SPNs. Unfortunately, employing the MLP reparameterization introduced in Section 4 does not guarantee that the layer-wise or depth-wise criteria would produce representations at different levels of abstraction in SPNs. We actually deem them inadequate, due to the peculiar constrained structure in SPNs.

As a first motivation, consider that the top layers in an SPN would comprise significant fewer nodes w.r.t. lower layers. Second, the choice of any other layer in the DAG would be somehow *arbitrary*. Even the depth of a layer seems an unsatisfactory criterion, since nodes with very different scopes, hence encoding parts of the input space at very different granularities, may still share the same depth. To confirm these claims, we visualize the network topology of the SPN models employed in our experiments (see Section 7) w.r.t. the scope information associated to their nodes. Let S be an SPN over RVs $\mathbf{X}$. We define the *scope length* of a node $n \in \mathbf{S}$ as $|\mathsf{sc}(n)|$. The scope length of S is $|\mathbf{X}|$. We plot the scope lengths in Figure 2. A *long tail* effect is visible: 80% or more of the nodes in each model have a scope length of 1 to 3. Additionally, top layers indeed comprise very few nodes—as expected on tree-shaped SPNs as learned by LearnSPN-like algorithms. Furthermore, nodes at the same depth level can show a high variance of scope lengths. These visualizations support the inadequacy of extracting representations from SPNs by collecting activations by depth.

Therefore, we start investigate alternative criteria to extract embeddings from an SPN. The simplest would be by collecting *all* inner nodes outputs—the longest embedding for a given SPN (excluding the overabundant leaves). Nevertheless, even this heuristic is somehow still unsatisfactory: i) it treats all neurons equally, despite their different roles in the network, and ii) embeddings of such a size can easily suffer from the curse of dimensionality when employed as features in predictive tasks. Therefore, we propose several additional criteria *to filter nodes* from this full embedding, to better understand the influence of the network topology over the extracted representations and also to investigate an effective way to reduce the size of an embedding.

We first devise filtering activations by *node type* (i), to assess the role of sum versus product nodes as feature extractors. Then, we argue that a hierarchy of representations at different levels of abstractions for SPNs can be captured by how the scope information decomposes along the network structure. We therefore correlate the complexity of a representation learned by a node to its scope length, creating embeddings by filtering nodes having *comparable scope lengths* (ii). We further investigate how the scope information correlates to the level of abstraction of the representations by aggregating activations from nodes sharing the *same scope information* (iii) by leveraging the recursive definition of SPNs. Figure 3 depicts all the different embeddings we propose to extract from one SPN.



(a) REC (b) CON (c) BMN

Fig. 2: *Scope length distributions.* Scope length distributions for SPN-III models on REC (Figs. 2a), CON (Figs. 2b), BMN (Figs. 2c) (cf. Section 7 for model and dataset details). as a histogram of their possible values (from 1 to $|\mathbf{X}|$) against the number of nodes having that scope length and belonging to a certain depth. A long tail distribution for number of nodes w.r.t. scope lengths is visible (top). Very different scope lengths are grouped at the same layer depth (bottom, a bar indicates there is *at least* one node of the corresponding scope length at that depth).

Before proceeding with an empirical evaluation of *how* meaningful the proposed embedding extraction criteria are, we try to gain a deeper understanding of *what* the representations learned by SPNs are, by leveraging visualization techniques.

6 Visualizing SPN representations

To investigate the hypothesis of SPNs learning a hierarchy of part-based representations, we visualize the *representations learned by single neurons* in the network. We do this by exploiting both the direct encoding to the input space that the scope function provides, and the ability of SPNs to perform MPE inference efficiently (even though approximately).

For DNNs, one generally assumes the feature learned by the i-th neuron in the j-th layer to be approximated by the representation in the input space $\mathbf{x}^* \sim \mathbf{X}$ *maximizing its activation* h_i^j [11, 40]. To obtain such a representation, one can compute the bounded norm solution of the following non-convex problem:

$$\mathbf{x}^* = \underset{\mathbf{x}, ||\mathbf{x}||=\gamma}{\operatorname{argmax}} \, h_{ij}(\mathbf{x}; \boldsymbol{\Theta}), \tag{1}$$

solvable through stochastic gradient descent after fixing the network parameters $\boldsymbol{\Theta}$, even though it is only feasible for a limited number of layers and not guaranteed to converge [11].

SPNs lend themselves to an analogous problem formulation whose solution can be found without expensive iterative optimization. Since in an SPN S each inner node n recursively defines a probabilistic distribution over its scope $\mathsf{sc}(n)$, maximizing its activation reduces to find its MPE assignment over its scope—the

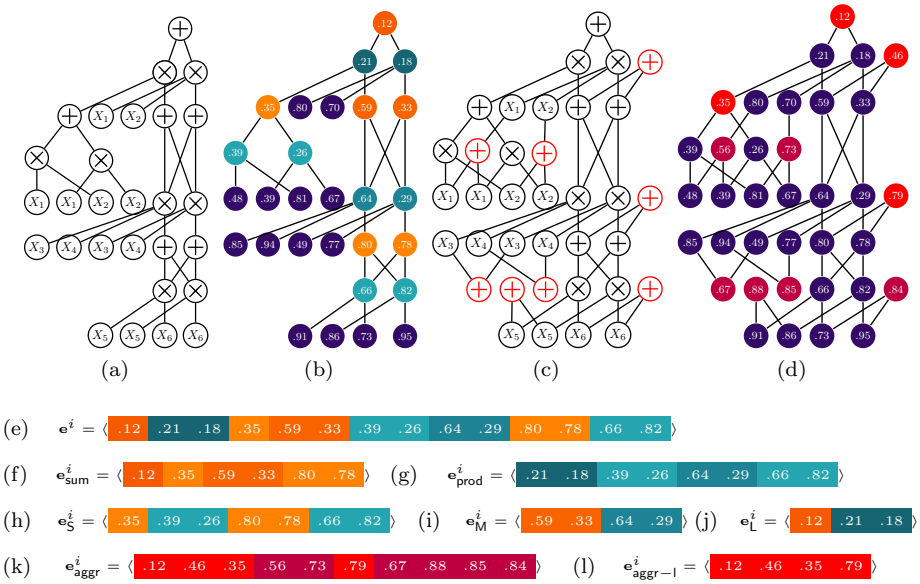


Fig. 3: *Extracting embeddings with SPNs.* By feeding the SPN S in (a) a sample $\mathbf{x}^i$, one evaluates it and collects its node activations (b). We devise several filtering criteria to build embeddings: collecting all inner node activations (e); filtering by node type, obtaining sum (f) and product only embeddings (g); filtering nodes by Small ($|\mathsf{sc}(n)| = 2$) (h), Medium ($|\mathsf{sc}(n)| = 4$) (i) and Large ($|\mathsf{sc}(n)| = 6$) (j) scope lengths; by aggregating all nodes by similar scopes (k) as represented by the red sum nodes introduced in (c) and evaluated in (d), or only inner nodes (l).

mode of the distribution S_n. Hence, one can reframe the problem in Equation 1 as:

$$\mathbf{x}^*_{|\mathsf{sc}(n)} = \operatorname*{argmax}_{\mathbf{x}} S_n(\mathbf{x}_{|\mathsf{sc}(n)}; \mathbf{w}). \tag{2}$$

This suggests that even the scope alone conveys semantics about the learned representations. Indeed, for image samples the visualization of the scope of each node corresponds to a shape against a background. The meaningfulness of such representations therefore correlates to the scope arrangement in the network.

