Visualizing and Understanding Sum-Product Networks

Antonio Vergari · Nicola Di Mauro · Floriana Esposito

Abstract Sum-Product Networks (SPNs) are recently introduced deep tractable probabilistic models by which several kinds of inference queries can be answered exactly and in a tractable time. The rising interest around them is due to their peculiar structural properties and their impressive performances in several application fields. Up to now, they have been largely used as *black box* density estimators, assessed only by comparing their likelihood scores on different probabilistic inference tasks. In this paper we *explore* and *exploit* the inner representations learned by SPNs, i.e. new representations for the input data that can be extracted from these deep architectures. We do this with a threefold aim: first we want to get a better understanding of the inner workings of SPNs, for instance, by looking at how inference and their learned representations are affected by the network structural properties; secondly, we seek additional ways to evaluate one SPN model and compare it against other probabilistic models, providing diagnostic tools to practitioners; lastly, we want to empirically evaluate how good and meaningful the extracted representations are, as in a classic Representation Learning framework. In order to do so we revise their interpretation as deep neural networks and we propose to exploit several visualization techniques on their node activations and network outputs under different types of inference queries. To investigate these models as feature extractors, we plug some SPNs, learned in a greedy unsupervised fashion on image datasets, in supervised classification learning tasks. We extract several *embedding* types from node activations by filtering nodes by their type, by their associated feature abstraction level and by their scope. In a thorough empirical comparison we prove them to be competitive against those generated from popular feature extractors as Restricted Boltzmann Machines. Finally, we investigate embeddings generated from random probabilistic marginal queries as means to compare other tractable probabilistic models on a common ground, extending our experiments to Mixtures of Trees.

Antonio Vergari · Nicola Di Mauro · Floriana Esposito
Department of Computer Science
University of Bari
Bari, Italy
E-mail: firstname.lastname@uniba.it

1 Introduction

Density estimation is the unsupervised task of learning an estimator for a joint probability distribution over a set of random variables (RVs) that are assumed to have generated a given set of observed samples. Once such an estimator is learned, one can use it to do *inference*, that is computing the probability of queries about certain states of all or a portion of the RVs considered. Many machine learning problems can be reframed as different kinds of inference tasks on a given probability distribution estimator; for instance, the classification of a target RV can be solved by Most Probable Explanation (MPE) inference [18]. If one were able to learn an estimator as a very good approximation of the underlying joint probability distribution, performing inference on such a model would lead to good approximate solutions in all tasks solvable by doing inference. From this perspective, learning a density estimator from data could be thought as one of the most general task in machine learning. Therefore, accurately estimating complex distributions from data and *efficiently* querying them are the required qualities for a good density estimator.

Sum-Product Networks (SPNs) [36] are recent tractable density estimators compiling a joint probability distribution into a deep architecture. For classical density estimators such as Probabilistic Graphical Models (PGMs) [18], e.g. Markov Networks (MNs) and Bayesian Networks (BNs), *exact* inference is exponential in the model treewidth and even approximate inference routines can result infeasible in practice [41]. On the other hand, exact complete evidence, marginal and conditional queries on an SPN are computable in linear time in the size of the network [36]. This is achievable if some easy-to-satisfy structural constraints like *decomposability* and *completeness* are guaranteed [36, 34]. The presence of such constraints, concerning the *scopes* of the network inner nodes, i.e. the RVs comparing in the distributions modeled by those nodes, differentiate SPNs from other kinds of probabilistic deep neural networks. SPNs have been successfully employed as density estimators in several applications ranging from computer vision [15, 30], to speech recognition [33, 39], natural language processing [7] activity recognition [3, 2] and fault localization [28]. The task of learning an SPN has been tackled both in the weight [36, 38, 53] and structure learning scenarios [16, 40, 13, 37], that is when a network structure is already available or has to be learned from data together with the model parameters. In both cases they have always been evaluated and directly compared to other probabilistic models on their likelihood scores only, even when they have been trained directly as discriminative models [15, 1]. All in all, SPNs have been learned and exploited as *black box* density estimators.

In this paper we investigate how to extract and exploit the *inner representations* learned by SPNs, i.e. the new data representations (features) that are automatically learned by an SPN as they are trained as density estimators in an unsupervised fashion. Jointly learning useful and exploitable representations for the input data while training a model for a certain task is the focus of Representation Learning (RL) [4, 5]. RL has been vastly approached in the literature for deep architectures in unsupervised, semisupervised and supervised settings [5]. Recent works heavily exploited visualization techniques on specific architectures like convolutional [52] or recurrent [17] models to investigate these model inner representations. However, SPNs differ from classical deep architectures in a number of ways and their peculiar network structure poses some challenges in determining which features to extract.

To the best of our knowledge, SPNs have never been employed as feature extractors, nor much attention has been given on leveraging their learned representations.

Our objective is threefold. In the first place we aim at **getting a better understanding of the inner workings of SPNs** by looking at their structural properties and at the features that can be extracted from these models (**O1**). Secondly, we want to **empirically evaluate the usefulness and meaningfulness of such representations** by plugging them in different supervised tasks (**O2**). Lastly, we want to **devise alternative ways to assess a learned SPN value** and to compare it to different probabilistic models, giving practitioners tools and procedures to evaluate and diagnose learned SPNs (**O3**). More concretely, throughout the paper we will try to answer the following research questions:

Q1 how are SPNs different from other neural architectures from a RL perspective? (**O1**)
Q2 how can representations at different levels of abstractions be extracted from an SPN? (**O1**, **O3**)
Q3 how does the network structure and structure learning impact the extracted representations? (**O1**, **O2 O3**)
Q4 are the features extracted from SPNs competitive against those extracted from other popular feature extractors? (**O2**)
Q5 how do SPN representations filtered by node type, scope and scope length perform? (**O2**)
Q6 how can the learned representations be evaluated against other tractable probabilistic models? (**O2**, **O3**)

We first review SPNs and revise their interpretation as feedforward neural networks (**Q1**), arguing for a scope-wise criterion to extract features with different complexities (**Q2**). To properly evaluate experimentally our claims we learn the structure of different SPNs as density estimators at three different model capacities on five benchmark image datasets (15 models total). To assess a network quality and investigate its structural properties we propose to exploit several visualization techniques on our reference models (**Q2**, **Q3**). In particular: evaluating them as generative models, we visualize their generated samples and we look into the features learned by each inner node. We note how this can be done by the means of MPE inference thanks to the *direct encoding* provided by node scopes. To get a deeper understanding of the node activations and roles, we visualize them as aggregates in the input space under complete evidence and marginal inference.

After that, we employ our reference models to extract features to be employed in a supervised classification task. The underlying assumption is to determine the value of an *embedding*, i.e. the representation of an input instance as a real valued vector, by the means of a proxy measure, i.e. the accuracy in predicting the values of a previously unseen target RV. In an extensive comparison we empirically demonstrate SPN embeddings to be quite competitive against those extracted from Restricted Boltzmann Machines (RBMs) [44]. RBMs are one of the most popular neural architectures for RL. They are highly expressive, yet *intractable*, density estimators, whose hidden unit outputs are used as feature extractors in several domains [19, 25] (**Q4**). Moreover, we propose to filter the embeddings extracted from SPNs by different criteria such as the associated node type, its scope and its *scope length* (i.e., the number of RVs in it). Not only we are looking at more compact representations, but also we are investigating each feature meaningfulness under

a RL lens (**Q1**, **Q4**, **Q5**). Lastly, we study the relevance of the representations encoded in our reference networks, learned by a structure learning algorithm, against the ones generated by evaluating several random marginal queries. We note how this approach can be applied to other tractable probabilistic models and we extend this evaluation to the Mixtures of Trees (MT) [27], proving the efficacy of SPNs as feature extractors in this setting as well (**Q3**, **Q6**).

The paper is organized as follows: in Section 2 we introduce and discuss SPN both as tractable density estimators and deep architectures; we describe our experimental setting and present our reference models in Section 4 and then we employ them to visualize the learned representations and to assess their structural properties in Section 5. Section 6 is about employing SPNs for RL. It contains the experimental evaluations of several kinds of embedding sets extracted from the reference models under different filtering criteria, and compared against RBM and MT models. A brief discussion on related approaches and state-of-the-art algorithms for SPNs is presented in Section 7. A wrap-up of all the derived conclusions is provided in the last Section (8).

1.1 Notation

In this paper, we are using the following notation. RVs are denoted as capital letters, e.g. X, Y, while (ordered) sets of RVs by bold capital letters, e.g. $\mathbf{X} = \{X_1, \ldots, X_n\}$, and their cardinality is expressed as $|\mathbf{X}| = n$. Complete evidence, also called a sample, configuration or instance, for $\mathbf{X}$, is denoted as $\mathbf{x}$, or more explicitly $\mathbf{x} \sim \mathbf{X}$; a single value from it is represented as x_j with the optional j index referring to the j-th RV, X_j. We define a set of observed samples (dataset) as the collection $\{\mathbf{x}^i\}_{i=1}^m$, using superscripts to denote each sample. Consider a subset of the RVs $\mathbf{Q} \subseteq \mathbf{X}$, we refer to sample $\mathbf{q} \sim \mathbf{Q}$ as the restriction of a sample on $\mathbf{X}$, denoted as $\mathbf{x}_{|\mathbf{Q}}$, to focus only on the values filtered by $\mathbf{Q}$. A probability distribution over $\mathbf{X}$ is denoted as $p_{\mathbf{X}}$ or simply p when this notation is not ambiguous. $p(\mathbf{Q})$ denotes computing the marginals over a set of RVs $\mathbf{Q} \subseteq \mathbf{X}$ while $p(\mathbf{x})$ denotes the evaluation of p corresponding to the single sample $\mathbf{x}$.

.

2 Sum-Product Networks

An SPN is a rooted weighted Directed Acyclic Graph (DAG) representing a computational graph encoding a distribution function over a set of RVs $\mathbf{X}$ in which the computational units output only weighted sums and products. Each sub-network rooted in a node of the graph is still an SPN encoding a restriction of the original distribution, defined over a subset of its RVs. An SPN can be more formally defined as follows.

Definition 1 (Sum-Product Network) A *Sum-Product Network* (SPN) S over RVs $\mathbf{X}$ is a rooted weighted DAG (network) consisting of distribution *leaves* (network inputs), *sum* and *product* nodes (inner nodes). Let $\mathsf{ch}(n)$ be the set of all the child (input) nodes of a node n and $\mathsf{pa}(n)$ the set of all its parent (output) nodes. A leaf n defines a tractable, possibly unnormalized, distribution ϕ_n over some RVs

in $\mathbf{X}$. A nonnegative weight w_{nc} is associated to each edge linking a sum node n to $c \in \mathsf{ch}(n)$. We indicate with $\mathbf{w}$ the set of all weights in the network S (network parameters). S_n denotes the sub-network rooted at node n and parametrized by $\mathbf{w}_n$, comprising all descendant nodes of n, recursively defined as the children of n and their descendants.

We now present the concept of *scope* of a node as the labeling function induced by the network structure that associates each node with a subset of the RVs comparing in the distribution encoded in the network.

Definition 2 (Scope) Let S be an SPN over RVs $\mathbf{X}$. The *scope* of a node n in S is denoted as $\mathsf{sc}(n) \subseteq \mathbf{X}$. The scope of a leaf node n is defined as the set of RVs over which ϕ_n is defined. The scope of an inner node n is defined as $\mathsf{sc}(n) = \bigcup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$. The scope of S is the scope of its root, i.e. $\mathbf{X}$.

The DAG structure in an SPN determines the evaluation order of the computational nodes in the network. Given a configuration for the inputs of the network, after all inner nodes have been evaluated, the root node outputs the evaluation of the encoded distribution at the given input configuration.

Definition 3 (SPN evaluation) Let S be an SPN over RVs $\mathbf{X}$ with parameters $\mathbf{w}$ and S_n be the sub-network rooted at node n. For each $\mathbf{x} \sim \mathbf{X}$, $S_n(\mathbf{x}_{|\mathsf{sc}(n)})$ and $S_n(\mathbf{x})$ are the same notation to indicate the output value of node n after $\mathbf{X} = \mathbf{x}$ is observed as the network input. Then, $S_n(\mathbf{x})$ can be computed as follows:

$$S_n(\mathbf{x}) = \begin{cases} \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)}), & \text{if } n \text{ is a leaf node} \\ \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x}) & \text{if } n \text{ is a sum node} \\ \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x}) & \text{if } n \text{ is a product node.} \end{cases} \tag{1}$$

The value of the whole network, i.e. $S(\mathbf{x})$, corresponds to the value of its root.

If a network S is *valid*, evaluating it corresponds to evaluate a joint unnormalized probability distribution $p_{\mathbf{X}}$. That is, $\forall \mathbf{x}, S(\mathbf{x})/Z = p(\mathbf{X} = \mathbf{x})$, where Z is the normalizing *partition* function defined as $Z = \sum_{\mathbf{x} \sim \mathbf{X}} S(\mathbf{x})$. This actually is possible since a complete and decomposable SPN correctly compiles the *extended network polynomial* encoding the distribution $p_{\mathbf{X}}$ [34]. To satisfy validity, it is sufficient for a network to be *complete* and *decomposable* [11, 36].

Definition 4 (Completeness, decomposability and validity) Let S be an SPN and let $\mathbf{S}_{\oplus}$ (resp. $\mathbf{S}_{\otimes}$) be the set of all sum (resp. product) nodes in S.

1. S is *complete* iff $\forall n \in \mathbf{S}_{\oplus}, \forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$.
2. S is *decomposable* iff $\forall n \in \mathbf{S}_{\otimes}, \forall c_1, c_2 \in \mathsf{ch}(n), c_1 \neq c_2 : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$.
3. If S is complete and decomposable, then it is *valid*.

If it holds that $\forall n \in \mathbf{S}_{\oplus}$, $\sum_{c \in \mathsf{ch}(n)} w_{nc} = 1$ and all leaves compute normalized distributions, then S will compute the exact, normalized distribution p, i.e. $\forall \mathbf{x}, S(\mathbf{x}) = p(\mathbf{X} = \mathbf{x})$. Such a network is called a *locally normalized* SPN. An unnormalized SPN can always be turned in a locally normalized SPN encoding the same distribution, in time linear to its size [34]. For the rest of the paper, when we will refer to SPNs we will assume them to be complete and decomposable, hence valid, and locally normalized, if not stated otherwise. Without loss of generality, we will also assume each network to have alternate inner node types, i.e. each product (resp. sum) node to be the network root or the child of a sum (resp. product) node.

