Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks

Antonio Vergari
antonio.vergari@uniba.it
University of Bari, Italy

Robert Peharz
rp587@cam.ac.uk
University of Cambridge, UK

Nicola Di Mauro
nicola.dimauro@uniba.it
University of Bari, Italy

Alejandro Molina
alejandro.molina@tu-dortmund.de
TU Dortmund, Germany

Kristian Kersting
kersting@cs.tu-darmstadt.de
TU Darmstadt, Germany

Floriana Esposito
floriana.esposito@uniba.it
University of Bari, Italy

Abstract

Sum-Product Networks (SPNs) are a deep probabilistic architecture that up to now has been successfully employed for tractable inference. Here, we extend their scope towards unsupervised representation learning: we encode samples into continuous and categorical embeddings and show that they can also be decoded back into the original input space by leveraging MPE inference. We characterize when this Sum-Product Autoencoding (SPAE) leads to equivalent reconstructions and extend it towards dealing with missing embedding information. Our experimental results on several multi-label classification problems demonstrate that SPAE is competitive with state-of-the-art autoencoder architectures, even if the SPNs were never trained to reconstruct their inputs.

Introduction

Recent years have seen a significant interest in learning tractable representations facilitating *exact* probabilistic inference for a range of queries in *polynomial* time (Lowd and Domingos 2008; Choi and Darwiche 2017). Being instances of arithmetic circuits (Darwiche 2003), Sum-Product Networks (SPNs) were among the first *learnable* representations of these kind (Poon and Domingos 2011). They are deep probabilistic models that, by decomposing a distribution into a hierarchy of mixtures (sums) and factorizations (products), have achieved impressive performances in various AI tasks, such as computer vision (Gens and Domingos 2012; Amer and Todorovic 2016), speech (Zohrer, Peharz, and Pernkopf 2015), natural language processing (Cheng et al. 2014; Molina, Natarajan, and Kersting 2017), and robotics (Pronobis, Riccio, and Rao 2017). So far, however, SPNs have mainly been used as "black box" inference machines: only their output—the answer to a probabilistic query—has been used in the tasks at hand.

We here extend the scope of SPNs towards Representation Learning (Bengio, Courville, and Vincent 2012). We leverage their learned inner representations to uncover explanatory factors in the data and use these in predictive tasks. Indeed, other probabilistic models have traditionally been used in similar ways, such as Restricted Boltzmann Machines (RBMs): After unsupervised training, a feature representation can be extracted and fed into a classifier (Coates, Lee, and Ng 2011; Marlin et al. 2010), or used to initialize another neural architecture (Hinton and Salakhutdinov 2006). A particular advantage of such generative encoding-decoding schemes is that we can work with a potentially easier-to-predict target space, e.g., for *structured prediction*. Several variants of non-probabilistic autoencoders exist (Vincent et al. 2010; Rifai et al. 2011), tackling this problem by jointly learning encoding and decoding functions via optimizing the closeness of their decoded reconstructions.

Specifically, we demonstrate that SPNs are naturally well suited for Representation Learning (RL), compared to the aforementioned models, since they offer: i.) *exactly* answering a wider range of queries in a tractable way, e.g. marginals (Poon and Domingos 2011; Bekker et al. 2015), enabling natural inference formulations for RL; ii.) a *recursive*, *hierarchical* and *part-based* definition, allowing the extraction of rich and compositional representations well suited for image and other natural data; and iii.) *time and effort saved* in hyperparameter tuning since both their structure and weights can be learned in a "cheap" way (Gens and Domingos 2013). This makes SPNs an excellent choice for *encoding-decoding* tasks: although learned in a generative way and without training them to reconstruct their inputs, SPNs extract surprisingly good representations and allow to decode them effectively.

To encode samples, we either use the SPNs' latent variable semantics or treat them as neural networks, adopting the natural inference scenario SPNs offer—computing a Most Probable Explanation (MPE) (Poon and Domingos 2011; Peharz et al. 2017)—dealing with both categorical and continuous embeddings. We characterize conditions when the proposed decoding schemes for SPNs deliver the same sample reconstructions. Moreover, we leverage MPE inference to cope with partial embeddings, i.e. comprising missing values—a difficult scenario even for probabilistic autoencoders. The benefits of the resulting *Sum-Product Autoencoding* (SPAE) routines are demonstrated by extensive experiments on Multi-Label Classification (MLC) tasks. SPAE is competitive to RBMs, deep probabilistic (Germain et al. 2015) and non-probabilistic autoencoders like contractive (Rifai et al. 2011), denoising (Vincent et al. 2010) and stacked autoencoders tailored for label embeddings (Wicker, Tyukin, and Kramer 2016), either embedding original features, the target space, or both.

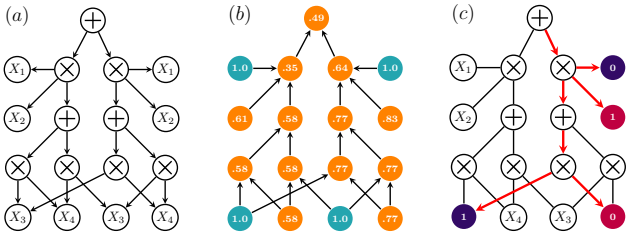

Figure 1: Visualizing MaxProdMPE. In (a), a complete and decomposable SPN S (sum weights are assumed uniform, omitted for clarity). In (b) we propagate node activations bottom-up (marginalized nodes in blue) when turning S into an MPN M to solve $\operatorname{argmax}_{\mathbf{q}\sim\mathbf{Q}} S(\mathbf{q}, X_2 = 1, X_4 = 0)$, $\mathbf{Q} = \{X_1, X_3\}$. In (c), the induced tree θ in the top-down traversal of M is shown in red. The assignment for RVs $\mathbf{Q}$ (resp. $\mathbf{O} = \{X_2, X_4\}$) labels the violet (resp. purple) leaves.

Sum-Product Networks

We denote random variables (RVs) by upper-case letters, e.g. X, Y, and their values as the corresponding lower-case letters, e.g. $x \sim X$. Similarly, we denote sets of RVs as $\mathbf{X}$, $\mathbf{Y}$ and their combined values as $\mathbf{x}$, $\mathbf{y}$. For $\mathbf{Y} \subseteq \mathbf{X}$ and a sample $\mathbf{x}$, we denote with $\mathbf{x}_{|\mathbf{Y}}$ the restriction of $\mathbf{x}$ to $\mathbf{Y}$.