To verify the validity of the scope length heuristics as a proxy for the abstraction level of a representation, we inspect the representations of nodes in Figure 4. There, visualizations for the representations learned have been selected by inspecting the scope length distributions of the considered model (see Figure 2), devising ranges for (S)mall, (M)edium and (L)arge scope lengths (see Section 7.3) for the largest SPN models we learned in Section 7. From there, we randomly extracted 9 neurons for each scope length range. [3]

[3] The randomness of the selection is visible in Figure 4 for the mid level representations for CAL—two nodes not only share the same scope but the same most activating input image as well. This might happen if a sum node and one of its maximising children are both chosen. Clearly, this highly structured part-based hierarchy cannot be equally visualized on smaller networks, e.g. SPN-I model on CAL (see Section 7).

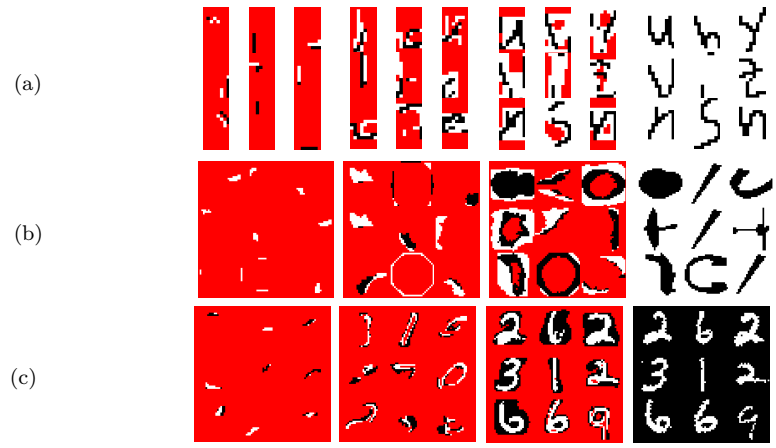


Fig. 4: *Representations visualization.* Visualization of representations of nodes sharing a similar scope length of increasing size ranges: small, medium and large (columns 1 to 3). For each scope range, representations are extracted from SPN-III models, according to Eq. 2, on OCR (a), CAL (b) and BMN (c) from 9 randomly selected nodes. For each node n, pixels corresponding to variables in $\mathsf{sc}(n)$ are colored black (resp. white) when their MPE assignment is 1 (resp. 0); pixels corresponding to variables not belonging to $\mathsf{sc}(n)$ are colored red, highlighting how recognizable *per se* are the shapes induced by scopes. Column 4 shows the training images nearest to those in column 3.

Even if spatial autocorrelation is not taken into account while learning the structure of the SPN considered (see Section 7), node scopes naturally capture recognizable part shapes. Clearly, this is not the case for higher-level features, i.e., features associated to scope lengths covering almost all the image.

The compositionality of the learned representations is evident through the different levels of the hierarchy: the longer the scope the higher the level of abstraction. The visualized features indeed resemble part-based filters at different levels of complexity: from pixel blobs to shape contours, to full shapes comprising background parts. We can therefore confirm the role of SPN nodes as probabilistic part-based filters. Lastly, by comparing the higher-level features extracted to the nearest training images one can inspect if they are their exact reconstructions. This is not the case for the features in Figure 4, even though they appear to be very specialized filters. We evaluate how this relates to the predictive performances of these representations in Section 7.

7 Representation Learning with SPNs

Here we empirically evaluate SPNs as feature extractors in a classical RL framework. We exploit embeddings extracted by different filtering criteria—leading to different feature sets—to train a classifier to predict unseen target RVs. We use the accuracy of such a classifier as a proxy measure to assess the *usefulness and effectiveness* of these representations [3]. We point out how we are not interested in achieving

Table 1: Datasets statistics: classes (c), dimensions (n) and samples number (m).

	c	n	m		c	n	m
REC	2	784 (28×28)	1000/200/50000	CAL	101	784 (28×28)	4100/2264/2307
CON	2	784 (28×28)	6666/1334/50000	BMN	10	784 (28×28)	50000/10000/10000
OCR	26	128 (16×8)	32152/10000/10000				

state-of-the-art accuracy scores on the dataset employed. Instead, our aim is to investigate how competitively the embeddings filtered by the proposed criteria rank when compared against themselves and the ones extracted by commonly employed *generative* models for RL like RBMs [37], DBNs [35], MADEs [15], and recently introduced VAEs [19].

7.1 Experimental setting

We employ five standard image classification benchmarks for evaluating DNNs on disentangling many factors of variations [23]: Rectangles (REC) [23], Convex (CON) [23], OCR Letters (OCR) [22], Caltech-101 Silhouettes (CAL) [24] and a binarized version of MNIST (BMN) [23]. We preprocessed them as in their original works and report their statistics in Table 1. Image samples are shown in Figure 5. Each dataset is a collection $\{\mathbf{x}^i \sim \mathbf{X}, y^i \sim Y\}_{i=1}^m$ comprising samples in the original feature space $\mathbf{X}$ and labeled by RV Y. We learn all our reference models on the $\mathbf{X}$ alone—in an *unsupervised* and *generative* way—discarding the class information Y.

As competitors, we employ RBMs as they have been extensively proven to be solid feature extractors [21, 24]; their deep version, DBNs [35], to measure the influence of latent RVs layered in a hierarchy as for SPNs; MADEs [15] as generative models that are also autoencoders—thus innately suited for RL—which, similarly to SPNs, exhibit a constrained and labeled structure, as imposed by the autoregressive property; and lastly VAEs [19] as autoencoders trained to be generative models. For a fair comparison, and to avoid numerical issues, we collect embeddings in the *log* domain for all models dealing with probabilities. We also want to learn networks with different *model capacities* in order to analyze how different structures, learned by the same algorithm, affect the usefulness of differently sized embeddings.

We learn RBMs having 500, 1000 and 5000 hidden units—providing embeddings of respective sizes—denoting them as RBM-5h, RBM-1k and RBM-5k. To generate embeddings, we evaluate the conditional probabilities of the hidden units given each sample. We train them using the Persistent Constrastive Divergence algorithm [24]. We select the learning hyperparameters by a grid search looking for the learning rate in $\{0.1, 0.01\}$, the batch size in $\{20, 100\}$ and the number of epochs in $\{10, 20, 30\}$ by comparing the best validation set pseudo-log-likelihoods.