2.1 Inference

Complete evidence inference consists of a single bottom-up (feedforward) evaluation of the network and proceeds according to Eq. 1. Therefore, it is guaranteed to be tractable as long as the network size is polynomial in $|\mathbf{X}|$.

Exact marginal inference can be computed with the same time complexity if the validity property holds [36]: the network can be evaluated in a similar fashion as for the complete evidence case. To compute a marginal query like $p(\mathbf{Q} = \mathbf{q}), \mathbf{Q} \subset \mathbf{X}$, one has to evaluate each leaf n as:

$$S_n(\mathbf{q}) = \begin{cases} p(\mathsf{sc}(n) = \mathbf{q}_{|\mathsf{sc}(n)}) & \text{if } \mathsf{sc}(n) \subseteq \mathbf{Q} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

and then propagate the outputs as before. This mechanism highlights the recursive probabilistic interpretation of SPNs, in that each sub-network modeling a probability distribution over its own scope, shall output 1 as the probability of marginalizing over all the RVs that scope. As a consequence, even conditional probability queries are also computable in tractable time, since $p(\mathbf{Q}|\mathbf{E}) = p(\mathbf{Q}, \mathbf{E})/p(\mathbf{E})$, for $\mathbf{Q}, \mathbf{E} \subset \mathbf{X}$. In addition to that, setting all leaf outputs to 1 equals to compute the *partition function* Z (whose value, for an unnormalized SPN, will differ from 1).

An approximation of MPE inference and the computation of an MPE assignment can be answered in linear time as well [29, 32]. Even if this does not correspond to the exact MPE solution as reported in [36], the algorithm proposed to compute it has been empirically proven to be a reasonable approximation [29]. We will now review the original algorithm proposed in [36] to find the MPE assignment for a set of RVs, as it will be the one employed for the filter visualizations in Section 5. Given an SPN S over $\mathbf{X}$, to find an MPE assignment:

$$\mathbf{q}^* = \operatorname*{argmax}_{\mathbf{q} \sim \mathbf{Q}} p(\mathbf{E}, \mathbf{q}) \tag{3}$$

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. M is built by substituting each $n \in \mathbf{S}_\oplus$ for a max node computing $\max_{c \in \mathsf{ch}(n)} w_{nc} M_n$ instead. Then M is evaluated bottom-up after setting all leaves n, $\mathsf{sc}(n) \subseteq \mathbf{Q}$, to output 1. A Viterbi-like top-down traversal traces back the MPE assignment for each RV in $\mathbf{Q}$. By starting from the root and following only the max output child branch of a max node and all the child branches of a product node [35], each instance determines a *tree* path whose leaves MPE assignments union forms the query answer [11].

2.2 Learning

A brief review of weight and structure learning algorithms is provided in Section 7. Here we focus on structure learning only and introduce a structure learning approach that is shared among many state-of-the-art algorithms. It also constitutes the base of the one that we employed to learn the reference models we will use later on. We are greatly interested in learning the structure of SPNs because, from a RL point of view, defining the nodes and their topology bears a greater representational bias towards the features that can be extracted than learning the weights only on a

fixed structure. Consequently, an empirical analysis of the extracted representations could potentially shed a light on the meaningfulness of structure learned by a particular algorithm. It could also offer a novel way to compare different structure learners. We will investigate this aspect in more detail in Section 6.4, where we will generate representations from random query evaluations of an SPN, comparing them with the representations built from recording all the structure nodes outputs.

Validity is not only the key for exact and tractable inference, but also an enabler for constraint based structure learning algorithms. The common factor of many of them is to apply a form of hierarchical co-clustering [16, 40, 13, 1, 47] exploiting the scope constraints of completeness and decomposability. To introduce a product node over children with disjoint scopes, RVs are split by some independence test, while sum nodes are created when samples over the same RVs are clustered by some similarity criterion.

The first learner to adopt such a schema is one of the most simple, yet effective, approaches still employed, LearnSPN [16]. LearnSPN learns *treed* SPNs, i.e. networks in which for each node n has only one parent node, i.e. $|\mathsf{pa}(n)| = 1$. By recursively partitioning the observed data matrix (samples $\times$ RVs), it grows the network top-down in a very greedy fashion. 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 variant of EM is employed to cluster rows together with λ as the penalty parameter governing the cluster number. A sum node n is inserted over k child nodes if the clustering 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 m. 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.

3 Sum-Product Networks Interpretations

LearnSPN never computes the likelihood of the model it is building directly, instead it keeps on adding sum and product nodes exploiting their *probabilistic semantics*. The natural probabilistic interpretation for sum nodes in a valid network is that of probabilistic mixtures over their children distributions whose coefficients are the children weights. Following from this, a categorical latent RV H_n, having values in $\{1, \ldots, |\mathsf{ch}(n)|\}$, can be associated to each sum node n. The network weights w_{nk} can also be interpreted as the probabilities $p(H_n = k|\pi(n))$, i.e. the chance of choosing the k-th child branch from node n, having already taken the path $\pi(n)$ from the root up to n. Analogously, since product nodes are evaluated as product of probability values, they identify factorizations over independent distributions [35, 34].

Similarly to *Arithmetic Circuits* (ACs) [11], to which they are equivalent for finite domains [40], SPNs are data structures efficiently representing the output of a *knowledge compilation process*. In such a process, the operations required to evaluate a probability distribution p are rearranged into the computational graph constituting the network structure. However, even if the class of distributions both SPNs and ACs encode is the same for discrete RVs with finite states, it's the latent RVs semantics in SPNs that allows for *direct* structure learning schemes, like the

one employed in LearnSPN. In such a structure learning algorithm, the compilation process is implicit.

Strictly speaking, SPNs, like ACs, are not PGMs, even if they are often called this way in the literature. In a classic PGM, edges represent a joint distribution decomposition into conditional independence relationships and nodes stand for RVs. On the other hand, in an SPN, edges deterministically determine the evaluation order of the network nodes, which, in turn, represent computational units performing computations. They are still probabilistic density estimators in that they can answer probabilistic inference queries over the joint distribution they encode by accepting different input configurations. They are generative models as well, since one can sample from the encoded distribution, a task not immediate for ACs if one ignored the SPN constructive equivalence proposed in [40].

3.1 SPNs as MLPs

SPNs can also be interpreted as a particular kind of feedforward deep Neural Networks (NNs) with nonnegative parameters $\mathbf{w}$, where the leaf distributions are input neurons whereas sum and product nodes are the hidden neurons, as noted in [30]. We argue that the peculiarity of SPNs as deep architectures lies in them being **labelled**, **constrained** and **fully probabilistic** NNs. They are labelled networks since the presence of the scope function associating RV sets to all nodes in the network. This peculiar feature is the key that enables a *direct encoding* of a neuron to a portion of the input space [5]. Based on this observation we will be able to visualize the neuron computations and their filter representation in the input space easily in Section 5. SPNs are constrained architectures in that the scope labels in a valid architecture are subjected to the completeness and decomposability constraints. From this it follows that a dense, i.e. fully connected feedforward architecture is hardly suitable for SPNs. Instead, scope constraint call for *sparsity* in their connections. Moreover, the decomposition of the scopes hints to each hidden neuron to be a **probabilistic part-based feature extractor**. We will investigate this hypothesis in Section 5.3 when we will try to visualize the representation associated to each inner node. Lastly, they are fully-probabilistic, because each neuron in an valid SPN still models a correct distribution over its scope. This differs from other neural density estimators like NADEs [22] and other autoregressive architectures, in which only the output neurons correctly compute probability values. This recursive structural property will help us formulate an experiment assessing the meaningfulness of the network topology as learned by a structure learner in Section 6.4. Even if explicitly stating these properties help in correctly classify SPNs as deep NNs, it provides little help in building them as NNs or in determining how to extract useful representations from them. We will now show how to reframe explicitly an SPN as a feedforward NN, showing how their structural peculiarities will impose novel representation extraction schemes and pose new questions.

In [35] SPNs are suggested to be a form of probabilistic Convolutional Neural Networks (CNNs) [23]. In such a CNN, product nodes would stand for the network filters with the role of average-pooling assigned to sum nodes and max-pooling to the max nodes in the corresponding MPN. However, this statement is somehow misleading, because, while the connections among the hidden neurons are indeed

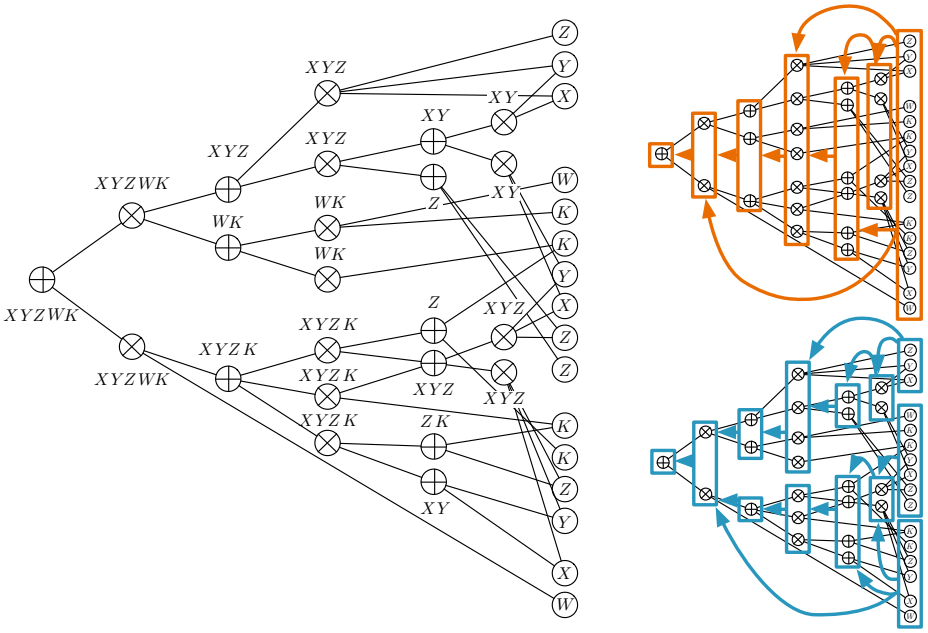


Fig. 1: An example of an SPN on the left on RVs X, Y, Z, W whose leaves are represented as labelled circles and whose inner nodes have their scope associated. On the right two possible layered representations.

sparse, they do not necessarily share a locality principle. Even tied weights cannot be immediately plugged into SPNs, differently from ACs. In addition to this, if SPNs were a probabilistic reparametrization of a CNN architecture, they would show some form of translation invariance. As we will show empirically in Section 5, this is not the case.

Instead, we propose a more immediate interpretation of SPNs as sparse probabilistic Multi Layer Perceptrons (MLPs) whose layers are rearranged in a DAG. A classic MLP consists of an input layer, a sequence of hidden layers and an output layer, each comprised by a certain number of neurons [6]. The output layer is usually determined with respect to the learning task, e.g. a linear classifier or a linear regressor are common choices [6, 5]. A usual MLP hidden layer of s neurons takes the form of a function of the input $\mathbf{x} \in \mathbb{R}^r$: $h(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ where σ is a nonlinear activation function (e.g. sigmoid, reLU), and $\mathbf{W} \in \mathbb{R}^{s \times r}$ is linear transformation with bias term $\mathbf{b}$.

To reframe an SPN as an MLP one first has to rearrange the nodes into layers containing nodes of the same type, and whose adjacent input and output layers are made up of nodes of different types. The input layer of the SPN would be the input layer of the MLP. For each other layer, based on its type, its output can be computed as follows. Let $\mathbf{S}(\mathbf{x}) \in \mathbb{R}^s$ denote the output function of a generic layer, i.e. $\mathbf{S}(\mathbf{x}) = \langle S_1(\mathbf{x}), \ldots, S_s(\mathbf{x}) \rangle$. If such a layer is a sum layer whose input layers comprise r nodes, its output can be computed as $\mathbf{S}(\mathbf{x}) = \log(\mathbf{W}\mathbf{x})$ where $\mathbf{W} \in \mathbb{R}_+^{s \times r}$

is the parameter matrix defining the sparse connections:

$$\mathbf{W}_{(ij)} = \begin{cases} w_{ij} & \text{if there is an edge between nodes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and $\mathbf{x} \in [0,1]^r$ representing the input probability values coming from the input nodes.

Similarly, if the layer were a product layer we would have that it 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 defined as:

$$\mathbf{P}_{(ij)} = \begin{cases} 1 & \text{if there is an edge between nodes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

and $\mathbf{x} \in \mathbb{R}^r$ representing the input probabilities in the *log* domain.

In this formulation the exponential and log functions act as nonlinearities and the input and output signals switch from the log to the exponential domain from one layer type to the other. One can assume the input layer output to be in the *log* (resp. exponential) domain if its adjacent layers are product (resp. sum) layers. Without loss of generality one can assume the last layer, comprising only the root node, to output always log probabilities.

The absence of a bias term $\mathbf{b}$ in the linear transformation is coherent with the common interpretation of probabilities: a mixture with all zero components sums to zero and a product of all 1 probabilities shall be 1.

With treed SPNs this representation leads to sequential architectures with very sparse weight matrices, in which each hidden layer receives connections from another hidden layer only and possibly from the input layer. More generally, the layers lend themselves to be arranged in a DAG structure based on their multiple input and output connections. Determining the number of layers in the DAG is somehow arbitrary, since choosing which node to assign a layer is only partially constrained by the node depth. Indeed, one can always break them up or merge them together to reduce or enhance the weight matrices $\mathbf{W}$ and $\mathbf{P}$ sparsity. In Figure 1 the same treed SPN is rearranged into a more sequential architecture first and as a less sparse DAG last for instance.

Such a representation offers a series of advantages. The first one is the ability to leverage efficient matrix computation and GPU computation libraries while computing $\mathbf{S}(\mathbf{x})$. Another one is the possibility to design structure learning algorithms as constrained optimization problems looking for $\mathbf{P}$ and $\mathbf{W}$ as those matrices encode the presence of an edge between two nodes for each nonzero entry.