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

When n is a *inner* node of S, i.e. $\mathsf{ch}(n) \neq \emptyset$, it is either a *sum* or *product* node. If n is a sum node, it computes a nonnegatively weighted sum over its children, i.e. $S_n = \sum_{c\in\mathsf{ch}(c)} w_{nc} S_c$. When n is a product node, it computes a product over its children, i.e. $S_n = \prod_{c\in\mathsf{ch}(c)} S_c$. The scope of inner node n is recursively defined as $\mathsf{sc}(n) = \bigcup_{c\in\mathsf{ch}(n)} \mathsf{sc}(c)$.

The *distribution* represented by SPN S is defined as the normalized output of its root. This distribution clearly depends both on the DAG structure of S and on its parameterization—the set of all sum-weights and any parameters of the leaf distributions—which we denote as $\mathbf{w}$.

Let $\mathbf{S}^{\oplus}$ (resp. $\mathbf{S}^{\otimes}$) be the set of all sum (resp. product) nodes in S. In order to allow for efficient inference, an SPN S is required to be *complete*, i.e. it holds that $\forall n \in \mathbf{S}^{\oplus}, \forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$, and *decomposable*, i.e. it holds that $\forall n \in \mathbf{S}^{\otimes}, \forall c_1, c_2 \in \mathsf{ch}(n), c_1 \neq c_2 : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$ (Poon and Domingos 2011). Furthermore, w.l.o.g. we assume that the SPNs considered here are *locally normalized* (Peharz et al. 2015), i.e. $\forall n \in \mathbf{S}^{\oplus}, \sum_{c\in\mathsf{ch}(n)} w_{nc} = 1$.

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

Inference To *evaluate the probability* of a sample $\mathbf{x} \sim \mathbf{X}$, we evaluate $\phi_n(\mathbf{x}_{|\mathsf{sc}(n)})$ for each leaf n. Subsequently, in a bottom-up pass probability $S_n(\mathbf{x}_{|\mathsf{sc}(n)})$ (short-hand $S_n(\mathbf{x})$) is computed for *all* $n \in \mathbf{S}$, till the root.

Marginals of the SPN distribution can be computed in time linear in the network size (Poon and Domingos 2011; Peharz et al. 2015). To evaluate the probability of $\mathbf{o} \sim \mathbf{O}$, where $\mathbf{O} \subset \mathbf{X}$, we simply evaluate $\phi(\mathbf{o})$ for the leaves (marginalizing any RVs not in $\mathbf{O}$) and proceed bottom-up to inner nodes in the usual way. Efficient marginalization is a remarkable key advantage of SPNs and other ACs, as this is a core routine for most inference scenarios.

On the other hand, the scenario of *Most Probable Explanation* (MPE) inference is generally NP-hard in SPNs (Peharz et al. 2017). Given two sets of RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$ and $\mathbf{Q} \cap \mathbf{O} = \emptyset$, and evidence $\mathbf{o} \sim \mathbf{O}$, inferring an MPE assignment for $\mathbf{Q}$ is defined as finding:

$$\mathbf{q}^* = \operatorname{argmax}_{\mathbf{q}} S(\mathbf{q} \,|\, \mathbf{o}) = \operatorname{argmax}_{\mathbf{q}} S(\mathbf{q}, \mathbf{o}). \qquad (1)$$

Eq. 1, however, can be solved efficiently and exactly in *selective* SPNs (Choi and Darwiche 2017; Peharz et al. 2017), i.e. SPNs where it holds that $\forall \mathbf{x}^i \in \mathbf{X}, \forall n \in \mathbf{S}^{\oplus} : |\{c \,|\, c \in \mathsf{ch}(n) : S_c(\mathbf{x}^i) > 0\}| \leq 1$. In selective SPNs, MPE is solved via the MaxProdMPE algorithm (Chan and Darwiche 2006; Peharz et al. 2017; Conaty, Mauá, and de Campos 2017).

For a given SPN S, MaxProdMPE first constructs the corresponding Max-Product Network (MPN) M by replacing each sum node with a *max node* $\max_{c\in\mathsf{ch}(n)} w_{nc} M_c(\mathbf{x})$ and each leaf distribution ϕ_n with a maximizing distribution ϕ_n^M (Peharz et al. 2017).[2] In the first bottom-up step, one computes $M(\mathbf{x}_{|\mathbf{O}})$ (Fig. 1b). In step two, via Viterbi-style backtracking, the MPE solution to Eq. 1 can be retrieved. Starting from the root, one follows all children of product nodes and one maximizing child of max nodes. From this top-down pass one determines an *induced tree* θ (Fig. 1c), a tree-shaped sub-network whose leaves scopes define a partition over $\mathbf{X}$. By taking the argmax over the leaves of the induced tree, one retrieves an MPE solution. For general SPNs, the solution delivered by MaxProdMPE is not exact, but is commonly used as an approximation to the MPE problem (Poon and Domingos 2011; Peharz et al. 2017).

Learning The semantics of SPNs enables simple and yet surprisingly effective algorithms to estimate both the network structure and parameters (Dennis and Ventura 2012; Peharz, Geiger, and Pernkopf 2013). Many variants (Rooshenas and Lowd 2014; Vergari, Di Mauro, and Esposito 2015; Melibari et al. 2016) build upon one of the most prominent algorithms, LearnSPN, a greedy top-down learner introduced by Gens and Domingos (2013). LearnSPN acts as a *recursive data crawler*, by decomposing a given data matrix along its rows (i.e. samples), generating sum nodes and estimating their weights, and its columns (i.e. RVs), generating products. This way of learning SPNs can also be interpreted as running a hierarchical feature extractor. Our proposed SPAE routines are the first approach to exploit this interpretation.

[2]The previously introduced notation for S, $\mathbf{S}$, $\mathbf{S}^{\oplus}$, etc. carries over as M, $\mathbf{M}$, $\mathbf{M}^{\mathsf{max}}$.

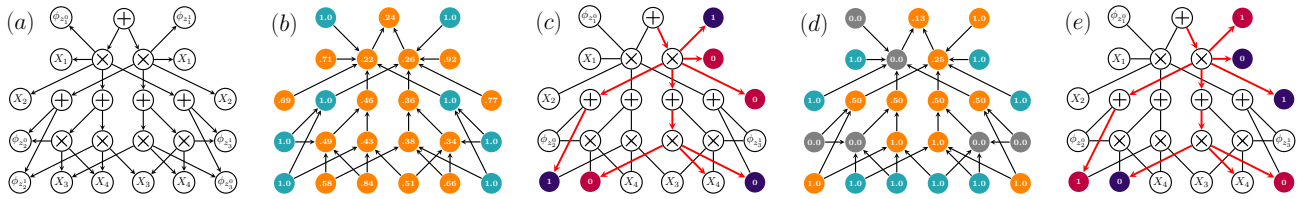

Figure 2: Encoding and decoding via LVs. (a) The SPN $\overline{S}$ augmenting the one in Fig. 1a (all sum weights assumed uniform). In (b), to encode $\mathbf{x}$=$(1, 0, 0)$, LVs in $\overline{M}$ are marginalized out (all indicators output 1, in blue). In (c) the tree θ' is induced top-down, selecting the query LV indicators (violet), computing $\mathbf{z}$=$(1, 1, 0)$. To decode $\mathbf{z}$, corresponding LV indicator activations are propagated bottom-up in (d). In (e), non-zero activations force the *(re)*construction of the tree θ'', leading to $\tilde{\mathbf{x}}$=$(1, 1, 0)$.