For MADEs (resp. DBNs) we build architectures comprising 500 and 1000 hidden units up to 3 hidden layers, and we denote them as MADE-5h and MADE-1k (resp. DBN-5h and DBN-1k). For both DBN and MADE models, we extract embedding by concatenating all the activations of all the nodes from all the layers, obtaining embeddings of sizes 3000 and 4500, respectively.

Similarly, for VAEs we stack up to three levels in the encoder comprising 500 or 1000 units each, denoting them as VAE-5h and VAE-1k, but investigate different

compression factors $\{0.7, 0.8, 0.9\}$ for the bottleneck layer. Again, we experimented with extracting representations from the bottleneck layer alone or by concatenating all the layers of the encoder, ultimately finding the latter to provide far more accurate predictions.

For MADEs we employ the ADADELTA method to schedule learning rates with decay rate 0.95; we set to 30 the max number of worsening iterations on the validation and a batch size of 100. We initialize weights by SVD. Other hyperparameters are selected by a grid search guided by the best validation set log-likelihoods. We look for gradient dumping coefficients in $\{10^{-5}, 10^{-7}, 10^{-9}\}$; we either set no mask cycling, and we set their maximum number to 300, or we cycle over 32 random masks; we investigate both ReLus and softplus as nonlinearities.

We select the DBNs hyperparameters by performing a grid search for the learning rate in $\{0.1, 0.01\}$, the batch size in $\{20, 100\}$ and the epoch numbers in $\{10, 20, 30\}$. For VAEs we employed the ADAM method as an optimizer, running it up to 1000 epochs with a patience of 50, performing a grid search for batch size $\{20, 100, 256\}$ and learning rate in $\{0.01, 0.001\}$ and setting $\beta_1 = 0.9$, $\beta_2 = 0.999$ and no decay. We investigate both ReLus and softplus as nonlinearities.

Differently from RBMs, DBNs, MADEs and VAEs, we can directly learn the structure of our SPN models from data (see Section 3). However, we do not have a direct way to control embedding sizes except for regularizing the structure learning phase. We employ LearnSPN-b [38], a variant of LearnSPN, as a structure learner. With the aim of slowing down the greedy hierarchical co-clustering process, LearnSPN-b always splits samples and RVs into two clusters, thus achieving deeper and more compact structures [38]. For each dataset we learn three differently regularized architectures by early stopping, varying paramater $\mu \in \{500, 100, 50\}$, and denote them as SPN-I, SPN-II and SPN-III models respectively. For all models we fix the pairwise statistical independence test threshold ρ always to 20 except for OCR, for which it is 15. We then perform a grid search to select the best leaf distribution smoothing factor $\alpha \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$. Table 2 reports the learned SPN structural statistics.

At first, we peek at the effect of different model capacities over the representations learned by all models by employing them as generative models and *visually inspecting samples generated from them*. For SPNs we employ the sampling scheme we introduced in Section 3. We check if models have learned representations just able to reconstruct the training set [22, 15], by comparing samples against the nearest, in the sense of the Euclidean distance, training samples. Samples from our least regularized SPN, SPN-III, are compared against those from DBN-1k and MADE-1k models in Figure 5. The presence of noise is evident for all models and datasets, with REC and CON being the hardest datasets. DBN-1k generated images are generally more recognizable, however they are very close to their training counterparts. This might suggest a form of overfitting for DBN models. The proximity of the generated samples w.r.t. training images is even more prominent for VAE-1k models, but expected in this case, as they are trained to explicitly reconstruct their inputs. Additionally, we note how SPN-III struggles to capture some straight lines on REC, differently from DBN-1k, hinting at SPNs not modeling some spatial correlations in the data. The extent to which these conjectures will affect the predictive power of the extracted embeddings is investigated in Sections 7.2, 7.3, 7.4 and 7.5.

Table 2: Structural statistics for the SPN reference architectures on REC, CON, OCR, CAL and BMN datasets, like number of nodes by type (sum, product, leaf), of unique scopes and the number of nodes for certain scope lengths, since they correspond to the sizes of the embeddings as filtered in Section 7.3.

		μ	depth	edges	sum nodes	prod nodes	leaves	unique scopes	scope length S	scope length M	scope length L
REC	SPN-I	500	5	2240	5	10	2226	789	3	6	6
	SPN-II	100	15	8145	163	327	7656	946	108	354	28
	SPN-III	50	15	9424	265	531	8629	1045	231	537	28
CON	SPN-I	500	7	13019	13	33	12974	797	6	0	40
	SPN-II	100	15	50396	308	627	49462	1083	573	90	272
	SPN-III	50	17	81330	1872	3755	75704	2439	3849	1302	476
OCR	SPN-I	500	17	7848	64	163	7622	191	18	42	167
	SPN-II	100	23	35502	1972	4005	29526	1537	3465	2033	479
	SPN-III	50	23	48548	4069	8200	36280	2159	8844	2940	485
CAL	SPN-I	500	9	8102	10	22	8071	794	3	0	29
	SPN-II	100	17	32267	206	415	31647	987	387	63	171
	SPN-III	50	19	53121	1821	3645	47656	2340	3777	1434	255
BMN	SPN-I	500	19	47215	184	370	46662	967	99	33	422
	SPN-II	100	25	168424	5493	10990	151942	4487	10049	5034	1400
	SPN-III	50	27	198573	10472	6172	20948	6172	21992	8031	1397

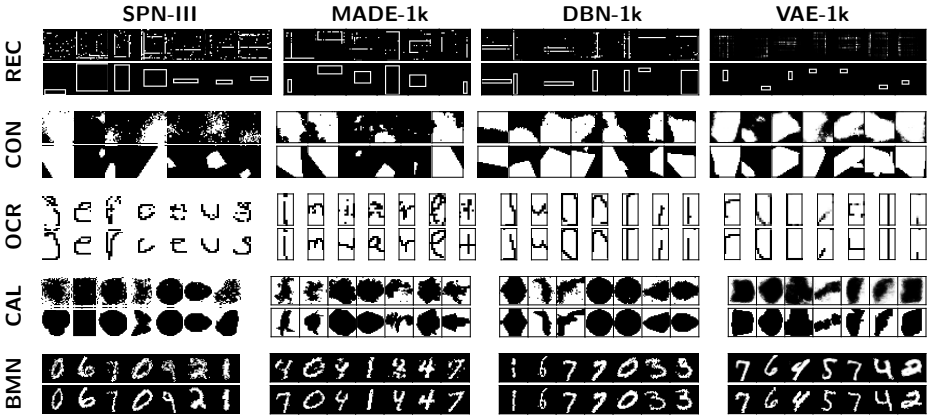


Fig. 5: *Sampling.* Seven samples from SPN-III (1st col), MADE-1k (2nd col), DBN-1k (3rd col), and VAE-1k (4th col) models on the first row, and their nearest neighbor images in the training set on the row below for REC, CON, OCR, CAL, and BMN.