Despite these algorithmic advantages, this kind of layered representation does not provide clues about which neurons to use as feature extractors in a representation learning framework. It neither clearly induces a hierarchy of representations at different levels of abstraction. For deep architecture it is common practice to reuse the top hidden layer (penultimate layer) outputs as the learned representations in other tasks on the same domain, while the ones coming from the lower layers can be employed in transfer learning schemes [5, 49]. The rationale behind this is that the learned representations are arranged in a hierarchy of different abstractions induced by the stacked layers, with the top layers extracting the most complex features. Visualizations of the learned representations as image filters have proven this conjecture empirically in several works [14, 52].

For the inadequacy of extracting representations layer-wise from an SPN we present several motivations. First of all, in a layered SPN, the top layer comprises a single node, the root, and the choice of any other single layer in the DAG of layers structure would be somehow arbitrary. Even the depth of a layer, defined as the length of the shortest path from the root to a node in the layer, seems an unsatisfactory criterion, since in the same layers there can be nodes computing distributions over very different scopes. Moreover, following the structural measures like the number of parameters as reported in [47] we believe that in SPNs built by LearnSPN-like algorithms the largest number of nodes is concentrated near the lower layers. We will investigate all these conjectures empirically on by inspecting our reference model structure topologies in Section 5.

Moved by these considerations, we start investigating how to extract representations from an SPN by collecting all nodes outputs. We then propose several filters on the nodes as alternative means to retrieve non layer-wise representations. We filter them by node type, and we aggregate them by scope. In addition to that, we suggest that a hierarchy of representations at different levels of abstractions for SPNs can be captured by exploiting the neurons as part-based feature extractors. Therefore, we correlate the complexity of a representation extracted by a node to the number of RVs in its scope, which we define as *scope length*.

Definition 5 (Scope length of a node) Let S be an SPN over RVs $\mathbf{X}$. The *scope length* of a node n in S is defined as $|\mathsf{sc}(n)|$. The scope length of S is $|\mathbf{X}|$.

As a practical argument, we note that if a scope determines a direct encoding to the input space, scopes of similar lengths would be associated to portions of the input space with similar sizes. At the same time, this scope length criterion also suggests the layer-wise representations to be not meaningful for SPNs: the nodes in same a layer could have different scope lengths, possibly representing both high and low level features.

In this Section we have started to tackle questions **Q1** and **Q2**. We reframed an SPN as a deep feedforward architecture made up by classic MLPs with a simple parametrization. However, since this architecture can be arranged in a DAG of layers quite arbitrary, the usual correlation between the depth of a layer and the abstraction level of the feature it extracts seems less motivated. We suggest to employ the scope information, e.g. the set of RVs associated to a node and its cardinality, as a criterion to derive features at different grains of abstractions. We will confirm these conjectures empirically in Section 5.3 by visualizing and comparing the representations at different scope lengths.

4 Experimental setup

In this section we define the experimental settings under which we learned the SPN models that we will employ as references for the visualizations and the feature extraction we will present in the following sections. In this phase, we learn both their structure and parameters in an unsupervised way, i.e. discarding the class information provided by each dataset.

Table 1: Structural statistics for the SPN reference architectures on REC, CON, OCR, CAL and BMN datasets.

		m	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

4.1 Datasets

We conduct our experiments on five standard benchmark image datasets: Rectangles (REC), Convex (CON), OCR Letters (OCR), Caltech-101 Silhouettes (CAL) and a binarized version of MNIST (BMN). REC has been created in [21] as a dataset of 28×28 binary pixel images representing wide and tall rectangle shapes; we reserved a portion of the training samples for validation, ending up with a 1000, 200 and 50000 samples splits for train, validation and test respectively. CON is made up by 28×28 binary pixel images depicting convex or concave full shapes [21]. Again we employ some training samples for validation: we split it into 6666, 1334 and 50000 samples for train, validation and test respectively. OCR comprises 16×8 binary pixels images of handwritten letters from 10 classes, split into 32152, 10000, 10000 for training, validation and testing as in [20]. CAL contains objects outlines represented in 28×28 binary pixel images and divided into 101 classes. They are split into 4100 examples for training set, 2264 for validation, and 2307 for test [25]. For BMN we used the splits provided in [21], consisting of 28×28 pixel images of handwritten digits (from 0 to 9), divided into 50000, 10000 and 10000 portions. Then we binarized them ourselves[1] by setting a pixel to 1 with probability proportional to its intensity as in [42]. Samples from all datasets are visible in Figure 4.

Even if these low resolution image datasets could be considered less hard for modern deep architectures, they still pose quite a few challenges for current SPN structure learning algorithms. First, the relatively high number of features and instances can likely fool the statistical independence tests employed during learning. Second, spatial ordering and autocorrelation among pixels are not taken into account in any way. Third, the high instance and feature numbers can potentially

[1] We didn't employ the one available in [22] since it lacks the class information.

make the learning step a bottleneck, or build networks with millions of nodes if the structure learner hyperparameters are not wisely chosen.

4.2 Reference models

For each dataset, we learn three SPN reference models on the $\mathbf{X}$ alone, i.e. leaving the Y class variable apart. Later in Section 6 we are employing it in the supervised phase where a classifier will be trained on the representations extracted from the SPN reference models to predict it. Our objective is to learn networks with different *model capacities* in order to analyze how different structures, learned by the same algorithm, influence inference and the learned representations. Model capacity with SPNs is related to their expressiveness and thus influenced by their size and depth [26]. This, in turn, will determine the maximum embedding size we can extract from a network, giving us a way to indirectly control it in our experiments against models like RBMs for which it can be directly chosen as the number of hidden neurons.

We employ LearnSPN-b [47], a variant of LearnSPN, as a structure learner. LearnSPN-b splits the observed sample matrix slices always into two while performing row clustering or checking for column independence. With the rationale of slowing down the greedy hierarchical clustering processes, it guides the structure search towards deeper and more compact networks without limiting their expressiveness nor reducing their accuracy as density estimators [47]. For each dataset we learn three different architectures by letting vary the early stopping parameter $m \in \{500, 100, 50\}$ and fixing the G-test threshold ρ to 20 for all datasets except for OCR, for which we set it to 15. While such a value for ρ prevents RVs to be declared independent too often (hence preventing SPNs to overgrow), the parameter m enables a form of early stopping, as noted in [47]. In such a way, we end up with three differently regularized architectures, each one comprising a different depth, number of nodes and so on. For the rest of the paper we will refer to them as SPN-I, SPN-II and SPN-III models, from the most regularized (for $m = 500$) to the least one (for $m = 50$). After an architecture is fully grown, we choose leaf distribution smoothing parameter value $\alpha \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$ with a grid search.

Table 1 reports structural statistics for the fifteen SPNs learned in this way. In addition to the ones presented in [47] like size and depth, we record the number of nodes divided by type (sum, product, leaf), the number of unique scopes and the number of nodes belonging to a certain scope length group, since they correspond to the sizes of the embeddings as they will be filtered in Section 6.2.

We provide all the code employed for the visualizations and experiments as openly available to reproduce them[2].

5 Visualizing Sum-Product Networks

Visualization techniques offer precious hints for understanding the representations learned by a deep architecture and their inner workings [14, 52, 17, 46]. We show

[2] `https://github.com/arranger1044/spyn-repr`

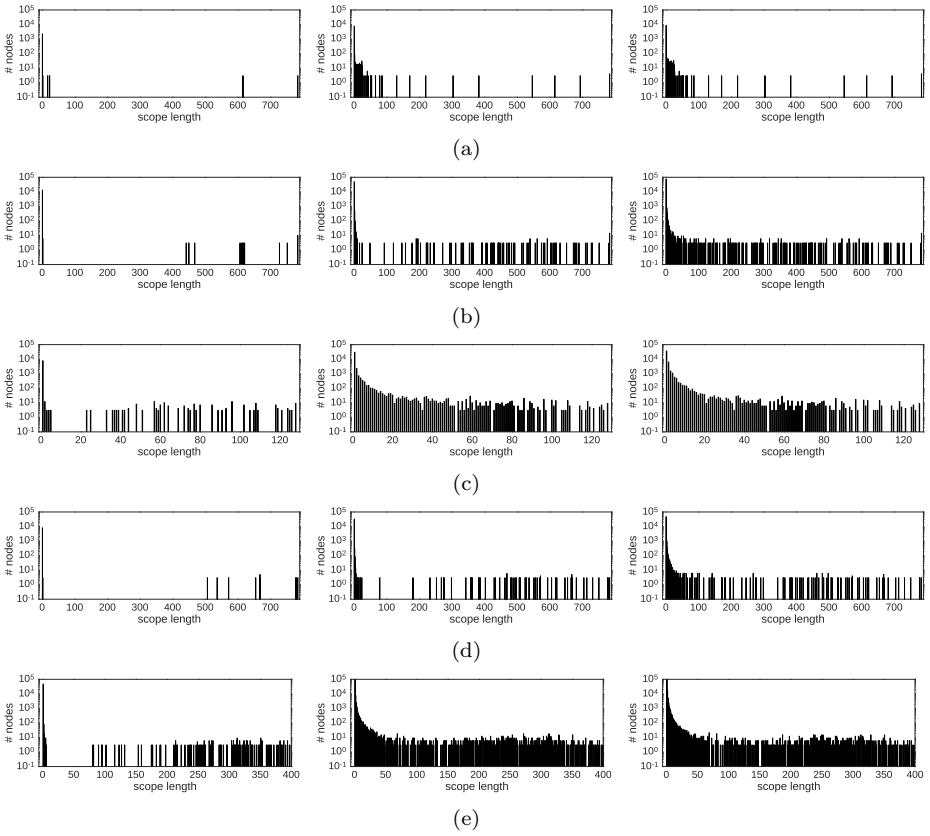


Fig. 2: Scope length distributions from SPN-I (left), SPN-II (middle) and SPN-III (right) models on REC (Figs. 2a), CON (Figs. 2b), OCR (Figs. 2c), CAL (Figs. 2d) and BMN (Figs. 2e, up to length 400 for the sake of readability).

how to take advantage of visualization techniques on SPNs to grasp their structural properties and workings of evidence, marginal and MPE inference and to determine the role of nodes as feature extractors. We also want to provide practitioners novel ways to inspect and assess SPN models. From this perspective, visualizations can be used as diagnostic tools to check for learning issues, e.g. the effect of regularization on model fitting, or to compare different models learned representations by visual inspection. In this section we will both leverage consolidated approaches and propose novel visualizations exploiting the interpretation of SPNs given in Section 3.

5.1 Visualizing scope distributions

Turned into sequential architectures of layers of alternated types (see Section 3 and Figure 1), our SPN-I, SPN-II and SPN-III reference models are indeed very deep, as reported in Table 1.

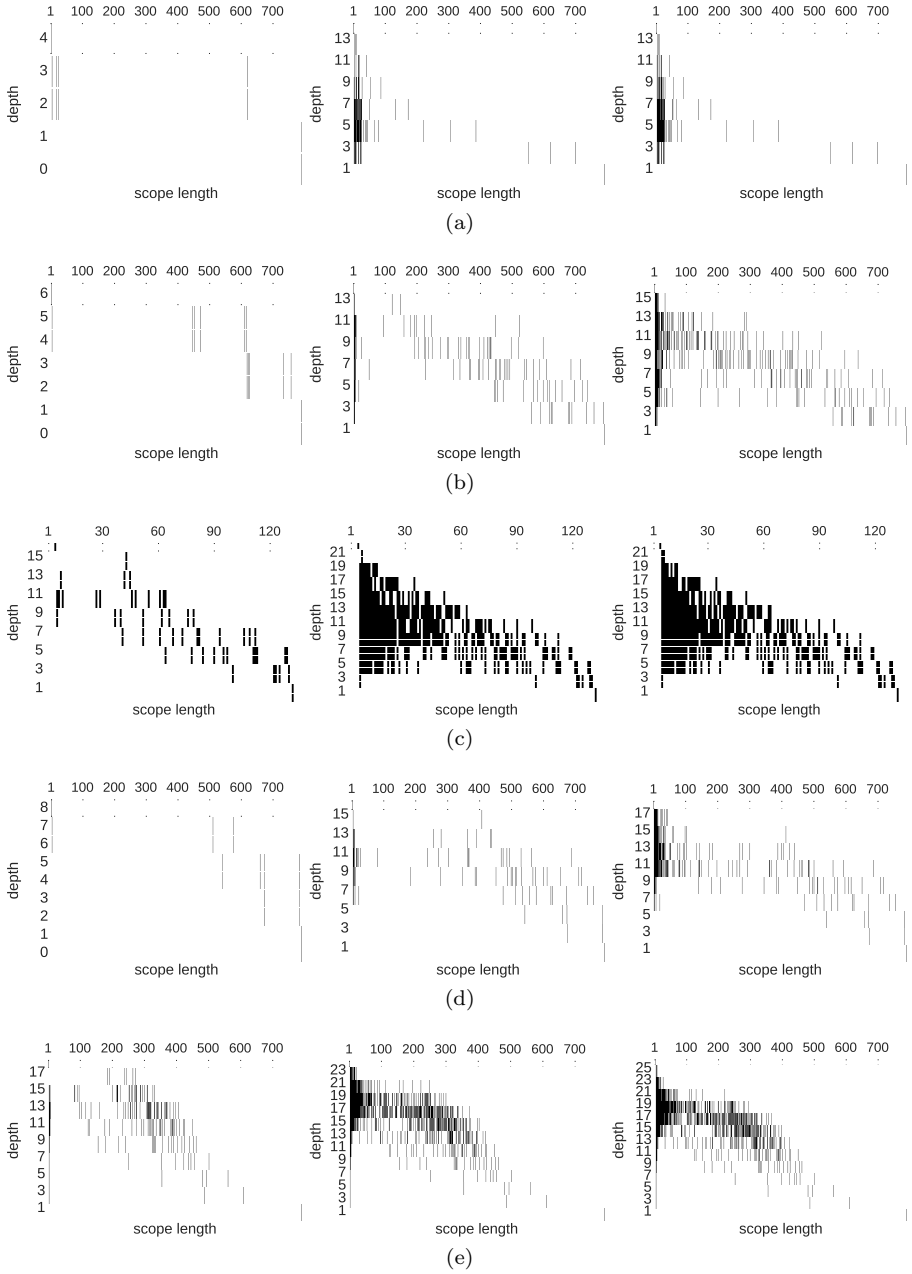


Fig. 3: Layer-wise scope length distributions for SPN-I (left), SPN-II (middle) and SPN-III (right) models for REC (Figs. 3a), CON (Figs. 3b), OCR (Figs. 3c), CAL (Figs. 3d) and BMN (Figs. 3e) datasets. A vertical bar in a layer indicates it contains at least one node of the corresponding scope length.