Sum-Product Autoencoding

We now extend the use of an SPN S, trained generatively to estimate $p(\mathbf{X})$ over some data set $\{\mathbf{x}^i \sim \mathbf{X}\}_i$, towards Representation Learning. More precisely, we are interested in *encoding* a sample $\mathbf{x}^i$ as an *embedding* $\mathbf{e}^i$ in a new d-dimensional space $\mathbf{E}_\mathbf{X}$ through a function f provided by the structure and parameters of S:

$$\text{(encoding)} \qquad \mathbf{e}^i = f_S(\mathbf{x}^i)\ .$$

For *decoding*, on the other hand, we seek for an approximate inverse function $g\colon \mathbf{E}_\mathbf{X} \to \mathbf{X}$ such that

$$\text{(decoding)} \qquad g_S(\mathbf{e}^i) = \tilde{\mathbf{x}}^i \approx \mathbf{x}^i,$$

is its reconstruction in the original feature space $\mathbf{X}$; here the embedding $\mathbf{e}^i$ can be the result of an encoding process f_S *or* the output of a predictive model whose target space is $\mathbf{E}_\mathbf{X}$. To this end, we will now develop two *encoding-decoding* schemes for SPAE, namely a probabilistic one—leveraging the latent variable semantics in SPNs— and a deterministic one, by interpreting SPNs as neural networks.

Probabilistic Perspective: CAT embeddings

By noting that each sum node $n \in \mathbf{S}^\oplus$ essentially represents a *mixture model* over its child distributions $\{S_c\}_{c\in\mathsf{ch}(n)}$, SPNs naturally lend themselves towards an interpretation as *hierarchical latent variable* (LV) models. In particular, we can associate to each $n \in \mathbf{S}^\oplus$ a marginalized LV Z_n assuming values in $\{0, \ldots, |\mathsf{ch}(n)| - 1\}$. The LV interpretation of SPNs helps to connect SPNs with existing probabilistic models (Zhao, Melibari, and Poupart 2015) and justifies approaches developed for LV models, such as the expectation-maximization algorithm (Peharz et al. 2017).

Let $\mathbf{Z}_S$ be the set of the LVs associated to all the sum nodes in an SPN S. By explicitly incorporating $\mathbf{Z}_S$ in our SPN representation by means of *augmented SPNs* (Peharz et al. 2017), we can reason about the joint distribution $\mathbf{V} = (\mathbf{X}, \mathbf{Z}_S)$ while exploiting the usual inference machinery of SPNs. Given an SPN S over $\mathbf{X}$, the augmented SPN $\overline{S}$ over $\mathbf{V}$ introduces indicator leaves $\phi_{z_{ik}} = \mathbb{1}\{Z_i = z_{ik}\}$ for each $Z_i \in \mathbf{Z}_S$ and value $z_{ik} \sim Z_i$. These indicators are interconnected via product nodes – so-called *links* – between sum nodes and their children, and essentially switch on/off particular children, making the the latent selection process of sum nodes explicit. In order to make $\overline{S}$ a complete and decomposable SPN over $\mathbf{V}$, additional structural changes may be necessary. In particular, for any $n \in \mathbf{S}^\oplus$ and $c \in \mathsf{ch}(n)$, let p_{nc} be the corresponding link, which is a child of n and a joint parent of c and $\phi_{z_{nc}}$. If n has a sum node ancestor a having a child *link node* m that is not ancestor of n, then a so-called *twin sum node* $\overline{n}$ is introduced which is a parent of all indicators $\{\phi_{z_{nk}}\}_k$ and a child of m. By this construction, the augmented SPN is a complete and decomposable SPN over $\mathbf{V}$. For details, see the supplementary material (Vergari et al. 2017) and (Peharz et al. 2017).

For an augmented SPN $\overline{S}$ it holds that $S(\mathbf{X}) = \overline{S}(\mathbf{X})$, i.e. when all $\mathbf{Z}_S$ are marginalized, we retrieve the distribution of S. Moreover, augmented SPNs are always selective, thus MPE can be solved exactly and efficiently (see previous Section). This property will be crucial in our SPAE for Representation Learning.

CAT embedding process The LV semantics of SPNs offers a natural way to formulate the encoding problem: embed each sample $\mathbf{x}^i$ into the space induced by their associated LVs, i.e. $\mathbf{E}_\mathbf{X} = \mathbf{Z}_S$ and $d = |\mathbf{S}^\oplus|$. Thus, we define f_S as:

$$f_S(\mathbf{x}^i) = f_{\mathsf{MPE}}(\mathbf{x}^i) \triangleq \tilde{\mathbf{z}}^i = \operatorname{argmax}_{\mathbf{z}^i} S(\mathbf{z}^i \,|\, \mathbf{x}^i), \qquad (2)$$

i.e. $\mathbf{x}^i$ is encoded as the *categorical* vector $\tilde{\mathbf{z}}^i$ comprising the most probable state for the LVs in S. Analogously, the decoding of $\tilde{\mathbf{z}}^i$ through g_S can be defined as:

$$g_S(\tilde{\mathbf{z}}^i) = g_{\mathsf{MPE}}(\tilde{\mathbf{z}}^i) \triangleq \tilde{\mathbf{x}}^i = \operatorname{argmax}_{\mathbf{x}^i} S(\mathbf{x}^i \,|\, \tilde{\mathbf{z}}^i). \qquad (3)$$

To solve both (2) and (3), we perform MPE inference on the joint pdf of $\mathbf{V} = (\mathbf{X}, \mathbf{Z}_S)$, by running MaxProdMPE on the augmented SPN $\overline{S}$. Each application of MaxProdMPE involves a bottom-up and a backtracking pass over the MPN $\overline{M}$ corresponding to $\overline{S}$, yielding in total 4 passes over $\overline{M}$, as illustrated in Fig 2.

For the encoding step, the bottom-up pass yields $\overline{M}(\mathbf{x})$, where LVs are marginalized (Fig 2b). A tree θ' is induced in the top-down pass (Fig 2c), and $\tilde{\mathbf{z}}^i$ is obtained by collecting the states corresponding to the indicators $\phi_{z_{jk}}$ in θ'.