7.2 Supervised representation learning with SPNs

Generally, we are interested in representing a sample $\mathbf{x}^i \sim \mathbf{X}$ as an *embedding* $\mathbf{e}^i$ in a new d-dimensional space $\mathbf{E}_\mathbf{X} \subseteq \mathbb{R}^d$ through a transformation f_θ provided by some model θ, i.e., $f_\theta(\mathbf{x}^i) = \mathbf{e}^i$. For an SPN S, f_S clearly is determined by the structure and parameters of S. Specifically, let $\mathbf{N} = \{n_j\}_{j=1}^d \subseteq \mathbf{S}$ be a set of nodes in S, filtered by a certain criterion. We build $\mathbf{e}^i$ by collecting the activations of

Table 3: Test set accuracy scores for full embeddings extracted with different SPN, RBM, DBN, MADE and VAE models, compared to the baseline LR model on all datasets. Bold values denote significantly better scores than all the others for a dataset.

		SPN			RBM			DBN		MADE		VAE	
data	LR	I	II	III	5h	1k	5k	5h	1k	5h	1k	5h	1k
REC	69.28	77.31	**97.77**	97.66	94.22	96.10	96.36	94.47	95.46	82.38	86.60	95.80	96.57
CON	53.48	67.48	78.31	**84.69**	67.55	75.37	79.15	81.12	81.61	71.96	76.90	79.45	78.89
OCR	75.58	82.60	**89.95**	**89.94**	86.07	87.96	88.76	87.48	88.21	84.40	84.18	86.38	86.89
CAL	62.67	59.17	65.19	66.62	67.36	68.88	67.71	69.53	**69.60**	61.94	64.76	64.62	64.67
BMN	90.62	95.15	**97.66**	97.59	96.09	96.80	97.47	97.06	97.51	94.10	95.18	96.98	97.10

nodes in $\mathbf{N}$, i.e., $e^i_j = S_{n_j}(\mathbf{x}^i)$. Therefore an embedding is the collection of the features expressed by a set of neurons, which could be potentially be visualized individually as showed in Section 6, by leveraging the fact that *they are indeed probabilities*.

In all experiments we train a *linear* classifier to predict Y from the embeddings extracted by our SPN and competitor models. The rationale here is to inspect if the new geometric space $\mathbf{E_X}$ has disentangled the input space enough to let a linear separator easily discriminate the classes in Y [3]. We employ a logistic regressor with an L2 regularizer in a one-versus-rest setting. We determine the L2 regularization coefficient C for each experiment in $\{0.0001, 0.001, 0.01, 0.1, 1.0\}$. As the simplest baseline, denoted as LR, we apply such a classifier directly on the original feature space $\mathbf{X}$. To test the statistical significance of the differences for the accuracies reported in Tables 3, 4, 5 and 6, we applied a paired signed rank Wilcoxon test, using a p-value of 0.05 for each possible pair of competitors. A result is rendered in bold if it is statistically better than all other non-bold results for a certain experimental setting. [4]

We are firstly interested in evaluating full embeddings comprising all nodes in a network, as the initial, simplest and uninformative attempt at measuring the predictive power of probability activations in SPN. Clearly, considering leaf node activations would result in too large embedding sizes (see Table 2 and Section 4). Therefore, as the first filtering criterion, we consider *full embeddings comprising only inner nodes* in S, i.e., $\mathbf{N} = \mathbf{S}^{\oplus} \cup \mathbf{S}^{\otimes}$. We devise a way to efficiently include the contribution of leaf nodes in Section 7.4. More generally, larger embedding sizes, even if it would definitely help a linear classifier better discriminate classes, could also let it suffer from the curse of dimensionality.

Test accuracy scores for all datasets, for LR, SPN, RBM, DBN, MADE and VAE models are reported in Table 3. Some datasets are inherently harder than others: if the LR baseline scores 90.6% of accuracy on the 10 classes of BMN—indicating the original feature representations to be disentangled enough—on the binary CON dataset it scores only 53.5%. All embeddings perform better than the LR baseline, proving their effectiveness in disentangling factors of variations, the only exceptions being SPN-I and MADE-5h models on CAL. However, while the latter embeddings comprise 500 values, the former only 32 (see Table 2), therefore acting

[4] Two or more results for a single experimental setting are in bold if they were not deemed to be statistically different, and all of them are significantly better than all the rest.

as a remarkable compressor for the original 784-dimensional space. Generally, there is a constant improvement by adopting less regularized SPN models, even if the accuracy difference between SPN-II and SPN-III models can be negligible at times.

On all datasets, with the exception of CAL, accuracies of SPN embeddings are better or competitive w.r.t. all other models. While additional hidden layers in DBNs make them slightly better than RBMs, the same is not true for MADE models, which underperform on almost all datasets. Remarkably, with the exception of REC, VAEs also underperform w.r.t. RBMs and DBNs.

It is remarkable how effective SPN embeddings are, considering the simple and greedy way in which both the network structures and parameters are unsupervisedly learned. The best accuracy on REC, 97.77% held by SPN-II is very close to the best score achieved by a *fully supervised* learner in [23]: 97.85% by an SVM. On CON, SPN-III scores a significantly higher accuracy than the best supervised model in [23]: 84.69% versus the 81.59% achieved by stacked autoencoders. This proves the practical utility of SPNs trained as generative models when plugged into predictive tasks: one obtains an expressive and tractable density estimator and, *at the same time and without retraining it*, can effectively extract effectively competitive features from it. It is worth asking if this performance gain is due to better modeling the data distributions. The answer is negative, since MADE log-likelihoods are higher that SPN ones. [5] We argue the effectiveness of SPN embeddings lies in the hierarchical part-based representations they provide, which are confirmed by the visual inspection of our models, as provided in Section 6. The positive effect of dealing with part-based representations in predictive tasks has been, indeed confirmed more than once in the literature, e.g. in [1, 12]. This, in turn, relates to how SPNs are learned by LearnSPN-like algorithms: while performing a form of hierarchical co-clustering over the data matrix, they implicitly discover meaningful ways to discriminate among data at different levels of granularity.

7.3 Filtering embeddings by node type and scope length

It worth looking for the nodes in an SPN most responsible for the surprisingly high accuracy scores obtained in Section 7.2. We do this as a means to reduce the size of SPN embeddings—as simply collecting all node activations is an unsatisfactory criterion easily suffering from the curse of dimensionality—and also to assess the importance of the representations at different levels of abstraction, confirming the scope length heuristics we propose. Note that selecting only a subset of nodes of the network as feature extractors is not the same as having a network composed only by those nodes—the contributions of the nodes filtered out are still present, even if indirectly, in the output activations of the collected nodes.