At first, we are interested in verifying our conjectures about the distribution of the nodes and their scopes to support the introduction of the scope length heuristics. If we plot scope length distribution for each model, as a histogram of the scope length possible values (from 1 to $|\mathbf{X}|$) against the number of nodes with that scope length, a power law effect is visible in Figure 2. 80% or more of the nodes in each model have a scope length of 1 to 3, while the remaining percentage is concentrated, for most of the models, up to 100 RVs. The rest of the long tail comprises scope lengths represented by very few nodes each (one on average).

We can also observe how leaves, having unitary scope lengths, are overabundant w.r.t. inner nodes. We decide not to include them in the following experiments about embedding extraction and evaluation. We adopt this representation bias, based on two considerations. First, we do not want our embeddings to have an excessive size. Such embeddings will likely incur in the curse of dimensionality, plus they would force us to build oversized embeddings for the competitor models for a fair experimental comparison. Second, we assume the discriminative power of univariate distributions as feature extractors to be far smaller than that of longer scope length features. We circumvent these issues by including leaf feature information in scope aggregated embeddings in Section 6.3.

Based on all these observations we define three scope length ranges to be later used in the following sections as a heuristics to group nodes as feature extractors at the same level of abstraction. We define Small scope lengths, comprising 2 to 3 RVs; scopes of Medium length containing up to 100 RVs for all datasets except for OCR for which we set it to 50; and lastly, a Large length category including all remaining lengths.

Then, we proceed to verify the arbitrary layered representation w.r.t. scope lengths. We visualize the scope length distributions for all layers of a model in Figure 3, in which each layer is labeled by its depth and each bar denotes the presence of at least one node of a certain scope length in layer. We confirm that multiple scope lengths are grouped together in a single layer and even if another layer partitioning would be possible, see Section 3, each node would still retain its depth, implying several scope lengths at the same depth. If the scope length of a node is proved to be a reasonable proxy measure for extracted feature complexity, this would imply that it is harder to directly exploit depth to determine features at different abstraction levels, as it happens with other deep architectures [14, 52].

5.2 Sampling

One of the most basic techniques to assess a generative model quality is the visual evaluation of its samples [20, 22]. To determine if a model has simply learned to reconstruct the training set, sampled instances are usually compared against the nearest training instances in the sense of the euclidean distance. An evaluation of this kind can be misleading in comparing models w.r.t. their loglikelihoods [46]. Nevertheless, we exploit sampling to investigate the effect of regularization on our reference models against their generative capabilities. We note that SPNs are prone to a very simple form of sampling, however, to the best of our knowledge, sampling has not been employed on SPNs in the literature. Apart from visual model assessment on image datasets, sampling can be effectively employed in comparison against probabilistic models for which the same likelihood measure is not easily

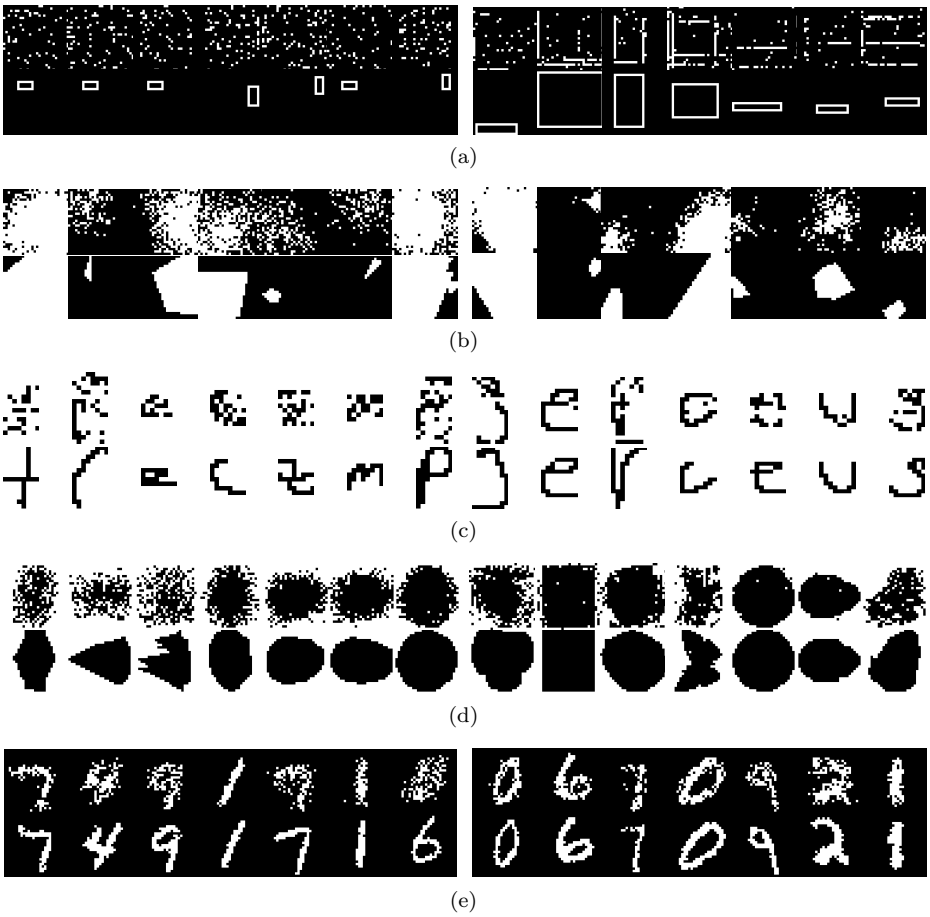


Fig. 4: Seven samples from SPN-I (left) and SPN-III (right) models on the first row, compared to their nearest neighbor images in the training set on row below for REC (Figs. 4a), CON (Figs. 4b), OCR (Figs. 4c), CAL (Figs. 4d) and BMN (Figs. 4e).

computable. In such cases, the same Parzen window density estimation can be applied on a congruous number of instances sampled from each model.

A sample from an SPN can be generated by remembering the probabilistic interpretation of the weights $\mathbf{w}$. To generate one sample, one traverses the network top-down. At each sum node n, the child branch c to follow is stochastically chosen with probability w_{nc}. Product node child branches are followed all together. Similarly to the descending step of MPE inference, in a valid network one is guaranteed to end up to a set of leaves representing exactly a complete scope partitioning. Then, each leaf can be asked to generate some observations for the RVs in its scope according to its own distribution.

To inspect our reference SPNs as generative models, we draw samples from the most and least regularized networks, SPN-I and SPN-III, for all datasets. Then we compare the samples to their nearest training instance as shown in Figure 4.

The presence of noise is evident for all models and datasets. However, SPN-III generated images are generally much less noisy and more recognizable, while SPN-I ones are less distinctive. This difference is more evident on CAL, suggesting a form of underfitting for the SPN-I model. On the other hand, on the same dataset SPN-III may have overfit since the samples seem very close reconstructions of their training counterparts. Whether this tendency also affects the extracted embeddings performances in a supervised task, shall be investigated in Section 6.

As a last remark, we note how the samples from REC suggest how both SPN-I and SPN-III architectures struggle to properly capture the straight line patterns. This in turn suggests that the learned SPNs are not capturing any form of translational invariance. The reason why this happens on REC and in a moderate form on CON, while is not visible with the other datasets, may lie in the fact that for OCR, CAL and BMN the training samples were all centered in their images.

5.3 Visualizing nodes as feature extractors

We now tackle the problem of extracting a visual representation for the learned features in our reference models. By exploiting the direct encoding that the scope function provides, we will investigate our conjectures about the hierarchy of representations in SPNs.

For deep neural networks, the visualization of the learned features can generally be done at a neuron level. The basic assumption is that the feature extracted by each neuron h_{ij} can be visually approximated by the representation in the input space that maximizes h_{ij} activation [14]. To reconstruct such a representation back into the input space $\mathbf{X}$, a common approach is to look for a bounded norm sample that solves this inverse non-convex maximization problem:

$$\mathbf{x}^* = \underset{\mathbf{x}, ||\mathbf{x}||=\gamma}{\operatorname{argmax}} \, h_{ij}(\mathbf{x}; \boldsymbol{\theta}) \ . \tag{6}$$

Its resolution can be tackled through SGD optimization after fixing all the network parameters $\boldsymbol{\theta} = \{\mathbf{W}^i, \mathbf{b}^i\}_{i=1}^{L}$, i.e. all weights and biases for all the L layers in the network. While the solver convergence is generally not guaranteed and the approach is feasible only for a certain number of layers, the empirical results on moderate sized networks suggest that the visualized representations are somehow stable across different iterates and are therefore representative of the extracted features [14].

SPNs lend themselves to an analogous problem formulation whose solution can be found without iterative optimization. First of all, recall that in an SPN S each inner node n defines a probabilistic distribution over its scope $\mathsf{sc}(n)$. This suggests that even the scope information alone, through a direct encoding of the input space, can enable a visualization of the learned features. To support this argument, consider the input domain of image samples, the visualization of the scope of each node corresponds to a shape against a background. If these shapes were meaningful by themselves, e.g. they are distinctive of some particular object, the scope information alone could be considered valuable enough.

Building on these considerations, one can reframe the problem of Eq. 6 for SPNs by recalling that maximizing a node activation equals to find its MPE assignment

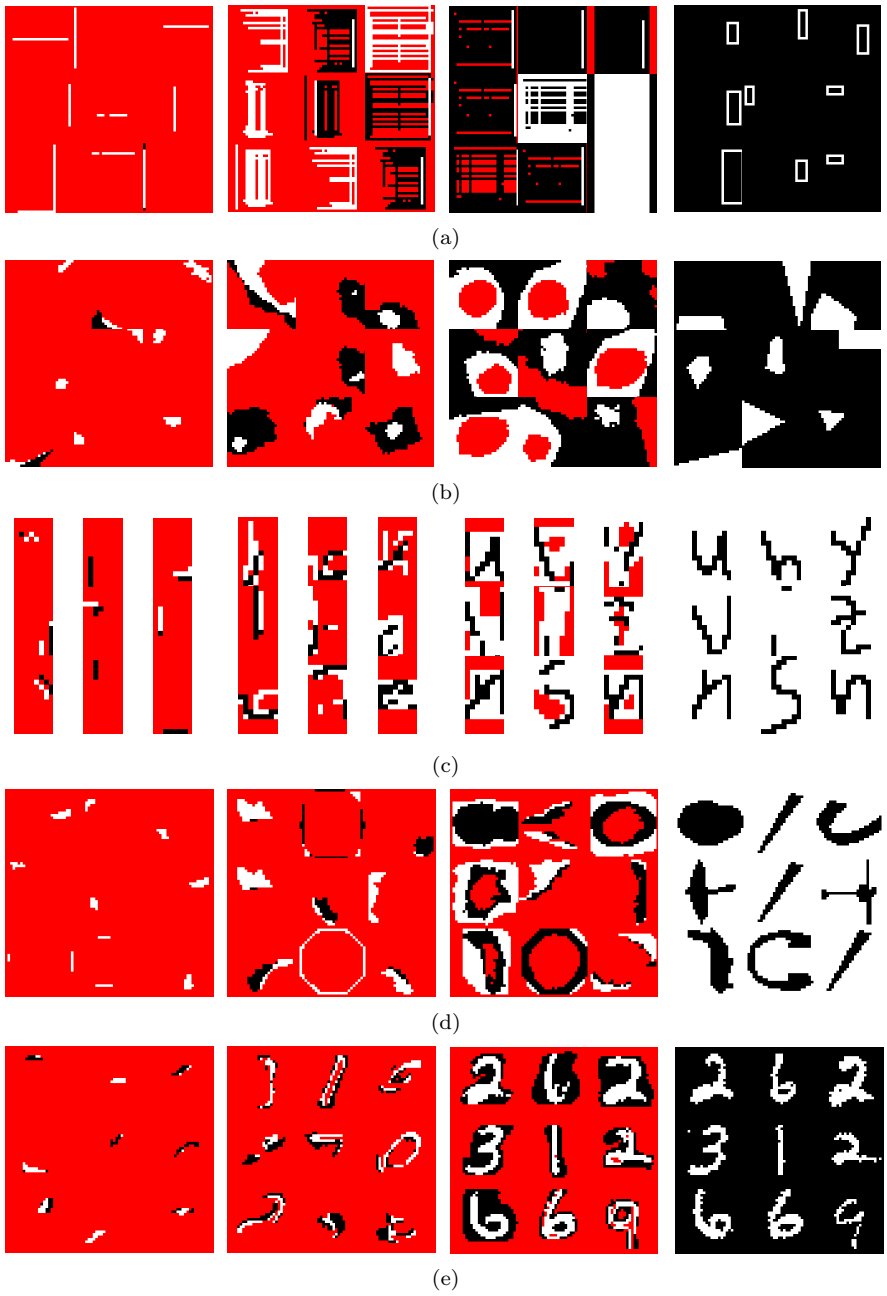


Fig. 5: Visualization of filters extracted from SPN-III models on REC (Figs. 5a), CON (Figs. 5b), OCR (Figs. 5c), CAL (Figs. 5d) and BMN (Figs. 5e). from 9 nodes with similar scope lengths of increasing sizes (columns 1 to 3). Column 4 shows the training images nearest to those in column 3.

limited to its scope $\mathsf{sc}(n)$:

$$\mathbf{x}^*_{|\mathsf{sc}(n)} = \operatorname*{argmax}_{\mathbf{x}} S_n(\mathbf{x}_{|\mathsf{sc}(n)}; \mathbf{w}) \ . \tag{7}$$

Leveraging the fully probabilistic nature of SPNs, this is equivalent to saying that the visual representation of a node as a feature extractor is the most probable sample according to the distribution encoded in the sub-network rooted at that node.

Even if exact MPE assignments in SPNs are not easy to compute (see Section 2.1), the approximate solution version still provides a valuable tool in understanding SPN nodes as filter. Moreover, consider that the joint approximate MPE assignment for all sub-networks distributions can be computed efficiently. To do so, one first evaluates the corresponding MPN network M once, after setting all leaves to output 1, then retrieves the best assignments by descending from each M_n.