For the decoding stage, the bottom-up pass computes $\overline{M}(\tilde{\mathbf{z}}^i)$ (Fig. 2d), i.e. the LVs now assume the role of "observed data", while $\mathbf{X}$ is now unobserved. A tree θ'' is grown in the top-down pass (Fig 2e), and $\mathbf{x}^*$ is built from the MPE assignments computed at the leaves contained in θ''.

Although operating on augmented SPNs provides a principled way to derive encoding and decoding routines, it also poses some practical issues. First, even if $\overline{S}$—being selective over $\mathbf{V}$—provides exact solutions to Eq. 2 and 3, the

embedding $\mathbf{z}^*$ typically contains LVs which do not contain any information about the data. This stems from the fact that a particular LV Z_n might be rendered independent from $\mathbf{X}$, given certain context of all LVs "above" Z_n (see (Peharz et al. 2017) for details). Second, the size of $\overline{M}$ can grow up to $|M|^2$ and explicitly constructing it would be wasteful.

Therefore, we will conveniently compute f_{MPE} and g_{MPE} operating directly on M, and not on $\overline{M}$, i.e. we simulate MaxProdMPE in the original MPN. The result is a shorter and context-dependent embedding vector, for which, however, the decoding stage should not rely on LVs which have been ignored in the encoding stage. This is formally established by the following proposition, showing that the two induced trees θ' and θ'' are identical.

Proposition 1. *Let S be an SPN over $\mathbf{X}$, M its corresponding MPN, and $\overline{M}$ its augmented version over $\mathbf{V} = (\mathbf{X}, \mathbf{Z}_S)$. Let θ' (resp. θ'') be the tree built by computing $f_{\mathsf{MPE}}(\mathbf{x}^i)$ (resp. $g_{\mathsf{MPE}}(f_{\mathsf{MPE}}(\mathbf{x}^i))$) on $\overline{M}$, given a sample $\mathbf{x}^i \sim \mathbf{X}$. Then, it holds that $\theta' = \theta''$.*

Prop. 1 also highlights a high level interpretation of encoding and decoding samples through $\mathbf{Z}_S$: $\tilde{\mathbf{z}}^i$ encodes $\mathbf{x}^i$ into a *tree representation*, decoding means to "materialize" the *same tree* back, and then to apply MPE inference over its leaves, yielding $\tilde{\mathbf{x}}^i$. In a sense, SPAE works by demanding encoding and decoding to subsets of leaves, using induced trees to select them for each sample considered.

We now introduce f_{CAT} and g_{CAT} as encoding and decoding routines for CATegorical embeddings, operating on M and producing reconstructions equivalent to f_{MPE} and g_{MPE}. To compute $f_{\mathsf{CAT}}(\mathbf{x}^i)$ one evaluates $M(\mathbf{x}^i)$ in a bottom-up pass and then grows a tree path θ while collecting the states:

$$z_j^i = \mathrm{argmax}_{k \in \{0,\dots,|\mathsf{ch}(n_j)|\}} \, w_{n_j c_k} M_{c_k}(\mathbf{x}^i), \qquad (4)$$

for each $Z_j \in \mathbf{Z}_S^\theta$, where $\mathbf{Z}_S^\theta$ is the set of LVs associated only to the max nodes traversed in θ. On the other hand, we build $g_{\mathsf{CAT}}(\tilde{\mathbf{z}}^i)$ as the procedure that mimics *only* the top-down pass of MaxProdMPE by materializing the tree path θ encoded in $\tilde{\mathbf{z}}^i$. Growing θ from the root, if the currently traversed node $n_j \in M$ is a max node, the embedding value z_j^i, corresponding to the LV Z_j associated to n_j, determines the child branch to follow; if it is a product node, all its child branches are added to θ, as usual. As before, performing MPE inference on the leaves of θ retrieves the decoded sample $\tilde{\mathbf{x}}^i = g_{\mathsf{CAT}}(\tilde{\mathbf{z}}^i)$.

Proposition 2. *Given a sample $\mathbf{x}^i \sim \mathbf{X}$, it hold that $g_{\mathsf{MPE}}(f_{\mathsf{MPE}}(\mathbf{x}^i)) = g_{\mathsf{CAT}}(f_{\mathsf{CAT}}(\mathbf{x}^i))$.*

By construction, embeddings via f_{CAT} equals the ones built by f_{MPE} up to the LVs in $\mathbf{Z}_S^\theta$, all other LVs are undefined in $\tilde{\mathbf{z}}^i$. Consequently, CAT embeddings are very sparse. While the undefined LVs can be estimated in several ways, their reconstructions through g_{CAT} would be unaffected[3].

[3]In the supplemental material (Vergari et al. 2017) we investigate also *dense* CAT embeddings, obtained by estimating the undefined LVs in $\tilde{\mathbf{z}}^i$ by applying Eq. 4 to them as well. We demonstrate their equivalent reconstructions and evaluate them empirically

Deterministic perspective: ACT embeddings

SPNs can also be interpreted as deep NNs whose *constrained topology* determines sparse connections and in which nodes are neurons *labeled* by the scope function sc, enabling a *direct encoding* of the input (Bengio, Courville, and Vincent 2012) and retaining a fully probabilistic semantics (Vergari, Di Mauro, and Esposito 2016). Indeed, like RBMs, but differently from deep estimators like NADEs and MADEs (Germain et al. 2015), each neuron activation, i.e. $S_n(\mathbf{x})$, is a valid probability value by definition. Following this interpretation, SPNs can also be used as feature extractors by employing neuron ACTivations to build embeddings, as it is common practice for neural networks and autoencoders (Rifai et al. 2011; Marlin et al. 2010).

ACT embedding process Let $\mathbf{N} = \{n_j\}_{j=1}^d \subseteq \mathbf{S}$ be a set of nodes in S, collected by a certain criterion. A sample $\mathbf{x}^i$ is encoded into a d-dimensional *continuous* embedding $f_S(\mathbf{x}^i) = \mathbf{e}^i \in \mathbf{E}_\mathbf{X} \subseteq \mathbb{R}^d$ by collecting the activations of nodes in $\mathbf{N}$, i.e. $e_j^i = S_{n_j}(\mathbf{x}^i)$. Let $f_{\mathsf{ACT}}(\mathbf{x}^i) \triangleq \mathbf{e}_M^i$ be the ACT embedding built from the inner nodes[4] of the MPN M built from S, i.e. $\mathbf{N} = \mathbf{M}^{\max} \cup \mathbf{M}^{\otimes}$. We can note how ACT embeddings implicitly encode an induced tree: node activations $\mathbf{e}_M^i$ are sufficient to determine which max node child branch to follow, according to Eq. 4. Therefore, we can build a decoder g_{ACT} that mimics the top-down pass of MaxProdMPE. Specifically, θ can be grown by choosing the child c_k of a max node n_j such that the value $w_{n_j c_k} e_{c_k}^i$ is the max among the k siblings, and hence equal to $e_{n_j}^i$. As for g_{CAT}, all product node children are traversed and $\tilde{\mathbf{x}}^i$ is built by collecting MPE assignments at the leaves of θ.