We apply the following filtering criteria to the inner node embeddings extracted previously. At first, we filter them by node type, to evaluate whether there is a pattern in *sum* ($\mathbf{N} = \mathbf{S}^{\oplus}$) versus *product node embeddings* ($\mathbf{N} = \mathbf{S}^{\otimes}$). Orthogonally, we filter nodes w.r.t. their *scope length* according to the heuristics about the hierarchy of abstractions as presented in Section 4. Based on the visualization on the scope length distributions provided there, we define (S)mall scope lengths,

[5] The trained RBM and DBN models do not allow to compute comparable log-likelihoods and comparing pseudo-log-likelihoods is not immediate.

Table 4: Test set accuracy scores for the embeddings filtered by node type (columns 2-7) and for SPN-III embeddings filtered by Small , Medium and Large scope lengths (columns 8-10). Bold values denote significantly better scores than all the others. ▲ indicates a better score than competitor embeddings with greater or equal size. ▽ indicates worse scores than competitor embeddings with smaller or equal size. The last columns report the accuracies for the SPN-III embeddings filtered by the scope length ranges S, M, L.

	SPN-I		SPN-II		SPN-III		SPN-III		
dataset	sum	prod	sum	prod	sum	prod	S	M	L
REC	72.46	62.25	**98.03**$^{\blacktriangle}$	97.06$^{\blacktriangle}$	**98.00**$^{\blacktriangle}$	97.04$^{\blacktriangle}$	88.73	**98.45**$^{\blacktriangle}$	93.91
CON	62.36	64.03	77.13$^{\blacktriangle}$	76.07$^{\blacktriangle}$	**83.59**$^{\blacktriangle}$	82.06$^{\blacktriangle}$	70.51$^{\triangledown}$	77.18	**83.32**$^{\blacktriangle}$
OCR	74.19	81.58	89.73$^{\blacktriangle}$	88.78$^{\blacktriangle}$	**90.02**$^{\blacktriangle}$	89.32	87.22$^{\triangledown}$	**89.29**$^{\blacktriangle}$	88.19$^{\blacktriangle}$
CAL	38.19	56.95	62.64	64.80	**66.58**$^{\triangledown}$	66.40$^{\triangledown}$	63.37$^{\triangledown}$	**66.23**$^{\triangledown}$	66.10
BMN	93.50	94.75	97.67	96.90$^{\triangledown}$	**97.80**	97.20$^{\triangledown}$	96.02$^{\triangledown}$	**97.42**$^{\triangledown}$	97.38

comprising 2 to 3 RVs; scopes of (M)edium length containing up to 100 RVs for all datasets, except for OCR where it is 50; and lastly, (L)arge length embeddings including all remaining lengths. We filter in this way only to embeddings from SPN-III models, as their scope length distributions have shown the highest variance.

Test accuracy results for the five filtering criteria are reported in Table 4. For SPNs with fewer nodes, the product nodes seem to contribute the most to the scored performance. On the other hand, when the model capacity is enough, e.g., with SPN-III models, *sum nodes act as efficient compressors*, greatly reducing the embedding size (cf. Table 2) and preserving the accuracy achieved by the full embeddings, or even improving it. This behavior is similar to what happens to max pooling in convolutional neural architectures, even though here we have *aggregations by weighted averages* as we are dealing with mixtures of valid probabilities. More generally, a *holistic effect* can be observed, sum and product nodes perform better together than when considered separately, even if slightly, and even when the size of a full embedding could suffer from the curse of dimensionality.

As reported in Table 4, embeddings from the smallest scope lengths are always the less accurate than both the full version and the ones filtered from longer scope lengths. Even if they are the embeddings with the largest size (cf. Table 2), the meaningfulness of the extracted features is minimal, as conjectured in Section 4. However, also *the contribution of the higher-level features is less prominent*. This confirms the intuition we had through the filter visualizations in Section 6: high level features in our reference models may be too specialized. As a result, in general, selecting only mid-level features proved itself to be a meaningful way to extract compressed, but still accurate, embeddings.

Filtered SPN embeddings are smaller than the RBM, DBN, MADE and VAE counterparts while their accuracies are comparable or better on three datasets out of five. The filtering process also improves the scores reported in Section 7.2 against fully-supervised models: e.g., the 97.80% accuracy on BMN achieved by the sum nodes of SPN-III, or the 98.45% scored by M scope length embeddings on REC.

7.4 Filtering embeddings by aggregating scopes

We now tackle embedding extraction by aggregating more node activations in a single feature. Again, we strive for shorter embeddings and, at the same time, we investigate if the leaves do play a negligible role as feature extractors. We propose to build embeddings by *averaging node outputs having the same scope*, leveraging the idea that all the nodes sharing the same scope are extracting different features for a single, shared, latent factor. Thus computing for each possible scope j in S: [6]

$$e_j^i = \frac{1}{|\{n|n \in S, \mathsf{sc}(n) = j\}|} \sum_{n \in \{n|n \in S, \mathsf{sc}(n)=j\}} S_n(\mathbf{x}^i). \tag{3}$$

The question concerning which nodes to consider for each aggregation can be answered in different ways. Constructing embeddings according to Eq. 3 requires collect the output of a *fictitious complete sum node* computing a uniform mixture over all nodes sharing the same scope (cf. Figure 3 (c)). Hence, we decide to aggregate only sum node outputs, since product nodes already contribute to their sum node parent feature extractions. As an alternative, we propose to aggregate always by scope both sum nodes and leaf nodes as well. In this way we can verify if the additional information they provide can be of some use (cf. Figure 3 (l)). Such a scope aggregation criterion is derived from the recursive definition of SPNs: as a sub-network rooted at a certain node is a valid probabilistic model over the RVs in that node scope, it is meaningful to look at the features extracted for each possible scope—or *resolution*—in the network.

As one can see in Table 5, leaf addition helps models with lower capacity like SPN-I, scoring the best accuracy for them on CAL. As the model capacity increases, however, the contribution of leaves becomes marginal or even zero. Generally, aggregated embeddings are comparably accurate w.r.t. the best corresponding sum embeddings, while being smaller. This empirically confirms the utility of scope aggregations as a heuristic to extract compact embeddings from an SPN.

As a general guideline, one would seek embeddings that are as compact and as informative as possible. We summarize the general findings from our extensive experimental suite. Sum node activations alone act as sufficient compressors for less regularized models, and as such they shall be preferred over products. Mid-level representations—embeddings belonging to nodes with medium scope lengths—are enough to preserve a good accuracy while reducing the embedding size. Contrary to classical deep models, only high-level representations are somehow slightly less informative, even if they provide the best compression. Scope aggregations prove to be a very effective size reduction heuristic, leveraging the recursive nature of SPNs as feature extractors, while deriving good predictive performances. Ultimately, our recommendation would be to first look at scope aggregations with SPNs, as they provide the best trade-off concerning embedding size and informative power.