We apply this procedure for all the fifteen reference models. To verity the validity of scope length heuristics as a proxy for determining the abstraction level of a representation, we inspect visualized representations (filters) of nodes sharing a similar scope length. Sample visualizations for different scope length ranges are shown in Figure 5. We color the pixels not belonging to a scope red, to stress how the depicted features are strictly bounded to portions of the input space and are not spatially invariant, differently from other neural network visualizations [14, 52]. Moreover, in this way we can better evaluate how the scope information alone can convey enough information. This is indeed true for many cases, since some filters are recognizable as meaningful object parts, e.g. see Figure 5d for an octagon shape or Figure 5e for digit strokes. But it seems not enough for high level features, i.e. features extracted from longer scope length nodes. Additionally, for the exception of REC, node scopes naturally form clusters of adjacent pixels. Recall from Section 2.2 that spatial autocorrelation is not taken into account by LearnSPN-b during learning. This visualization highlights how the learner was able to capture such dependencies by itself.

We confirm that features grouped by very different scope lengths appear meaningfully clustered. In fact, the visualized activations resemble part-based filters at different level of complexity: from pixel blobs to shape contours, to full shapes comprising background parts, e.g. Figure 5e. The role of SPN nodes as probabilistic part-based filters seems to be empirically confirmed on all datasets, for the exception of REC in which the meaning conveyed by the visualization is not immediate. This likely follows from the lack of translation invariance of SPNs: highly variant and not centered objects are harder to be decomposed in compact parts. CNN kernels, on the other hand, are not bounded by scope locations and are representations that can appear in every part of an image [52].

As a last consideration from this kind of visualization, if compared against their nearest training images, higher level features are not their exact reconstructions but they still appear to be very specialized filters. The issue of feature overspecialization will be tested in the RL experiments in Section 6.

5.4 Visualizing inference

We now tackle the visualization of different kinds of inference in SPNs, i.e. we visualize in the input space the node activations after a particular query has been

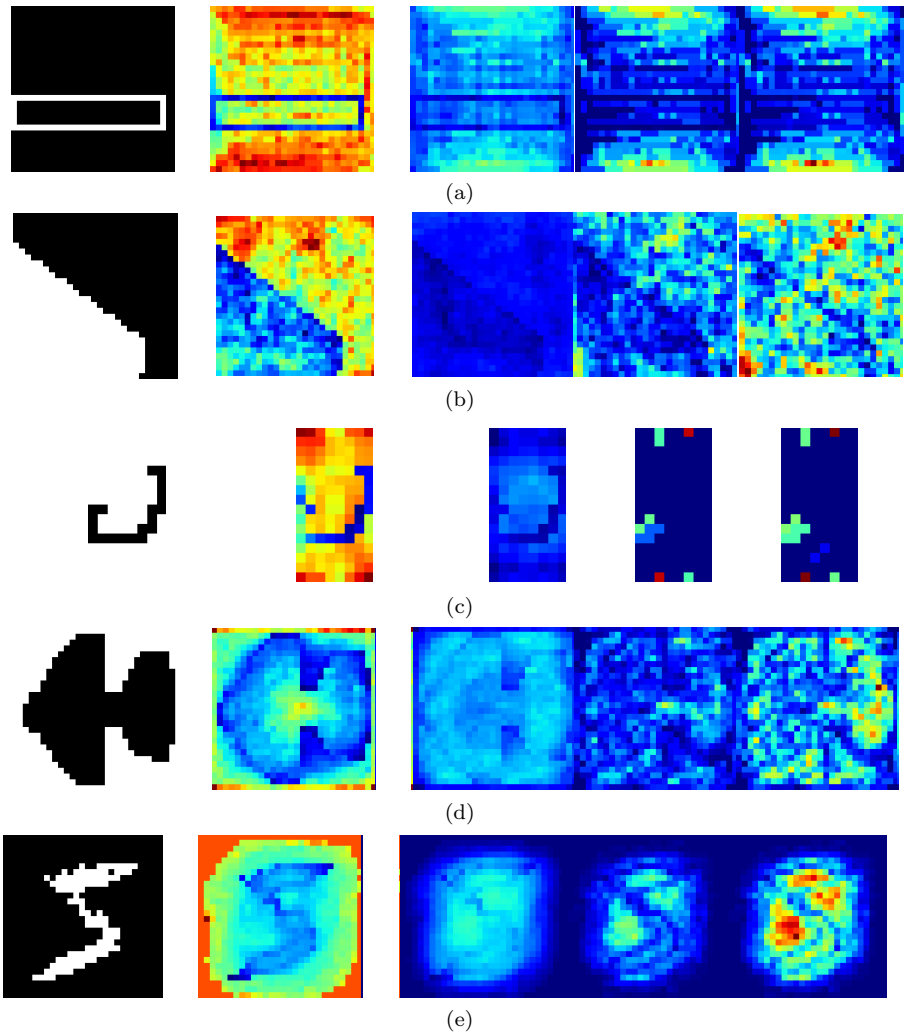


Fig. 6: Node activations visualizations for some samples (col. 1) for SPN-III models on REC (Figs. 6a), CON (Figs. 6b), OCR (Figs. 6c), CAL (Figs. 6d) and BMN (Figs. 6e). Stronger cumulative activations in red, weaker in blue. Showing evidence activations on all nodes (col. 3), normalized by scope (col. 2), only on sum (col. 4) and product (col. 5) nodes.

formulated. Our objective is to determine whether there are differences among nodes with different types while the network is evaluated, as well as investigate how the network structure affects inference. Again, we will provide several visualizations for our reference models showing their effectiveness as tools to compare and diagnose them.

Differently from MPE filters, the joint visualization of a set of node activations after a query is less immediate to map back to the input space. First, each visual

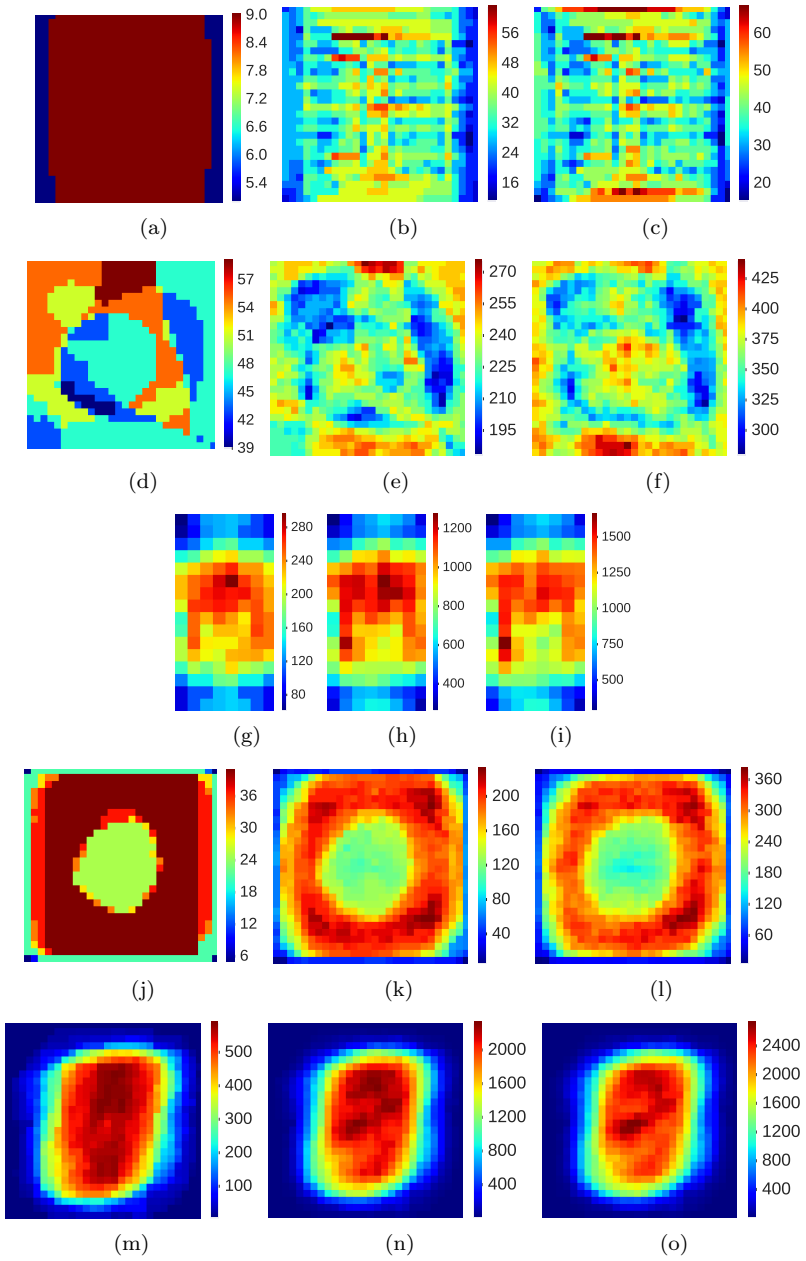
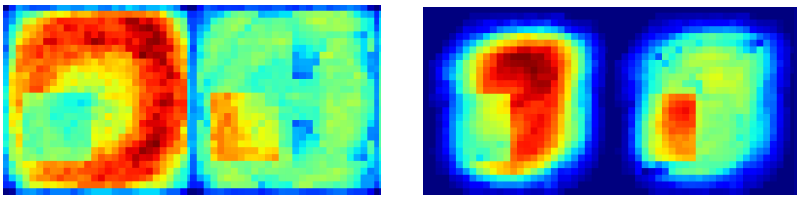


Fig. 7: Partition function computation visualizations for SPN-I (left), SPN-II (middle) and SPN-III (right) models on REC (Figs. 7a-7c), CON (Figs. 7d-7f), OCR (Figs. 7g-7i), CAL (Figs. 7j-7l) and BMN (Figs. 7m-7o).



(a) (b)

Fig. 8: Node activations visualizations for marginal queries on SPN-III models on CAL (Figs. 8a) and BMN (Figs. 8b) on the same samples used in Figures 6d and 6e. Stronger cumulative activations in red, weaker in blue. Marginalizing over a set of RVs from a patch of adjacent pixels (right) and over all the remaining RVs (left).

representation has to be done on a per instance or query basis, second, one has to deal with values coming from nodes with overlapping scopes. To cope with the latter issue, one approach one can take is to aggregate node outputs scope-wise. We propose to visualize the node activations as the result of summing their node probability outputs, thus emphasizing the pixel regions with the greatest total activation. For complete evidence queries, i.e. for an input image $\mathbf{x}$, we create a representation in the input space, $\tilde{\mathbf{x}}$, where each pixel is computed as

$$\tilde{x}_i = \sum_{n:X_i \in \mathsf{sc}(n)} S_n(\mathbf{x}) \ . \tag{8}$$

While this approach can help to highlight structural properties of learned networks, it is clearly biased towards regions having more nodes with overlapping scopes, thus falsifying the impact of some node against the others. To fix this bias we propose another representation, $\hat{\mathbf{x}}$, defined as:

$$\hat{x}_i = \sum_{n:X_i \in \mathsf{sc}(n)} S_n(\mathbf{x})/N_i \tag{9}$$

where N_i is the number of nodes having X_i in their scopes.

In Figure 6 we report some sample visualizations for SPN-III models after a complete image is considered as a query. Stronger cumulative activations are reported in red, while weaker in blue. By inspecting all node activations at first, one can note the emerging pattern of lower activations generally corresponding to the pixel regions near the query image shape in the normalized representation. This is less prominent in the unnormalized version. These aspects suggest how the probabilities coming from node activations act in transforming the input space into the representation space: discriminative features deviate from the mean sample which shall be represented with the highest probability, hence are assigned lower values. Again, we leave to the empirical evaluation in Section 6 the answer to whether or not this new representation space proves effective in a supervised classification task.

Continuing with complete evidence queries, we can limit the activations visualized to sum or product nodes only. Unnormalized sample visualizations in Figure 6 show how product nodes almost share the most intense activation areas with sum

nodes, but the cumulative activations of the former ones are more prominent. This aspect can be motivated with product nodes being more in number (see Table 1). It also supports the conjecture of sum nodes acting as compressors for product node outputs, as noted in Section 3. Whether or not the product node greater activations carry useful information will be tested in Section 6.2.

The unnormalized visualization of $\tilde{x}$ can be exploited also as a structural map showing where the "denser" area in a network are. Consider the visualization of the computation of the partition function Z (see Section 2.1), Eq. 9 would give a flat map of 1s for a valid SPN. On the other hands, the unnormalized representation can carry information about the node scope distributions in the network. In fact, applying Eq. 8 while all the $S_n(\mathbf{x})$ equal to 1 yields the count of nodes having a certain RV in their scope. We propose to visualize such a computation for all our reference models to compare their structural properties, both across the different datasets and regularization effect. Their visualizations are shown in Figure 7. It is evident how models learned on different datasets have "focused" their attention on different image regions, i.e. have employed a larger number of nodes to model more complex distributions for specific portions of the image. For instance, it reveals how the models on BMN concentrate its nodes on the center pixels (Figures 7m- 7o) while the ones on CAL focus on a ring-like area to better capture the silhouette outlines (Figures 7j- 7l). The effect of the regularization process is more evident on SPN-I models, for which large regions share the same numbers of nodes. In general, the less the regularized model, the more detailed the visualization. Figure 7d shows an interesting "segmented" visualization for SPN-I on CON, which suggests that the structure learning process may have completed only few clustering iterations. Again, this can be a hint at underfitting, as it appears to be the case for SPN-I on CAL (Figure 7j).

From this structural perspective, the visualization of complete evidence inference offers a way to recover a sort of "mean activation" signal. The lower activation regions in Figure 6 are representations deviating from this signal. To further analyze this aspect, we recur to marginal queries and visualize the node activations employing Eq. 8. We determine a set or RVs $\mathbf{Q} \subset \mathbf{X}$ corresponding to a square patch of pixels of size 10×10. We then ask a query to our models marginalizing over $\mathbf{Q}$ and another one marginalizing over $\mathbf{X} \setminus \mathbf{Q}$. The aim is to inspect the activations w.r.t. a set of focus RVs and while the remaining ones are 'don't care' RVs. Sample visualizations for these queries activations are reported in Figures 8a and 8b for starting image samples from CAL and BMN respectively. As expected, it is visible how the regions we marginalize over resemble the partition function computation, while the ones we are interested into are similar to portions of the visualizations for the complete evidence in Figure 6.