Proposition 3. *Given a sample $\mathbf{x}^i \sim \mathbf{X}$, it holds that, $g_{\mathsf{ACT}}(f_{\mathsf{ACT}}(\mathbf{x}^i)) = g_{\mathsf{CAT}}(f_{\mathsf{CAT}}(\mathbf{x}^i))$.*

While Props. 2 and 3 ensure that CAT and ACT SPAE routines provide the same decoding to one sample encoded by them, the information content of each embedding is clearly different. Consequently, CAT and ACT SPAE routines will yield different performances when employed for predictive tasks. Lastly, instead of selecting all inner nodes in M we could have employed only a subset of them, e.g. selecting only nodes near the root, seeking high level representations (Vergari, Di Mauro, and Esposito 2016). However, we would need to accommodate g_{CAT} and g_{ACT} in order to deal with the "missing" nodes. We deal with this next.

Dealing with Partial Embeddings

Up to now we have considered only *fully decodable embeddings*, i.e. embeddings comprising all the information required to materialize a *complete* and well-formed tree necessary to decode $\mathbf{e}$ into $\tilde{\mathbf{x}}$. In some real cases, however, only incomplete or *partial* embeddings are available: some values e_j are corrupted, invalid or just missing. E.g., consider data compression, one may want to store only the most relevant values of an embedding and discard the rest. Since SPAE

[4]In the supplemental material (Vergari et al. 2017) we investigate ACT embeddings using leaf activations as well.

routines are built on probabilistic models, they offer a natural and efficient way to deal with such cases. As follows, we denote with $e_j \notin \mathbf{e}$ the embedding j-th value missing.

In theory, for ACT embeddings, a partial embedding $\mathbf{e}^i$ would still be decodable if each missing value could be reconstructed by the corresponding node non-missing children activations, i.e. for each $M_n(\mathbf{x}^i) \notin \mathbf{e}^i$ it holds that $\forall c \in \mathsf{ch}(n)\colon e^i_c = M_c(\mathbf{x}) \in \mathbf{e}^i$. For the general case, we propose to impute CAT and ACT missing values by performing MPE inference to estimate them.

In practice, if for an ACT (resp. CAT) embedding the component $e^i_j \notin \mathbf{e}^i$ (resp. $z^i_j \notin \mathbf{z}^i$) corresponds to a node n_j activation (resp. LV Z_j state), then it can be imputed by employing MaxProdMPE on the sub-network M_{n_j}. In a sense, this generalizes to inner nodes and their scopes what already happens at leaf level. Lastly, consider that to impute k missing values from one embedding, instead of running MaxProdMPE k times, we can run the bottom-up pass just once for the whole M and then reuse computations while traversing top-down for the k nodes. In the experimental section, we empirically evaluate the effectiveness and resilience of SPAE on embedding values missing at random.

Discussion of CAT and ACT Embeddings

Before moving on to our empirical evaluation let us provide some intuition of CAT and ACT embeddings.

CAT embeddings from an SPN S are points in the latent space induced by $\mathbf{Z}_S$. They are *compact* and *linearized* representations of the tree paths in S, also called the complete sub-circuits resp. *induced tree components* of S (Zhao, Melibari, and Poupart 2015; Zhao and Poupart 2016). Directly encoding only leaf subsets would not have allowed the flexibility and robustness to deal with partial embeddings. Evaluating the visible effects of conditioning on subsets of CAT embedding components is harder than dealing with the continuous case (Bengio, Courville, and Vincent 2012), since we cannot easily interpolate among categorical vectors nor we explicitly model dependencies in $\mathbf{Z}_S$ (Peharz et al. 2017). However, one can still interpret the semantics of learned CAT embeddings by visualizing the latent factors of variations encoded in $\mathbf{Z}_S$ through the *clusters* of samples sharing the same representations. For an SPN learned on a binarized version of MNIST[5], Fig. 3 depicts random samples sharing the same CAT encoding. Even though samples may belong to different digit classes, they clearly share stylistic aspects like orientation and stroke.

ACT embeddings, on the other hand, are points in the space induced by a collection of proper *marginal pdfs*. From their neural interpretation, SPN nodes are *part-based filters* operating over sub-spaces induced by the node scopes. Sum nodes act as averaging filters, and product nodes compose other non-overlapping filters. From the perspective of LearnSPN-like structure learners, each filter is learned to capture a different aspect of the sub-population and subspace of the data it is trained on. Thereby, each SPN comprises a hierarchy of filters at different *levels of abstraction*.

[5]See the supplemental material (Vergari et al. 2017) for details, code and learning settings for all experiments

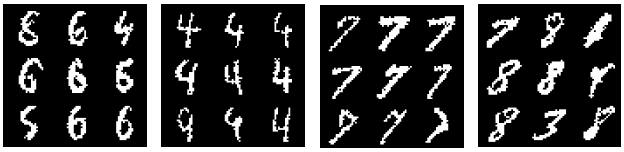

Figure 3: Visualizing the factors of variations encoded by CAT embeddings: 4 clusters of 9 binarized MNIST samples sharing the same encoding. Each cluster clearly shows some latent style pattern like straight, curved or slanted strokes.

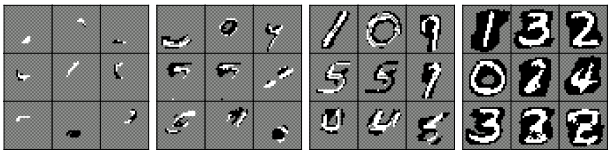

Figure 4: Visualizing SPN activations learned on binarized MNIST: 4 clusters of node filters with similar scope lengths, i.e. $|\mathsf{sc}(n)|$. The *compositionality* of the learned representations is evident through the different levels of the hierarchy: the longer the scope the higher the level of abstraction. The scope information alone (out-of-scope pixels are depicted in a checkerboard pattern) may be able to convey a meaningful representation of "object parts", e.g. the 'O' shapes.