7.5 Semi-supervised representation learning

Up to now, we empirically demonstrated the meaningfulness and effectiveness of SPNs representations when plugged into supervised tasks. In real world scenarios,

[6] $S_n(\mathbf{x}^i)$ values are in the exp domain and finally e_j^i is projected in the log domain.

Table 5: Test accuracies for the embeddings extracted by aggregating node outputs with the same scope, when leaves are not counted (no-leaves) and when they are considered (leaves). Bold values denote significantly better scores than all the others. ▲ indicates a better score than competitor embeddings with greater or equal size. ▽ indicates worse scores than competitor embeddings with smaller or equal size.

	SPN-I		SPN-II		SPN-III	
dataset	no-leaves	leaves	no-leaves	leaves	no-leaves	leaves
REC	72.47	75.92$^{\triangledown}$	**97.94**$^{\blacktriangle}$	**97.99**$^{\blacktriangle}$	**97.94**$^{\blacktriangle}$	**98.02**$^{\blacktriangle}$
CON	62.35	66.49$^{\triangledown}$	77.21$^{\blacktriangle}$	78.05	**83.52**$^{\blacktriangle}$	**83.84**$^{\blacktriangle}$
OCR	74.32	81.85	89.71$^{\blacktriangle}$	89.68$^{\blacktriangle}$	**89.90**$^{\blacktriangle}$	**89.91**$^{\blacktriangle}$
CAL	38.10	63.19$^{\triangledown}$	62.59	62.76$^{\triangledown}$	**66.49**$^{\triangledown}$	**66.58**$^{\triangledown}$
BMN	93.51	94.83$^{\triangledown}$	97.64$^{\blacktriangle}$	97.62$^{\blacktriangle}$	**97.80**	**97.80**

however, it is more likely that only a portion of the samples are labeled. Here we investigate whether in such a *semi-supervised learning* scenario these representations are still exploitable. Formally, we consider a set of samples $\{\mathbf{x}^i\}_{i=1}^m$ for which only a reduced set of $l < m$ labels $\{y^j\}_{j\in\mathcal{L}}$, $\mathcal{L} \subset \{i\}_{i=1}^m, |\mathcal{L}| = l$ is available. From our perspective on density estimation, nothing really changes—we will still exploit the same representation extracted on the RVs $\mathbf{X}$—hence reusing the same embeddings previously generated with our reference models.

We employ the *label spreading* algorithm [44] as the base classifier over all representations from all our models. In a nutshell, y^j from labeled samples are spread to the unlabeled samples that are *closer in the embedding space*. In particular, we adopt a k-nearest neighbor approach ($k = 7$) to classify samples. We set the clamping factor to 0.2 and used up to 30 iterates to let the label propagation process converge. The meaningfulness of the extracted representations in this scenario is still measured by the scored accuracy, however, it will now be more correlated to the ability of the new geometric space to facilitate label spreading by proximity.

To thoroughly evaluate the meaningfulness of all embedding spaces, we repeat the classification experiment by varying the number of available labels. As a common scheme over the datasets, we run a learning task allowing 1, 10, 60, 100 and 600 labeled training samples per class. In the end, we evaluate the following learning regime: 2, 20, 120, 200, 600 labeled samples for REC and CON; 26, 260, 1560, 2600, 7800 samples for OCR; 101, 505, 1010 samples for CAL; and 10, 100, 600, 1000, 3000 labeled samples for BMN. We repeat each experiment ten times.

We employ embeddings from the RBM-5k, DBN-1k, MADE-1k and VAE-1k models as competitors, as they achieved the highest accuracies in the supervised setting. We use embeddings from SPN-III models comprising sum nodes, large scope lengths or scope aggregations without leaves since they provide a good compromise between size and accuracy, as seen in Section 7.4. Lastly, as a baseline we run label spreading over the original feature space $\mathbf{X}$, denoting it as LP.

As reported in Table 6, SPNs embeddings provide a significant improvement over the baseline LP leveraging the original features, going from random guessing (50.73%) to 65.63% by only using 2 labeled samples on the REC dataset. Compared to all other embeddings, the SPN ones are very competitive, generally providing significantly better accuracy scores when labels are very scarce. In particular, they are competitive or statistically comparable to the second best ones—DBN-1k

Table 6: Mean and standard deviation (over ten runs) of test accuracies for semi-supervised learning experiments with label propagation on embeddings extracted from the sum nodes of SPN-III (sum), from its nodes with large scopes (L) or from scope aggregations (aggr) compared against the baseline LP and embeddings extracted from RBM-5k, DBN-1k, MADE-1k and VAE-1k models on all datasets and for a different number of available labels (l). Bold values denote significantly better scores than all the others for a dataset and a certain number of labeled examples.