Up to now we have visualized several node activations given a certain query as these are the most natural constituents for building representations from an SPN. A complementary route one can take to extract features from an SPN is to collect the SPN root outputs only given several queries. From this perspective, one can compare the outputs of different models to the same inference query. Moreover, this approach offers a way to assess the goodness of a structure learner when comparing representations extracted from node activations against those generated from random queries. We will exploit this idea further in Section 6.4

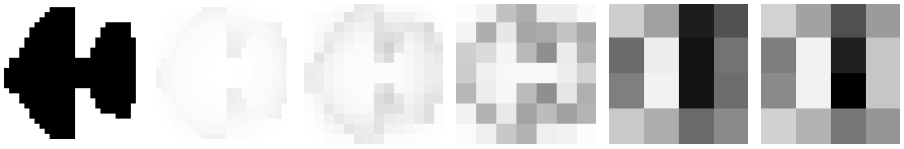


Fig. 9: Examples of different marginalization queries on a sample image from CAL (col. 1) for an SPN-I model, comprising marginalizations over 1×1 (col. 2), 2×2 (col. 3), 4×4 (col. 4), 7×7 (col. 6) pixel patches. The last image depicts the same 7×7 marginals as computed by SPN-III. Highest likelihoods in white, lowest in black (columns 2-6).

when comparing our reference models as feature extractors against other tractable probabilistic models.

If these are marginal queries, one can visualize their answer in the input space without coping with overlapping scopes if the queries concern disjoint sets of RVs. In this way we are extracting a representation from the root node outputs according to different portions of the input. That is, given a partitioning on the image pixels $\{\mathbf{Q}_i\}_{i=1}^{k}, \mathbf{Q}_i \subset \mathbf{X}, \mathbf{Q}_i \cap \mathbf{Q}_j = \emptyset, i, j = 1, \ldots, k$ in which each $\mathbf{Q}_i$ corresponds to a set of adjacent pixels, one input image can be visualized as colored in patches, each one representing the output of the SPN when asked to marginalize over all other RVs. That is, given an image $\mathbf{x}$, we are interested in visualizing each patch $\tilde{\mathbf{x}}_{|\mathbf{Q}_i}$ as:

$$\tilde{\mathbf{x}}_{|\mathbf{Q}_i} = S(\mathbf{x}_{|\mathbf{Q}_i}) = p(\mathbf{Q}_i = \mathbf{x}_{|\mathbf{Q}_i}) \ . \tag{10}$$

With this approach we are can look into user-defined regions even if there is no single node with that exact scope.

Figure 9 shows different sample visualizations for marginal queries on a sample image from CAL, comparing SPN-I model outputs on different pixel partitioning at different scales. It is again evident how shape contours are determined by low probability areas (darker patches). Clearly, the bigger the patch size the lower the likelihood values across all the dataset, but at the same time, the greater the differences between differently grained estimators like SPN-I and SPN-III (last image in Figure 9). With each model having its visual "signature", the effect of regularization across them is recognizable as the presence of darker patches.

In this Section we have gathered empirical evidences for our conjectures concerning questions **Q1** and **Q2**. We confirmed the meaningfulness of the scope and scope length heuristics by visualizing the extracted features by the means of MPE inference. Regarding question **Q3** we proposed several visualizations, among which the visualization of node activations, and in particular of the partition function computation, highlights how the structure learned can influence the extraction of features belonging to particular scope areas. In the end, we indirectly tackled the problem of how to compare SPNs to other probabilistic models (questions **Q3** and **Q6**) with our visualizations. More specifically, region based marginal query visualizations as signature patches, can be applied to other tractable models. Moreover, they suggest the experimental comparison protocol we employ in Section 6.4.

6 Representation Learning with Sum-Product Networks

In this section we empirically evaluate SPNs as feature extractors in a classical RL framework. From our reference models we actually extract different feature sets from each reference model and use each of them to train a linear classifier to predict the previously unseen class RV Y for each dataset. In a set of thorough experiments, we use these representation accuracy scores as a proxy to assess their usefulness and effectiveness. We point out how we are not interested in doing a state-of-the-art accuracy score on these datasets. Instead, we firstly want to investigate whether these representations are comparable against other commonly employed feature extractors for RL, like RBMs. In addition to that, we exploit such a comparison as a way to assess the contribution of different parts of a network structure in extracting new representations. Lastly, we devise a novel way to extract embeddings from tractable probabilistic models based on iterated random marginal queries, enabling a way to evaluate SPNs against other tractable models, as it has been hinted in the last part of Section 5.4.

More formally, given the dataset samples $\{\mathbf{x}^i\}_{i=1}^m, \mathbf{x}^i \in \{0,1\}^n$, an SPN reference model S and a filtering criterion f, we generate a new sample set $\{\mathbf{e}^i\}_{i=1}^m$ such that each embedding $\mathbf{e}^i \in \mathbb{R}^d$ is extracted from S according to f, i.e. $\mathbf{e}^i = f_S(\mathbf{x}^i)$. The chosen filtering criterion determines the dimension of the new vector space, d, which we will refer to as embedding capacity or size.

In all our experiments we employ a simple linear classifier to be trained on each embedding set to predict Y. The rationale behind this common RL approach is to inspect if the new geometric space induced by the embeddings has disentangled the input space enough to let a linear separator easily discriminate the associated classes [5]. We employ a logistic regressor with an L2 regularizer in a one-versus-rest setting. We leverage the implementation available in the scikit-learn framework[3]. For each experiment involving one feature set, we determine the L2 regularization coefficient C[4] value in $\{0.0001, 0.001, 0.01, 0.1, 1.0\}$, choosing the model with the best validation accuracy. As the simplest baseline possible, we apply such a classifier directly on the initial data representation, i.e. $\{\mathbf{x}^i\}_{i=1}^m$. We are denoting the model trained in such a way as LR in the following sections.

6.1 Node activations as features

As a first experiment we consider local interactions in the form of inner node outputs in an SPN S after evaluating $\mathbf{x}^i$, i.e. $e^i_j = S_j(\mathbf{x}^i)$ for a generic node set indexed by $j = 1, \ldots, d$. We collect all node outputs in the *log* domain to be able to represent even very small probabilities that would go to zero in the *exp* domain.

As the first filtering criterion, we extract all nodes in S whose scope length is strictly larger than one. We are interested in evaluate the performance of all nodes as feature extractors, however, comprising even the leaf nodes would determine too large embedding sizes (see Table 1 and Section 5.1). Even if a larger embedding capacity could definitely help a linear classifier in discriminating the disentangled representations, it could let it incur in the curse of dimensionality as

[3] `http://scikit-learn.org/`

[4] in scikit-learn this has the same meaning of the SVM regularization coefficient, hence smaller values indicate more regularized linear models.

Table 2: Test set accuracy scores for the embeddings extracted with the best SPN, RBM models and with the baseline LR model on all datasets. Bold values denote significantly better scores than all the others for a dataset.

	LR	SPN-I	SPN-II	SPN-III	RBM-5h	RBM-1k	RBM-5k
REC	69.28	77.31	**97.77**	97.66	94.22	96.10	96.36
CON	53.48	67.48	78.31	**84.69**	67.55	75.37	79.15
OCR	75.58	82.60	**89.95**	**89.94**	86.07	87.96	88.76
CAL	62.67	59.17	65.19	66.62	67.36	**68.88**	67.71
BMN	90.62	95.15	**97.66**	97.59	96.09	96.80	97.47

well. Additionally, it would pose a problem if we were to learn RBMs of comparable model capacity. Nevertheless, We investigate the contribution of the information coming from leaf nodes in Section 6.3.

As reference competitors we consider RBMs having 500, 1000 and 5000 hidden units (d) to take into account the effect of increasing a model representation capacity in the same way we did with our reference SPNs. We refer to them as RBM-5h, RBM-1k and RBM-5k respectively. We end up with fifteen RBM models as well. Applying the same experimental setting, we train them on the $\mathbf{X}$ alone by using the Persistent Constrastive Divergence (PCD) algorithm, leveraging the implementation available in scikit-learn. We follow [25] in which PCD has been demonstrated to be a very good overall performer across different benchmark tasks to train the weights of an RBM. For the weight learning hyperparameters we run a grid search for the learning rate in $\{0.1, 0.01\}$, the batch size in $\{20, 100\}$ and the number of epochs in $\{10, 20, 30\}$. We then select the best models according to their pseudo-log likelihood validation scores. To generate an embedding from an RBM model, we evaluate the conditional probabilities of the hidden units given each sample. To make the comparison fairer we transform these values in the *log* domain in the same way we do for our SPN representations.

Accuracy scores for the test splits of all five datasets, for LR, SPN and RBM reference models are reported in Table 2. From these results one can first note how LR model alone can score a 90.6% of accuracy on BMN, indicating the input representations for the ten classes to be already disentangled enough. By the same line of thought, CON can be considered a harder dataset, having LR score only 53.5% on a binary classification task. Each embedding set extracted from a reference model performs better that their baseline counterparts, meaning that they have been able to effectively disentangle the latent factors in the new representations, even if learned in an unsupervised way. The only exception seems to be the SPN-I model on CAL, scoring a poorer result which can finally be ascribed to underfitting after all the hints from the previous Section visualizations. On the other hand, even if the visualization of the samples from models trained on REC and CAL (see Figures 4a 4b) and of their extracted filters (see Figure 5a 5b) seemed poor, the improvement in their accuracy scores is quite relevant.

Concerning the regularization effect on the SPN reference models, one can observe a constant improvement gained by less regularized ones, even if the performance difference between SPN-II and SPN-III models can be negligible, e.g. on

Table 3: Test set accuracy scores for the embeddings extracted with SPN models and filtered by node type. Results for SPN-III embeddings filtered by Small , Medium and Large scope lengths are reported in columns 8-10. Bold values denote significantly better scores than all the others. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

	SPN-I		SPN-II		SPN-III		SPN-III		
	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

OCR and BMN datasets. A similar patter can be seen with more capable versus less capable RBM models. Only in one case, on CAL, RBM-5k overfit.

By comparing SPN reference model scores to RBM ones, it can be seen how the former are very competitive if not better than the latter. This is true for the less regularized SPN models on all dataset except for CON. It is also remarkable how high are these accuracy scores if one considers the highly greedy and fully unsupervised way in which the SPN structures have been learned. The best performance on REC, held by SPN-II, scores a classification error of 2.23% which is very close to the best error score achieved by the best *supervised* learner in [21]: 2.15% by an SVM with RBF kernel. In the case of CON, the 15.31% error score of SPN-III is even lower than the best one of 18.41% achieved by supervisedly trained Stacked Auto Encoders in [21].

6.2 Filtering embeddings

To explore the contribution of different nodes to the learned representations, we apply different filtering criteria that select a strict subset of the embeddings extracted in the previous section. At first, we filter features as extracted by the inner node types, to evaluate whether there is a pattern in sum versus product node embeddings. Additionally, we filter features based on the scope length of their node extractors, defining three embedding sets for Small, Medium and Large scope length ranges (see Section 4). We employ in this second experimentations only embeddings extracted from SPN-III models, as they have shown the more homogeneous distributions across all scope lengths (see Figure 3). In this case, we evaluate our scope length heuristic as a means to reduce the embedding size and to assess the importance of the representations at different levels of abstraction. It is worth noting that selecting only a fraction 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.

Test accuracy results for the five filtering criteria are reported in Table 3. For SPNs with fewer nodes, the product nodes seem to contribute most to the scored performance. On the other hand, when the model capacity seems congruous, e.g. with SPN-III models, sum nodes act as efficient compressors, greatly reducing the embedding size (see Table 1 as a reference) and preserving the accuracy scores achieved by the full embeddings or even increasing them. More generally, a holistic effect can be observed, sum and product nodes together score better, even if slightly, accuracies than alone, even when the size of a full embedding could incur in the curse of dimensionality.

The last column section of Table 3 reports the accuracies for the SPN-III embeddings filtered by the scope length ranges S, M, L. Embeddings from the smallest scope lengths are always the less accurate both than the full version and than the ones filtered from longer scope lengths. Even if they are the embeddings with the largest capacity (see Table 1), the meaningfulness of the extracted features is minimal, as conjectured in the previous Sections. However, also the contribution of the higher level features is less prominent. This confirms the intuition we had through the filter visualizations: high level features in our reference models may be too much specialized, and likely prone to overfitting. As a result, in general, selecting only mid level features proved itself to be a meaningful way to extract compressed, but still accurate, embeddings.

If accuracies from Table 3 are to be compared against those scored by RBM models, one can note filtering the SPN full embeddings produces representations whose size can be shorter than their RMB counterparts, while accuracies are comparable or better on three datasets over five. The embedding filtering process, when achieving better scores, can also improve on the performances reported in the previous Section about supervised models. The error rate of 2.20% on BMN achieved by the sum nodes of SPN-III is one remarkable example, as it is the 1.54% error rate scored by SPN-III M scope length embeddings on REC.

6.3 Aggregating scopes

In this Section we investigate a filtering criterion based on aggregating more node activations in a single feature. We propose to build an embedding based on the aggregation of node outputs having the same scope, leveraging the idea that all the nodes sharing the same scope are extracting different features over a same latent factor. The most natural aggregation function we can employ is the mean, thus actually computing for each possible scope j in S

$$e_j^i = \frac{1}{|\{n | n \in S, \mathsf{sc}(n) = j\}|} \sum_{n \in \{n | n \in S, \mathsf{sc}(n) = j\}} S_n(\mathbf{x}^i) \tag{11}$$

where $S_n(\mathbf{x}^i)$ values are in the *exp* domain and the final computation is projected in the *log* domain for a fair comparison against all other embeddings involved in the previous experiments.

The question concerning which nodes to consider for each aggregation can be answered in different ways. We note how constructing embeddings according to Eq. 11 equals to collect the output of a fictitious valid sum node introduced in the network to compute a uniform mixture over all nodes sharing a same scope.