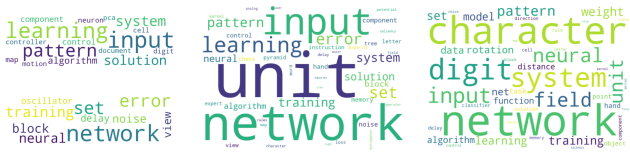

Figure 5: Visualizing SPN activations on the NIPS corpus at different levels of abstraction via wordclouds: the words within a node's scope with sizes proportional to their MPE relevance counts. Compositionality is clearly visible in the center topic obtained by composing the top and bottom subtopics on the left—the one in the center corresponds to a product node over the two children on the left. When different nodes share a scope (center and right), they filter for alternative topic meanings. (Best viewed in color)

This innate compositionality enhances the interpretability of the embedded representations. Filters from neurons in NNs can be visualized back in the original input space as the data samples producing their largest activation by solving an inverse problem by costly iterative optimization (Erhan et al. 2009). For a node n in an SPN S over $\mathbf{X}$ this equals to compute $\mathbf{x}^*_{|\mathsf{sc}(n)} = \mathrm{argmax}_{\mathbf{x}\sim\mathbf{X}} S_n(\mathbf{x})$, which can be solved by MaxProdMPE, requiring only two evaluations of S. Fig. 4 shows some of these approximated MPE solutions into the pixel space as the filters learned by the SPN trained on the aforementioned version of MNIST. Note how differently complex local patterns emerge when they are sorted by their hierarchy level, e.g. from small blobs to shape contours and finally full digits. Similarly, for a Poisson SPN (Molina, Natarajan, and Kersting 2017) learned on the bag-of-words NIPS text datasets, the hierarchy over MPE

filters can be easily interpreted as structure over topics visualizable through word counts, Fig. 5.

As already stated, CAT and ACT embeddings act differently when plugged in predictive tasks. Indeed, when employed as features for a predictor (its input) we expect ACT embeddings to perform better than CAT ones due to their greater information content. E.g. the digits in a cluster in Fig. 3 are indistinguishable by their CAT embeddings while their ACT representations differ. Conversely, when employed to encode target RVs (a predictor's output) we forsee *classification* for the CAT case to be easier than *regression* with ACT embeddings, since the latter greater variability in values and the simpler prediction task due to the sparsity of the former. However, mispredicted ACT components may still be able to grow a complete tree path, differently from CAT ones. All in all, how much competitive our SPAE routines are has to be verified empirically.

Finally, consider that, for both CAT and ACT embeddings, the choice of the embedding size d is data-driven: it depends on the SPN structure learned from data. Hence, d does not need to be fixed or tuned by hand as for other neural models. Nevertheless, to "control" it one can either regularize structure learning or not consider some nodes in the SPN, i.e., dealing with partial embeddings.

Experimental Evaluation

To evaluate SPAE, we focus on Multi-Label Classification (MLC): predict the target labels represented as binary arrays $\mathbf{y}^i \sim \mathbf{Y}$ associated to sample $\mathbf{x}^i \sim \mathbf{X}$. MLC is challenging testbed for autoencoders, since it not only asks for encoding and decoding in the feature space, but also in the label space. Especially, our aim here is to investigate the following questions: **(Q1)** How close to perfect reconstructions can SPAE routines get on networks learned from real data? **(Q2)** How meaningful and useful are these representations when employed for complex predictive tasks? **(Q3)** How resilient to missing embedding values are SPAE decoding schemes?

Experimental Protocol Since there is no unique way to assess performances on structured output, we report the jaccard (JAC) and exact match (EXA) scores, as metrics highly employed in the MLC literature and whose maximization equals to focus on different sets of probabilistic dependencies (Dembczyński et al. 2012)[6]. For all experiments we use 10 standard MLC benchmarks: Arts, Business, Cal, Emotions, Flags, Health, Human, Plant, Scene and Yeast, preprocessed as binary data in 5 folds as in (Di Mauro, Vergari, and Esposito 2016). We learn both the structure and weights of all models on $\mathbf{X}$ and $\mathbf{Y}$ separately for each fold. For SPNs we employ LearnSPN-b (Vergari, Di Mauro, and Esposito 2015), a variant of LearnSPN learning deeper structures.

(Q1) Reconstruction performance In order to assess the validity of the proposed encoding/decoding processes *per-se*, here, for each SPN learned on $\mathbf{X}$ (resp. $\mathbf{Y}$) we measure the average fold JAC and EXA scores between samples $\mathbf{x}^i$ and their reconstructions $g(f(\mathbf{x}^i))$ (resp. $\mathbf{y}^i$ and $g(f(\mathbf{y}^i))$). Recall that for this task CAT and ACT routines provide the same decoding (Prop. 3). While SPNs are not trained to reconstruct their inputs, monitoring these scores helps understanding if structure learning autonomously provided a structure and parametrization favoring autoencoding. Reconstructions for test samples over $\mathbf{X}$ score 69.70 JAC and 16.29 EXA on average considerably increasing to 84.86 resp. 66.50 over $\mathbf{Y}$. Detailed results are reported in the supplemental material. This suggests that SPNs have been able to model label dependencies easier, growing leaves to better discriminate different states on smaller data.

(Q2) MLC prediction performance. In the simplest (fully-)supervised case for MLC one would learn a predictor p, from the original feature space to the label one: $\mathbf{X} \overset{p}{\Rightarrow} \mathbf{Y}$. We define three more learning scenarios in which to employ the representations learned by SPNs: I) learn p on the features encoded by a model r: $(\mathbf{X} \xrightarrow{f_r} \mathbf{E_X}) \overset{p}{\Rightarrow} \mathbf{Y}$; II) learn p to predict the embedded label space $\mathbf{E_Y}$ encoded by a model t, then use t to decode them back to $\mathbf{Y}$: $(\mathbf{X} \overset{p}{\Rightarrow} (\mathbf{Y} \xrightarrow{f_t} \mathbf{E_Y})) \xrightarrow{g_t} \mathbf{Y}$; III) combine the previous two scenarios, encoding both feature and label spaces: $((\mathbf{X} \xrightarrow{f_r} \mathbf{E_X}) \overset{p}{\Rightarrow} (\mathbf{Y} \xrightarrow{f_t} \mathbf{E_Y})) \xrightarrow{g_t} \mathbf{Y}$. We employ for p only linear predictors to highlight the ability of the learned embeddings to *disentangle* the represented spaces. For scenario I) p is a L_2-regularized logistic regressor (LR) while for II) and III) it may be either a LR or a ridge regressor (RR) if it is predicting CAT or ACT embeddings.