	l	LP	SPN-III sum	SPN-III L	SPN-III aggr	RBM 5k	DBN 1k	MADE 1k	VAE 1k
REC	**2**	50.73±3.7	65.17±12.2	61.51±10.0	**65.53**±11.7	53.19±2.9	56.75±6.6	51.76±4.4	54.99±9.5
	20	63.18±8.1	79.16±4.5	**80.77**±3.9	78.73±4.3	66.87±5.6	77.92±4.6	58.78±2.8	65.27±4.4
	120	81.25±6.2	88.89±2.2	**90.97**±1.0	88.95±2.2	84.46±2.7	90.34±2.3	68.56±3.5	79.23±3.7
	200	84.95±5.5	90.38±1.2	**91.77**±1.7	90.51±1.2	86.90±5.4	**92.62**±1.1	70.97±2.3	83.20±3.1
	600	85.89±6.2	87.83±0.9	92.45±0.9	87.68±1.0	88.09±4.6	**93.36**±0.6	71.83±3.9	81.97±2.9
CON	**2**	**49.83**±0.5	48.66±1.5	49.11±1.0	48.63±1.4	49.58±0.4	48.97±1.1	49.26±0.7	49.43±1.5
	20	49.82±0.5	**51.10**±0.9	50.51±0.7	**51.11**±0.9	49.92±0.4	50.18±0.6	50.43±0.6	50.37±1.2
	120	49.82±0.3	**52.99**±1.1	52.09±0.7	**53.02**±1.2	49.91±0.3	50.29±0.4	51.16±0.6	50.88±0.9
	200	49.86±0.2	**53.54**±1.0	52.47±0.9	**53.55**±1.0	49.96±0.3	50.38±0.5	51.35±0.8	51.15±0.8
	600	49.84±0.2	**54.73**±0.5	53.58±0.6	**54.75**±0.5	50.03±0.2	50.25±0.2	51.92±0.5	52.30±0.9
OCR	**26**	29.51±4.1	**40.69**±3.4	**41.00**±3.6	**41.23**±3.0	38.74±4.4	37.70±3.5	20.69±3.3	36.08±3.4
	260	50.29±1.3	**65.45**±1.9	**66.61**±1.5	**66.06**±1.8	62.67±1.6	63.48±1.5	37.22±1.3	61.10±1.2
	1560	62.75±0.9	**76.29**±0.4	**76.97**±0.3	**76.27**±0.4	73.61±0.3	74.67±0.3	52.37±0.9	72.32±0.3
	2600	65.19±0.4	78.50±0.4	**79.24**±0.2	78.86±0.3	75.94±0.4	76.98±0.4	56.20±0.8	74.74±0.3
	7800	69.10±0.4	81.93±0.3	**82.48**±0.3	**82.15**±0.4	79.54±0.1	80.14±0.3	62.65±0.2	78.95±0.4
CAL	**101**	33.94±4.6	37.21±9.5	**39.71**±3.6	37.50±3.6	28.84±3.7	36.06±3.4	37.67±4.0	36.79±5.0
	505	50.19±0.6	51.43±1.0	52.09±0.8	51.40±1.0	44.48±1.1	51.69±0.7	52.39±0.7	**53.47**±1.4
	1010	51.46±0.7	54.56±0.4	54.69±0.6	54.38±0.5	46.58±0.9	53.06±0.7	53.76±0.7	**56.03**±0.6
BMN	**10**	47.43±8.6	**61.70**±9.5	58.76±9.1	60.43±8.4	59.46±10.8	**61.71**±9.7	48.22±10.2	54.30±10.6
	100	79.27±4.1	89.98±1.3	87.86±1.2	88.90±1.1	88.98±2.0	90.68±1.0	82.92±2.7	**91.04**±2.4
	600	90.31±0.6	94.36±0.2	93.37±0.3	93.65±0.3	94.04±0.4	94.45±0.1	89.68±0.8	**94.93**±0.4
	1000	91.15±0.5	94.72±0.2	93.96±0.3	94.14±0.3	94.32±0.3	94.81±0.2	90.44±0.6	**95.29**±0.3
	3000	92.29±0.2	95.21±0.1	94.73±0.1	94.75±0.1	94.86±0.1	95.11±0.1	91.57±0.3	**95.85**±0.1

representations. Only on BMN, does VAE-1k achieve a higher accuracy when enough labels are provided. Compared to the result in the supervised case, we can argue that the manifold learned by VAEs on BMN is potentially smoother, even if less separable class-wise. While this is somehow expected by VAEs, since they are trained to optimize a loss tailored towards such latent representations, the fact that the generatively learned SPNs generally achieve similar performance on BMN and even perform slightly better with on very few labels is quite remarkable.

The CON dataset has proved to be hard for all methods, with no improvements from random guessing, with the exception of SPN embeddings, which are the only one yielding a slight accuracy improvement to 54.75%. Concerning the filtering or aggregation criterion employed, all demonstrate the ability to create a meaningful geometric space in which distances implicitly favor classification, even if class information was unavailable when the density estimators were learned. How well same-class samples are reachable by proximity is visible in Figure 6. For instance, on REC the two red and blue classes are represented by two giant "well connected" components in the case of SPN-III, while for DBN-1k they are more fragmented, thus explaining why SPN embedded spaces require a fewer number of labeled examples to perform classification more accurately.

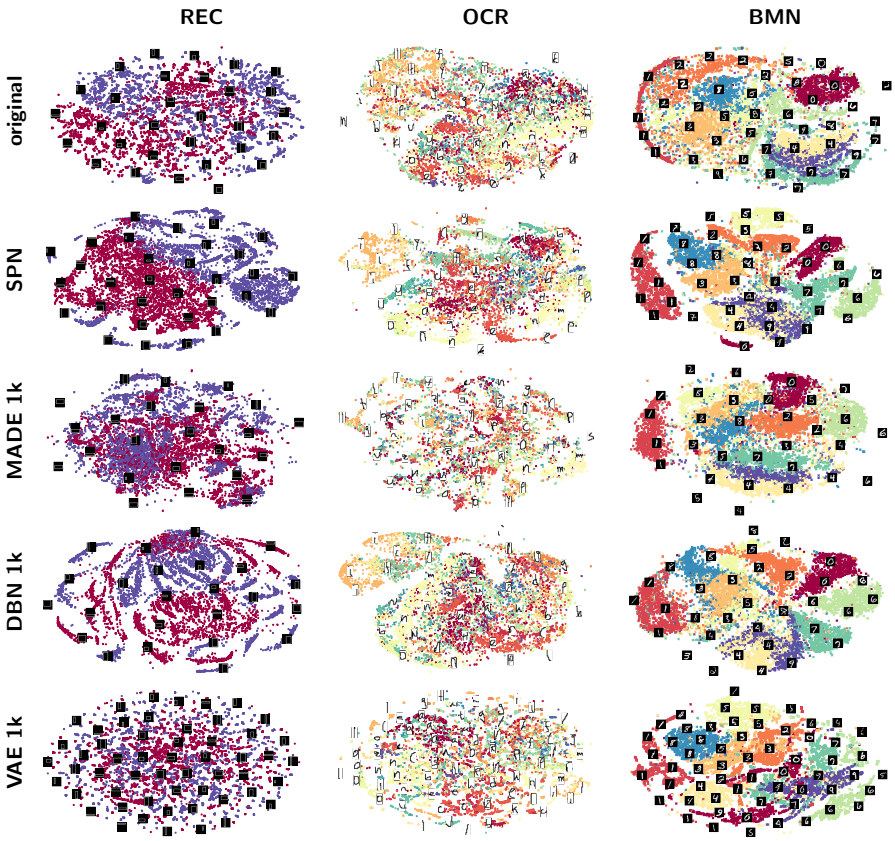


Fig. 6: *t-SNE plots.* The 2-d t-SNE plots of the SPN-III Large scope embeddings against those from MADE-1k, DBN-1k, VAE-1k and the original data for REC (left), CON (center) and BMN (right). Colors indicate samples from different classes. Miniatures represent image samples.

7.6 Experimental wrap-up

In this Section, through our extensive set of experiments on (semi-)supervised tasks we definitely confirmed the usefulness of SPNs as tools for RL—embeddings extracted from SPNs have been proven to be competitive against those from RBMs, DBNs, MADEs and VAEs. The main advantage these tractable probabilistic models provide, w.r.t. all other competitors, is that, at the end of the day, one can still exploit the same model to perform exact inference for a wide range of queries, along employing it to extract informative feature representations.

Concerning the reason of the effectiveness of these representations when employed in predictive tasks, again, one has to look at how SPNs are learned: structure learning as performing a form of hierarchical co-clustering (see Section 3). It is definitely interesting evaluating how different structure learning algorithms could

lead to different representations and how well these would perform in predictive tasks.