Table 4: Test set accuracy scores for the embeddings extracted with SPN models by aggregating node outputs with the same scope. Results for when leaves are not counted in the aggregarion (no-leaves columns) are reported alongside the case in which they are considered (leaves columns). Bold values denote significantly better scores than all the others for each dataset. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

	SPN-I		SPN-II		SPN-III	
	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**

From this perspective, we are providing our SPN with a number of additional sum root nodes, each one representing a particular scope in the network. Such a representation could be exploited also in other learning schemes, e.g. weight learning to ameliorate the vanishing gradient issue afflicting SPNs [36]. Following all these considerations, we decide to aggregate only sum node outputs and not product node ones, since the latter already contribute to their sum node parent feature extractions. As an alternative filtering criterion, we propose to aggregate 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, while keeping the embedding capacity small thanks to the mean aggregation. Refer to Table 1 to determine these embedding sizes as the number of unique scopes in our reference models.

Table 4 shows the test accuracy results for all the reference models when leaves are included or not in scope aggregation embeddings. It is visible how leaf addition helps models with lower capacity like SPN-I ones, scoring the best accuracy for them on CAL. As the model capacity increases, the contribution of leaves becomes marginal or even null.

As a general tendency, accuracies for the leaves-included variants are comparable to the best sum embeddings for the corresponding models, while their sizes are reduced. This is an empirical confirmation of the goodness of scope aggregations as a heuristic to extract embeddings from an SPN.

As a recap about all the filtering schemes employed so far and to better compare their accuracy scores w.r.t. their embedding capacities, we present a series of scatterplots in Figure 10. We repeat our general findings: sum node activations alone act as sufficient compressors for less regularized models; mid level of abstraction representations, i.e. embeddings belonging to nodes with medium scope lengths, are a good heuristic to preserve test accuracy and reduce while reducing the embedding size; scope aggregations prove to be an additional effective embedding size reduction heuristic.

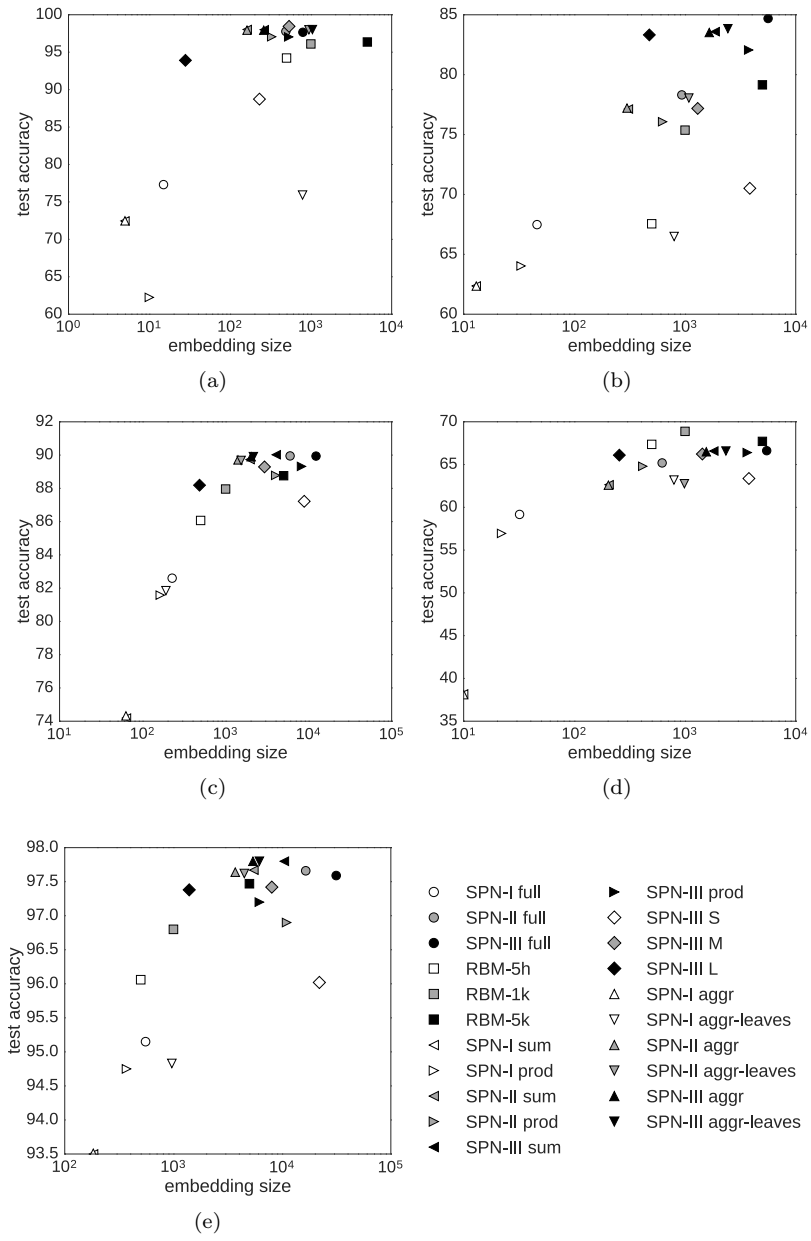


Fig. 10: Embedding capacity versus test accuracy for SPN and RBM models on REC (Figs. 10a), CON (Figs. 10b), OCR (Figs. 10c), CAL (Figs. 10d) and BMN (Figs. 10e) according to the different filtering criteria employed: all inner nodes (full), only sum or product nodes, filtered by scope length of small (S), medium (M) and large (L) sizes and scope aggregations without (aggr) or with leaves (aggr-leaves).

6.4 Probabilistic queries as features

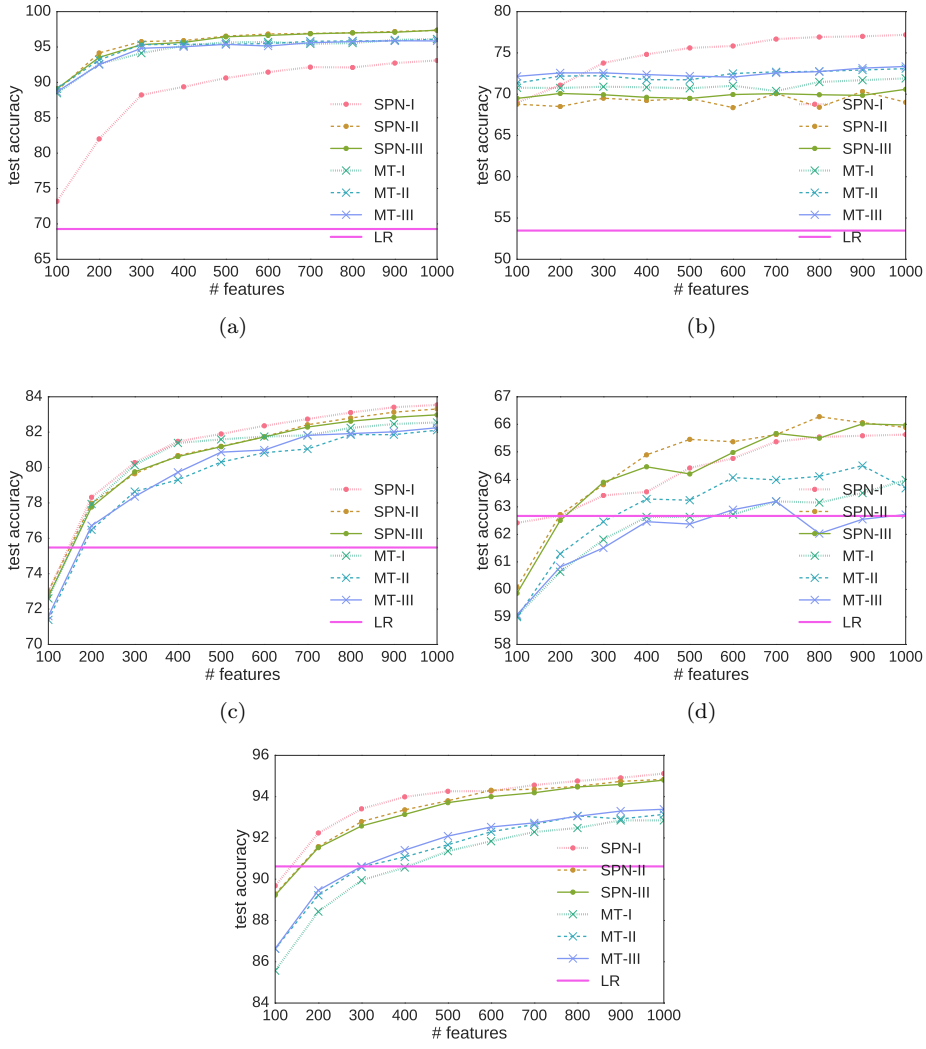


Fig. 11: Test accuracies for SPN and MT models on REC (Figs. 11a), CON (Figs. 11b), OCR (Figs. 11c), CAL (Figs. 11d) and BMN (Figs. 11e). against 1000 features generated as random marginal queries evaluations. Logistic regressor as a baseline is reported as LR.

In this Section we investigate another take in extracting embedding from SPNs by evaluating the networks several times against a fixed set of template queries. By

doing so we are registering only the root activations, i.e. the network outputs to some inputs, treating the density estimator as a *black box* inference tool. Actually, such embedding generation scheme is applicable not only to SPNs but also to every tractable probabilistic models for which the template queries can be answered in tractable time. To the best of our knowledge, this is a novel way to exploit tractable probabilistic models to generate embeddings and can be proved useful in extracting representations from already learned density estimators to be employed in new tasks. Moreover, it is a model agnostic way to compare models previously not comparable on the same supervised task, similarly to how Parzen windows are now exploited to compare different generative models likelihood performances.

We employ marginal queries following the considerations from Section 5.4 to generate query templates over some randomly chosen sets of RVs to be asked to our reference models. We extend our comparison to the Mixture of Trees distributions (MT) [27]. Despite their simplicity, they have been proven to be very competitive against SPNs, likelihood-wise [47]. MT models define a mixture model over tree distributions, for which marginal inference can be computed in linear time in $|\mathbf{X}|$ [27]. We are interested in performing structure learning on such models. MT trees are learned with the Chow-Liu algorithm [8], while the mixture coefficients are estimated through the EM algorithm. We employ mtlearn, the implementation of MT available in the Libra toolkit [24]. To investigate different model capacities as we did for all our reference models, we learned mixtures with 3, 15 and 30 trees for each dataset (fifteen models total), from here on denoted as MT-I, MT-II, MT-III respectively.

The process to generate the query templates is as follows. For each feature j we have to generate, $j = 1, \ldots, d$, we stochastically select a scope $\mathbf{Q}_j \subset \mathbf{X}$ and then we compute each embedding component as the evaluation of a marginal query over Q_j according to a certain probabilistic model, i.e. $e^i_j = p(\mathbf{Q}_j = \mathbf{x}^i_{|\mathbf{Q}_j})$. For an SPN model S this reduces to compute $S(\mathbf{Q}_j = \mathbf{x}^i_{|\mathbf{Q}_j})$. To understand why tractable marginal inference is mandatory, consider this: in a naive computation one would have to ask a model to answer $d \cdot m$ marginal queries, and even if one cashes the answers for each possible observable state configuration for the RVs $\mathbf{Q}_j$, assuming all of them to have at most l values, $l < m$, it would end up with $d \cdot l$ evaluations. Being these random query evaluations, one would expect d to be large enough to let some statistical pattern emerge. As a practical example, since in our experiments we set $d = 1000$ as a reasonable large enough embedding size, the naive computation of such an embedding set for the test split of REC alone would require 50000000 query evaluations.

This kind of constrained evaluation offers also a way to assess the meaningfulness of the nodes generated by the structure learner algorithm as feature extractors, when compared to ones whose scope is randomly generated. Remember that a node n in valid SPNs defines a distribution over its scope. One could think at this distribution as the marginalization of the whole joint over the scope of n, that is $S_n(\mathbf{x}) \approx S(\mathbf{x}_{|\mathsf{sc}(n)}) = p(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$. This is clearly not an equality but a very rough approximation. It is moved by the intuition gained from the visualization of marginal inference compared to the whole partition function computation (see Section 5.4). If one accepts this sort of approximation, then the answer for a marginal query $p(\mathbf{Q}_j)$ can be thought as the output of a fictitious node with scope $\mathbf{Q}_j$. By comparing the accuracy scores of features extracted from these randomly

generated fictitious nodes against those collected from nodes built by a structure learning algorithm, one can evaluate this structure learner effectiveness from a non likelihood-wise point of view.

We evaluate all the SPN and MT models on the same 1000 randomly generated query templates on each dataset. For each query we select RVs corresponding to adjacent pixels in a rectangular image patch having minimum sizes of 2 pixels and maximum of 7 pixels for OCR and 10 pixels for all the other datasets. Then, we train a logistic regressor on each embedding set as in the same experimental protocol employed for the other experiments.

Figure 11 shows the accuracy scores by SPN-I, SPN-II, SPN-III and MT-I, MT-II, MT-III on each dataset, measured while adding 100 features at a time. It is visible how such embedding accuracies improve as the number of feature increases. This is true in almost all scenarios with the exception of REC and CON. On these two datasets, a sort of model "herding" can be noted: model performances are not growing noticeably while features are added, except for the more reguarized model, SPN-I on REC and of the least regularized one, SPN-III on CON. We experimented on OCR and BMN by adding other 1000 randomly generated features, however, the accuracy gained was almost null.

All the LR baselines are overcome by both SPN and MT models. No more than 300 features appear to be enough for SPNs to beat LR scores. This demonstrate how this representation extraction scheme can be effective empirically. Instead, if compared to the embeddings extracted as node activations, these embeddings perform worse in almost all cases. This is an empirical affirmative answer to whether or not LearnSPN-b built meaningful networks even from a RL point of view. However, it shall be noted that heavily regularized models like SPN-I perform far better in this setting than in the ones from previous Sections. This suggests that an SPN representation capacity could go beyond the actual number of nodes in the network (remember SPN-I to have very few inner nodes, see Table 1).

Lastly, if compared to MT models, SPN ones are performing definitely better on OCR, CAL and BMN, on the other hand, only SPN-I on CON demonstrates far superior accuracies after 300 features. A similar argument about a regularized model capacity can be stated in a minor way for MT models on CAL and BMN.

In this Section we definitely confirm the validity of the scope length criterion to extract features at certain grains of complexity (questions **Q2**, **Q5**). From our supervised RL experimentations we proved SPNs to be more than simply competitive to RBMs as feature extractors (question **Q4**) evaluating embeddings generated with several criteria (question **Q5**). We also devised a way to extend this kind of comparison to other tractable probabilistic models, proving not only SPNs to be competitive against MT models (question **Q6**) but also assessing the meaningfulness of the structure learned by LeanSPN-b by comparing the embeddings from their inner node activations against those generated from random marginal queries (question **Q3**).