As a *proxy* measure to the meaningfulness of the learned representations, we consider their JAC and EXA prediction scores and compare them against the natural baseline of LR being applied to raw input and output, $\mathbf{X} \overset{\mathsf{LR}}{\Rightarrow} \mathbf{Y}$. We also employ a standard *fully-supervised* predictor for structured output, a Structured SVM employing CRFs to perform tractable inference by Chow-Liu trees on the label space ($\mathsf{CRF_{SSVM}}$) (Finley and Joachims 2008). Concerning r and t models as encoder and decoder competitors, we employ generative models such as: 1) RBMs, whose conditional probabilities have been largely used as features (Larochelle and Bengio 2008; Marlin et al. 2010), with 500, 1000 and 5000 hidden units (h); 2) deep probabilistic autoencoders as MADEs (Germain et al. 2015) concatenating embeddings from 3 layers of 500 or 1000 (resp. 200 or 500) hidden units for $\mathbf{X}$ (resp. $\mathbf{Y}$); non-probabilistic autoencoders (AEs) with 3 layer deep encoder and decoder networks, compressing the input by a factor $\gamma \in \{0.7, 0.8, 0.9\}$ such as 3) stacked AEs (SAEs) tailored for MLC (Wicker, Tyukin, and Kramer 2016); 4) contractive AEs (CAEs) (Rifai et al. 2011) and 5) denoising AEs (DAEs) (Vincent et al. 2010). Note that for all the aforementioned models we had to perform an extensive grid search first to determine their structure and then to tune their hyperparameters (see supplemental material).

Used configurations for each setting are reported in Tab. 1 together with the scores aggregated over all 10 datasets as averaged relative improvement w.r.t. the LR baseline. The higher the improvement, the better. As one can see, SPAE is highly competitive to all other models in the three scenarios, for all scores, even when compared to the fully supervised and discriminative $\mathsf{CRF_{SSVM}}$. More in detail, sce-

[6] In the supplemental material we report also hamming scores.

Table 1: Average relative test set (percentage) improvements w.r.t LR on 10 benchmark datasets for multi-label classification. For each scenario, score, and decoding method, best results are bold. The extra two columns use kNN decoding, improved scores w.r.t normal decoding are denoted by ∘.

	$\mathbf{X} \overset{p}{\Rightarrow} \mathbf{Y}$	JAC	EXA		
baseline	p: LR	0.00	0.00		
	p: $\mathsf{CRF}_{\mathsf{SSVM}}$	+15.83	+103.90		
	$(\mathbf{X} \overset{f_r}{\longrightarrow} \mathbf{E_X}) \overset{\mathsf{LR}}{\Rightarrow} \mathbf{Y}$				
scenario I	r: $\mathsf{RBM}_{h\in\{500,1000,5000\}}$	+1.46	-1.62		
	r: $\mathsf{MADE}_{h\in\{500,1000\}}$	+2.57	+2.99		
	r: $\mathsf{CAE}_{\gamma\in\{0.7,0.8,0.9\}}$	-0.15	+4.13		
	r: $\mathsf{DAE}_{\gamma\in\{0.7,0.8,0.9\}}$	+0.70	+4.17		
	r: $\mathsf{SPAE}_{\mathsf{ACT}}$	**+3.54**	**+17.18**		
	r: $\mathsf{SPAE}_{\mathsf{CAT}}$	-11.90	-11.53		
				g_{kNN} JAC	g_{kNN} EXA
	$(\mathbf{X} \overset{p}{\Rightarrow} (\mathbf{Y} \overset{f_t}{\longrightarrow} \mathbf{E_Y})) \overset{g_t}{\longrightarrow} \mathbf{Y}$				
scenario II	t: $\mathsf{MADE}_{h\in\{200,500\}}, p$: RR	-30.42	-28.02	∘+14.57	∘+88.62
	t: $\mathsf{SAE}_{\gamma\in\{0.7,0.8,0.9\}}, p$: RR	+5.96	+95.78	-	-
	t: $\mathsf{CAE}_{\gamma\in\{0.7,0.8,0.9\}}, p$: RR	+7.60	+78.81	∘**+25.81**	+132.03
	t: $\mathsf{DAE}_{\gamma\in\{0.7,0.8,0.9\}}, p$: RR	+13.39	+102.22	∘+17.01	+71.20
	t: $\mathsf{SPAE}_{\mathsf{ACT}}, p$: RR	+15.19	+98.58	∘+21.94	∘+107.00
	t: $\mathsf{SPAE}_{\mathsf{CAT}}, p$: LR	**+24.07**	**+141.81**	+22.83	**+134.43**
	$((\mathbf{X} \overset{f_r}{\longrightarrow} \mathbf{E_X}) \overset{p}{\Rightarrow} (\mathbf{Y} \overset{f_t}{\longrightarrow} \mathbf{E_Y})) \overset{g_t}{\longrightarrow} \mathbf{Y}$				
scenario III	r, t: MADE, p: RR	-27.15	-25.14	+12.77	+85.78
	r, t: $\mathsf{CAE}_{\gamma\in\{0.7,0.8,0.9\}}, p$: RR	+5.21	+79.20	∘+24.32	∘+125.21
	r, t: $\mathsf{DAE}_{\gamma\in\{0.7,0.8,0.9\}}, p$: RR	+13.97	+98.25	∘+16.67	+76.17
	r: $\mathsf{SPAE}_{\mathsf{ACT}}, t$: $\mathsf{SPAE}_{\mathsf{ACT}}, p$: RR	+15.98	+106.65	∘+21.45	∘+109.7
	r: $\mathsf{SPAE}_{\mathsf{CAT}}, t$: $\mathsf{SPAE}_{\mathsf{CAT}}, p$: LR	+13.73	+107.05	+11.61	+102.08
	r: $\mathsf{SPAE}_{\mathsf{ACT}}, t$: $\mathsf{SPAE}_{\mathsf{CAT}}, p$: LR	**+25.47**	**+144.78**	**+23.47**	**+135.36**

nario I proved to be hard for many models, suggesting that the dependencies on $\mathbf{X}$ might not contribute much to predict $\mathbf{Y}$ (Dembczyński et al. 2012). ACT embeddings yield the largest improvements. As expected, by encoding $\mathbf{X}$ into a less informative space, CAT embeddings cannot beat the LR baseline. In scenario II, disentangling label dependencies is beneficial for all models. CAT embeddings largely outperforms all models, confirming sparsity over $\mathbf{E_Y}$ to help. Scenario III confirms the above observations, showing the advantages of combining both input and target embeddings, and both ACT and CAT achieving the overall best scores.

So, which aspect influenced the most the reported gains? Larger embedding sizes are not responsible, since our learned SPNs have sizes comparable to or smaller than RBM, and MADE (see supplemental material). It is also not due to a better model of the data distributions: MADE log-likelihoods are higher than SPN ones. We argue that it is due to the hierarchical part-based representations of SPAE embeddings: while performing its hierarchical co-clustering, LearnSPN discovers meaningful ways to discriminate among data at different granularities.