8 Conclusions

In this work we investigated how the internal representations learned by SPNs can be understood, extracted and exploited. We did that through visualization techniques exploiting the peculiarity of inference and structure in SPNs. We interpreted them as peculiar MLPs and extended their use to RL. For this purpose, we devised several embedding extraction schemes, after noting how classical layer or depth-wise criteria for SPNs are inadequate, evaluating their meaningfulness in a series of (semi-)supervised classification task. Concerning **Q1** and **Q2**, we confirmed the meaningfulness of a scope length heuristics to correlate a node feature abstraction level both visually and experimentally. We investigated the impact of the learned structure on network inference and on the learned representations. Sum embeddings have been demonstrated to provide the best size versus accuracy compromise, as well as scope aggregations. Concerning **Q3**, the embedding extracted from SPNs have been proven to be competitive against those from solid feature extractors such as RBMs, DBNs, MADEs and VAEs. All in all, we provided a better understanding of the inner workings of SPNs by uncovering what are their learned representations, and how to effectively exploit them. As a result, we also provided alternative ways—to the classical log-likelihood comparison—to assess the value of a learned SPN by visualizing and exploiting its inner representations.

References

1. Agarwal S, Awan A, Roth D (2004) Learning to detect objects in images via a sparse, part-based representation. IEEE TPAMI 26(11):1475–1490
2. Amer MR, Todorovic S (2015) Sum product networks for activity recognition. IEEE TPAMI 38(4):800–813
3. Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. IEEE TPAMI 35(8):1798–1828
4. Cheng W, Kok S, Pham HV, Chieu HL, Chai KMA (2014) Language modeling with Sum-Product Networks. In: ISCA, pp 2098–2102
5. Choi A, Darwiche A (2017) On relaxing determinism in arithmetic circuits. In: ICML, pp 825–833
6. Coates A, Lee H, Ng AY (2011) An analysis of single layer networks in unsupervised feature learning. In: AISTATS, pp 215–223
7. Conaty D, de Campos CP, Mauá DD (2017) Approximation complexity of maximum A posteriori inference in sum-product networks. In: UAI
8. Darwiche A (2003) A differential approach to inference in bayesian networks. JACM 50(3):280–305
9. Delalleau O, Bengio Y (2011) Shallow vs. deep sum-product networks. In: NIPS, pp 666–674
10. Dennis AW, Ventura D (2015) Greedy Structure Search for Sum-product Networks. In: IJCAI, pp 932–938

11. Erhan D, Bengio Y, Courville A, Vincent P (2009) Visualizing Higher-Layer Features of a Deep Network. Tech. rep., University of Montreal
12. Felzenszwalb PF, Girshick RB, McAllester D, Ramanan D (2010) Object detection with discriminatively trained part-based models. IEEE TPAMI 32(9):1627–1645
13. Gens R, Domingos P (2012) Discriminative Learning of Sum-Product Networks. In: NIPS, pp 3239–3247
14. Gens R, Domingos P (2013) Learning the Structure of Sum-Product Networks. In: ICML, pp 873–880
15. Germain M, Gregor K, Murray I, Larochelle H (2015) MADE: masked autoencoder for distribution estimation. In: ICML, pp 881–889
16. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
17. Hsu W, Kalra A, Poupart P (2017) Online structure learning for sum-product networks with gaussian leaves. CoRR abs/1701.05265
18. Karpathy A, Johnson J, Li F (2015) Visualizing and understanding recurrent networks. CoRR abs/1506.02078
19. Kingma DP, Welling M (2013) Auto-encoding variational bayes. In: ICLR, 2014
20. Koller D, Friedman N (2009) Probabilistic Graphical Models: Principles and Techniques. MIT Press
21. Larochelle H, Bengio Y (2008) Classification using discriminative restricted boltzmann machines. In: ICML, pp 536–543
22. Larochelle H, Murray I (2011) The Neural Autoregressive Distribution Estimator. In: AISTATS, pp 29–37
23. Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y (2007) An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. In: ICML, pp 473–480
24. Marlin BM, Swersky K, Chen B, Freitas ND (2010) Inductive Principles for Restricted Boltzmann Machine Learning. In: AISTATS, pp 509–516
25. Martens J, Medabalimi V (2014) On the Expressive Efficiency of Sum Product Networks. CoRR abs/1411.7717
26. Molina A, Natarajan S, Kersting K (2017) Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In: AAAI, pp 2357–2363
27. Peharz R, Geiger B, Pernkopf F (2013) Greedy Part-Wise Learning of Sum-Product Networks. In: ECML/PKDD, pp 612–627
28. Peharz R, Kapeller G, Mowlaee P, Pernkopf F (2014) Modeling speech with sum-product networks: Application to bandwidth extension. In: IEEE ICASSP, pp 3699–3703
29. Peharz R, Tschiatschek S, Pernkopf F, Domingos P (2015) On theoretical properties of sum-product networks. In: AISTATS
30. Peharz R, Gens R, Pernkopf F, Domingos PM (2016) On the latent variable interpretation in sum-product networks. CoRR abs/1601.06180
31. Poon H, Domingos P (2011) Sum-Product Networks: a New Deep Architecture. In: UAI, pp 337–346
32. Rahman T, Gogate V (2016) Merging strategies for sum-product networks: From trees to graphs. In: UAI

33. Rashwan A, Zhao H, Poupart P (2016) Online and distributed bayesian moment matching for parameter learning in sum-product networks. In: AISTATS
34. Rooshenas A, Lowd D (2014) Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In: ICML, pp 710–718
35. Salakhutdinov R, Hinton G (2009) Deep boltzmann machines. In: AISTATS, pp 448–455
36. Simonyan K, Vedaldi A, Zisserman A (2013) Deep inside convolutional networks: Visualising image classification models and saliency maps. CoRR abs/1312.6034
37. Smolensky P (1986) Information processing in dynamical systems: Foundations of harmony theory. Tech. rep., DTIC Document
38. Vergari A, Di Mauro N, Esposito F (2015) Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning. In: ECML/PKDD, pp 343–358
39. Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol PA (2010) Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR 11:3371–3408
40. Yosinski J, Clune J, Nguyen AM, Fuchs T, Lipson H (2015) Understanding neural networks through deep visualization. CoRR abs/1506.06579
41. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: ECCV, pp 818–833
42. Zhao H, Melibari M, Poupart P (2015) On the Relationship between Sum-Product Networks and Bayesian Networks. In: ICML, pp 116–124
43. Zhao H, Adel T, Gordon G, Amos B (2016) Collapsed variational inference for sum-product networks. In: ICML, pp 1310–1318
44. Zhou D, Bousquet O, Lal TN, Weston J, Schölkopf B (2004) Learning with local and global consistency. In: NIPS, pp 321–328