7 Related Work

7.1 SPNs

The *network polynomial* has been introduced for probability distribution encoded by BNs by Darwische, along the differential approach to inference in [10, 11]. Its extended version has been presented in [34] as a means of performing the differential approach on SPNs whose leaves model any kind of tractable distribution (as the ones we are referring to), even continuous pdfs.

MPE inference and MPE assignments have been thought to be exactly computable in linear time in SPNs as well [36, 12]. However, only recently, MPE inference has been stated to be hard [29, 32] in SPNs, and an MPE assignment has been proved to be found efficiently only on *augmented SPN*s, networks in which the latent RVs associated to sum nodes are made explicit, providing a form of determinism that enables the correct maximization of the network polynomial.

Weight learning for SPNs has been proposed in different fashions. Given an SPN initialized with a generic but valid architecture, weight learning can be done by employing the EM approach or gradient descent as in [36]. The weights can be updated by gradient steps since in SPNs efficient computation of the likelihood gradient can be obtained by backpropagation. EM, on the other hand, exploits the latent variable semantics of SPNs. The computation of the gradient for very deep SPNs can be an issue [36]. This problem has been overcame in [36, 15] by adopting a hard gradient approach, i.e. exploiting MPE inference to update the weights along a descent path by hard counts.

Instead of using a generative likelihood loss, in [15] a discriminative one has been employed in a hard gradient descent scheme leading to discriminative training of SPNs, obtaining state-of-the-art results on some standard image classification tasks. However, having to deal with a fixed, hand crafted network structure, has determined these algorithms to be less and less popular. Only very recently, weight learning has regained interest with novel online and distributed learning schemes. However, it is still hard to obtain a largely better likelihood score than an SPN whose structure has been learned as well, by just employing weight learning. Some interesting recent attempts comprise [38], proposing the online Bayesian Moment Matching algorithm (OBDMM), and [53], implementing a collapsed variational approximation for bayesian weight learning. In the former, an extensive empirical comparison shows OBDMM effectiveness and scalability in comparison to online extensions of gradient descent, exponentiated gradient and EM on many benchmarks datasets. In the latter, the new bayesian approach outperforms OBDMM both in the online and distributed settings.

Structure learning algorithms, on the other hand, have been largely investigated. As already stated in Section 2, many of them share with LearnSPN the intuition behind the iterative clustering processes along the instance and feature dimensions. More different structure learning schemes can be found in [30] in which a bottom-up approach is employed, still exploiting the probabilistic interpretation of sum and product nodes. A likelihood score-based local search approach has been tried in [31] on a deterministic version of SPNs, Selective SPNs, for which the likelihood function is computable in closed form.

While the theoretical properties of SPNs are being thoroughly investigated, their node interactions and practical interpretability have received little or no attention.

These recent works focused on studying SPNs expressive power or relating them to other probabilistic models. In [26] it is demonstrated how their expressive efficiency potentially increases as their depth grows. In [34] the notion of generalized network polynomial is introduced, and it was shown that consistency, a less strict constraint for product nodes still ensuring validity, does not lead to exponentially more compact networks. In [54] it is demonstrated how they are equivalent to bipartite BNs with Algebric Decision Diagrams modeling their conditional probability tables.

We follow [47] and argue that likelihood scores alone do not provide a complete picture and a qualitatively and quantitatively comparison of learned architectures structural properties is needed.

7.2 Visualization Tecniques

Visualizations provide important tools to assess a model from a qualitatively perspective, and have proven to be complementary to quantitative analysis. The visualizations of what has been learned by a neural network can be traced back to scientific visualizations of the sign and magnitude of an MLP weight matrices, techniques like Hinton's diagrams and bond diagrams are reviewed in [9]. Several visualizations of deep architectures have been proposed in the literature, the most common and simplest one being the visualization of sampled instances [20, 22, 46].

Recently, the need of understanding deep models successes more *in depth*, lead to studies focused on particular architectures, highlighting their peculiarities and investigating how to exploit them. For instance this has been done for CNNs in [52] and for Recurrent Neural Networks even more recently in [17]. In [52] a new visualization technique, using a multi-layered Deconvolutional Network, to project the feature activations back to the input pixel space as in [51], has been introduced. It is able to reveal the input sample that excite individual nodes the most at any layer in the model. Such a visualization has been exploited to handcraft a CNN architecture scoring state-of-the-art accuracy results on a benchmark image recognition task.

In this paper, we followed the work in [14] to visualize each neuron learned feature from an arbitrary layer as the input instance maximizing its activation, as already described in Section 5.3. Extending the work of [14], recent works have explored how to impose natural image priors while determining the maximally activating images for a neuron. In [43] the optimization problem is recast in two variants: finding the best image maximizing a class score and computing a saliency map for a query image sample, given a class. The same approach is adopted in [50] in which, however, a stronger prior as a regularizer is added to the loss score and an efficient software tool able to visualize all hidden neurons is presented. Employing visualizations as debug tools to spot "blind spots" for networks, adversarial images, i.e. images crafted to fool a network output, are synthesized in [45].

7.3 Representation Learning

Representation Learning works have extensively studied how to extract useful features in unsupervised, semisupervised and supervised settings from both deep and shallow models. For a comprehensive review of the field of RL, see [5]. RBMs

alone have been employed in several studies comparing different weight learning regimes and evaluating them both as generative models and feature extractors, see [19, 20, 25] just to cite a few. They also have been employed as the inspiration to build successful autoregressive models like the Neural Autoregressive Distribution Estimator (NADE) [22]. In [22] their likelihood formula is employed to derive a neural formulation for a step of a mean field inference routine. Along RBMs, a popular architecture are Denoising Autoencoders (DAE) [48]. For all these neural network 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, even if greedy, structure learning schemes are possible. They both enable a "deeper" form of RL, in which the extracted representations can be assessed against the learned structure and vice versa.

To the best of our knowledge, SPNs have never been employed as feature extractors. The only task in which they have been used to produce a structured output is input reconstruction. MPE inference is exploited to predict the value of RVs unobserved at test time, e.g. the pixels in an image [36] or frequencies in a spectrogram [33]. However, in all those cases they have been treated as black box density estimators.

In [45] it is argued that the representational power of a deep architecture shall not be searched in individual neurons but in the space generated by all high level layers. This is proven by comparing high level neuron outputs and random combinations of them, showing little variations performance-wise. This is somewhat similar to the intuition we had in Section 6.4 when we observed SPN-I models to perform better when random fictitious node outputs were collected, although this is only a speculation to be investigated more extensively.

8 Conclusions

In this work we investigated how the internal representations learned by SPNs can be extracted, exploited and better understood. We have explored different visualization techniques on the computation of evidence, marginal and MPE queries. We employed several embedding sets extracted from our reference models to empirically evaluate their meaningfulness in a supervised classification task.

Concerning **Q1** we argue that even if SPN can be reframed as labelled constrained and fully probabilistic MLPs, their structures may not be suited to extract embedding in the classic layer-wise setting. For this reason we advance a scope heuristics to aggregate features at different levels of abstract. We confirmed the meaningfulness of using scope length ranges to correlate a node feature abstraction level first visually by the means of MPE inference and then experimentally. We found medium sized scope length embeddings to provide the best accuracies (**Q2**). We investigated the impact of the learned structure on network inference and on the learned representations. We noted how the visualization of node activations aggragated by scope can provide visual clues on the denser sub-structure. A different role for product nodes has been hinted in such visualizations and confirmed in the classification task but only for high capacity models. Sum nodes have been demonstrated to provide the best size versus accuracy compromise. All in all a holistic effect can be seen and the representation capacity of an SPN seems to go beyond its single node activations (**Q3**). The embedding extracted from SPNs have

been proven to be rather competitive against the ones extracted from RBMs of even or more capacity This has been verified for several filtering criteria as well. Along with the already stated sum node and medium sized scope length filtered embeddings, scope aggregation provides compact and accurate representations. Their increased accuracy for small capacity models derives from leveraging the information of previously excluded leaf nodes (**Q4**, **Q5**). Lastly, comparing the node activation embeddings against those generated by random marginal queries reveals the meaningfulness of the structure learned in a greedy unsupervised manner, but also suggests the representation power of an SPN to go beyond its single node outputs (**Q3**, **Q6**).

Hence, we can confirm the three objectives we set at the beginning. We obtained a better understanding of the inner workings of SPNs (**O1**), by visualizing their structure and behaviour under different types of inference queries, plus, we empirically evaluated the usefulness and meaningfulness of different representations one can extract from them (**O2**), and by doing so we provided alternative ways to assess a learned SPN value (**O3**).

As future works, We plan on investigating the random query RL scheme in a more extensive way, e.g. by exploiting other types of probabilistic queries to generate features or studying how this can be related to the effective expressive capacity of these architectures. Another interesting aspect to research is how to exploit further the learned representations. As a first option, stacking more density estimators on these representations seems quite challenging. A more direct way to follow can be directly plugging them in end to end deep architectures to train them both as density estimators and classifiers at once, e.g. in a semi-supervised learning scheme.

Acknowledgements The authors wish to thank Pierpaolo Basile for the computational resources kindly offered.

References

1. Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm. In *Uncertainty in Artificial Intelligence*, 2015.
2. Mohamed Amer and Sinisa Todorovic. Sum product networks for activity recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2015.
3. Mohamed R Amer and Sinisa Todorovic. Sum-product networks for modeling activities with stochastic structure. In *(CVPR), 2012 IEEE Conference on*, pages 1314–1321. IEEE, 2012.
4. Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.
5. Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised Feature Learning and Deep Learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.
6. Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

7. Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Adam Chai. Language modeling with Sum-Product Networks. In *INTERSPEECH 2014*, pages 2098–2102, 2014.
8. C Chow and C Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
9. Mark W. Craven and Jude W. Shavlik. Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, 1:399–425, 1991.
10. Adnan Darwiche. A differential approach to inference in bayesian networks. *J.ACM*, 2003.
11. Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge, 2009.
12. Aaron Dennis and Dan Ventura. Learning the Architecture of Sum-Product Networks Using Clustering on Varibles. In *Advances in Neural Information Processing Systems 25*, pages 2033–2041. Curran Associates, Inc., 2012.
13. Aaron Dennis and Dan Ventura. Greedy Structure Search for Sum-product Networks. In *IJCAI'15*, pages 932–938. AAAI Press, 2015.
14. Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network. *ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.*, 2009.
15. Robert Gens and Pedro Domingos. Discriminative Learning of Sum-Product Networks. In *Advances in Neural Information Processing Systems 25*, pages 3239–3247, 2012.
16. Robert Gens and Pedro Domingos. Learning the Structure of Sum-Product Networks. In *Proceedings of the ICML 2013*, pages 873–880, 2013.
17. Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.
18. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
19. Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the ICML 2008*, pages 536–543, 2008.
20. Hugo Larochelle, Yoshua Bengio, and Joseph P. Turian. Tractable Multivariate Binary Density Estimation and the Restricted Boltzmann Forest. *Neural Computation*, 22, 2010.
21. Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. In *Proceedings of the ICML 2007*, pages 473–480, 2007.
22. Hugo Larochelle and Iain Murray. The Neural Autoregressive Distribution Estimator. In *International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.
23. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
24. Daniel Lowd and Amirmohammad Rooshenas. The Libra Toolkit for Probabilistic Models. *Journal of Machine Learning Research*, 16:2459–2463, 2015.

25. Benjamin M Marlin, Kevin Swersky, Bo Chen, and Nando D Freitas. Inductive Principles for Restricted Boltzmann Machine Learning. In *AISTATS 2010*, pages 509–516, 2010.
26. James Martens and Venkatesh Medabalimi. On the Expressive Efficiency of Sum Product Networks. *CoRR*, abs/1411.7717, 2014.
27. Marina Meilă and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.
28. Aniruddh Nath and Pedro M. Domingos. Learning tractable probabilistic models for fault localization. *CoRR*, abs/1507.01698, 2016.
29. Robert Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Graz University of Technology, SPSC, 2015.
30. Robert Peharz, Bernhard Geiger, and Franz Pernkopf. Greedy Part-Wise Learning of Sum-Product Networks. In *ECML-PKDD 2013*, 2013.
31. Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *Workshop on Learning Tractable Probabilistic Models*. LTPM, 2014.
32. Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M. Domingos. On the latent variable interpretation in sum-product networks. *CoRR*, abs/1601.06180, 2016.
33. Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP2014*, 2014.
34. Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. *The Journal of Machine Learning Research*, 2015.
35. Hoifung Poon. Tutorial on spn. *NIPS2011*, 2011.
36. Hoifung Poon and Pedro Domingos. Sum-Product Networks: a New Deep Architecture. *UAI 2011*, 2011.
37. Tahrima Rahman and Vibhav Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, pages ??–??, 2016.
38. Abdullah Rashwan, Han Zhao, and Pascal Poupart. Online and distributed bayesian moment matching for parameter learning in sum-product networks. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1469–1477, 2016.
39. Martin Ratajczak, S Tschiatschek, and F Pernkopf. Sum-product networks for structured prediction: Context-specific deep conditional random fields. *Proc Workshop on Learning Tractable Probabilistic Models*, 1:1–10, 2014.
40. Amirmohammad Rooshenas and Daniel Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In *Proceedings of ICML 2014*, 2014.
41. Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996.
42. Ruslan Salakhutdinov and Iain Murray. On the Quantitative Analysis of Deep Belief Networks. In *Proceedings of the ICML 2008*, volume 25, 2008.
43. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
44. Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

45. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.
46. L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, Nov 2016.
47. Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning. In *ECML-PKDD 2015*, 2015.
48. Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of ICML 2008*, 2008.
49. Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.
50. Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.
51. M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *International Conference on Computer Vision*, pages 2018–2025, 2011.
52. Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of the 13th European Conference on Computer Vision*, pages 818–833. Springer International Publishing, 2014.
53. Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed variational inference for sum-product networks. In *In Proceedings of the 33rd International Conference on Machine Learning*, volume 48, 2016.
54. Han Zhao, Mazen Melibari, and Pascal Poupart. On the Relationship between Sum-Product Networks and Bayesian Networks. In *ICML*, 2015.