(Q2) kNN Decoding We isolate the effectiveness of our decoding schemes by employing a 5 nearest neighbor decoder ($g_t = g_{\mathsf{kNN}}$) over $\mathbf{E_Y}$ for scenarios II and III. Even here (last two columns of Tab. 1), SPAE is competitive. While g_{kNN} generally improves decoding for autoencoder models, tree paths encoded in CAT embeddings achieve better scores when decoded by g_{CAT}.This suggests that exploiting a form of sparse structured information—growing a tree top-down—is likely beneficial to decoding.

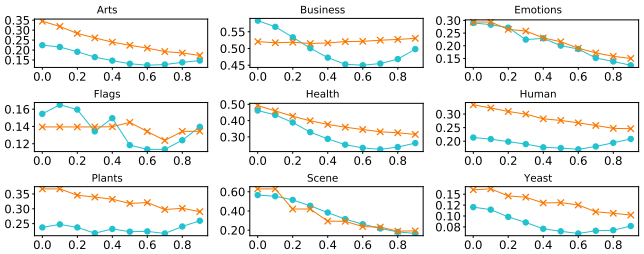

Figure 6: Partial embeddings: Average EXA scores (y axis) while imputing different percentages of missing random embedding components (x axis) by employing ACT (blue circles) or CAT (orange crosses). (Best viewed in color)

(Q3) Resilience to missing components Lastly, we evaluate the resilience to missing at random values when decoding CAT and ACT label embeddings in scenario II. That is, for each predicted label embedding we remove at random a percentage of values varying from 0 (full embedding) to 90%, by increments of 10%. Fig. 6 summarizes the EXA results for CAT and ACT decoding on 9 datasets (the EXA score on Cal is always 0). First, both routines are quite robust, degrading performances by less than 50% on average when half components are missing. Second, CAT scores decay slower than ACT and are generally better. Third, one can note the positive effect in predicting the label modes by MPE when almost all values are missing. In summary, all questions **(Q1)**-**(Q3)** can be answered affirmatively.

Conclusions

We investigated SPNs under a Representation Learning lens, comparing encoding and decoding schemes for categorical and continuous embeddings. An extensive set of experiments on Multi-Label Classification problems demonstrated that the resulting framework of Sum-Product Autoencoding (SPAE) indeed produces meaningful features and is competitive to state-of-the art autoencoders.

SPAE suggests several interesting avenues for future work: explore embeddings based on other instances of arithmetic circuits (Choi and Darwiche 2017), extracting structured representations, and to perform differentiable MPE inference allowing SPNs to be directly trained to reconstruct their input, bridging the gap even more between SPNs, autoencoders and other neural networks.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable feedback. RP acknowledges the support by Arm Ltd. AM and KK acknowledge the support by the DFG CRC 876 "Providing Information by Resource-Constrained Analysis", project B4. KK acknowledges the support by the Centre for Cognitive Science at the TU Darmstadt.

References

Amer, M., and Todorovic, S. 2016. Sum product networks for activity recognition. *TPAMI* 38(4):800–813.

Bekker, J.; Davis, J.; Choi, A.; Darwiche, A.; and Van den Broeck, G. 2015. Tractable learning for complex probability queries. In *NIPS*.

Bengio, Y.; Courville, A. C.; and Vincent, P. 2012. Unsupervised Feature Learning and Deep Learning: A review and new perspectives. *arXiv* 1206.5538.

Chan, H., and Darwiche, A. 2006. On the robustness of most probable explanations. In *UAI*, 63–71.

Cheng, W.; Kok, S.; Pham, H. V.; Chieu, H. L.; and Chai, K. M. A. 2014. Language modeling with Sum-Product Networks. In *INTERSPEECH 2014*, 2098–2102.

Choi, A., and Darwiche, A. 2017. On relaxing determinism in arithmetic circuits. In *ICML*, 825–833.

Coates, A.; Lee, H.; and Ng, A. Y. 2011. An analysis of single layer networks in unsupervised feature learning. In *AISTATS*.

Conaty, D.; Mauá, D. D.; and de Campos, C. P. 2017. Approximation complexity of maximum a posteriori inference in sum-product networks. In *UAI*, 322–331.

Darwiche, A. 2003. A differential approach to inference in bayesian networks. *J.ACM*.

Dembczyński, K.; Waegeman, W.; Cheng, W.; and Hüllermeier, E. 2012. On label dependence and loss minimization in multi-label classification. *MLJ* 88(1):5–45.

Dennis, A., and Ventura, D. 2012. Learning the Architecture of Sum-Product Networks Using Clustering on Varibles. In *NIPS*, 2033–2041.

Di Mauro, N.; Vergari, A.; and Esposito, F. 2016. Multi-label classification with cutset networks. In *PGM*.

Erhan, D.; Bengio, Y.; Courville, A.; and Vincent, P. 2009. Visualizing Higher-Layer Features of a Deep Network. *ICML 2009 Workshop on Learning Feature Hierarchies*.

Finley, T., and Joachims, T. 2008. Training structural svms when exact inference is intractable. In *ICML*, 304–311.

Gens, R., and Domingos, P. 2012. Discriminative Learning of Sum-Product Networks. In *NIPS*, 3239–3247.

Gens, R., and Domingos, P. 2013. Learning the Structure of Sum-Product Networks. In *ICML*, 873–880.

Germain, M.; Gregor, K.; Murray, I.; and Larochelle, H. 2015. MADE: masked autoencoder for distribution estimation. *arXiv* 1502.03509.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science*.

Larochelle, H., and Bengio, Y. 2008. Classification using discriminative restricted boltzmann machines. In *ICML*, 536–543.

Lowd, D., and Domingos, P. 2008. Learning arithmetic circuits. In *UAI*, 383–392.

Marlin, B. M.; Swersky, K.; Chen, B.; and Freitas, N. D. 2010. Inductive Principles for Restricted Boltzmann Machine Learning. In *AISTATS*, 509–516.

Melibari, M.; Poupart, P.; Doshi, P.; and Trimponias, G. 2016. Dynamic sum product networks for tractable inference on sequence data. In *PGM*.

Molina, A.; Natarajan, S.; and Kersting, K. 2017. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In *AAAI*.

Peharz, R.; Tschiatschek, S.; Pernkopf, F.; and Domingos, P. 2015. On theoretical properties of sum-product networks. In *AISTATS*.

Peharz, R.; Gens, R.; Pernkopf, F.; and Domingos, P. M. 2017. On the latent variable interpretation in sum-product networks. *TPAMI* 39:2030–2044.

Peharz, R.; Geiger, B.; and Pernkopf, F. 2013. Greedy Part-Wise Learning of Sum-Product Networks. In *ECML-PKDD*.

Poon, H., and Domingos, P. 2011. Sum-Product Networks: a New Deep Architecture. In *UAI*.

Pronobis, A.; Riccio, F.; and Rao, R. P. N. 2017. Deep spatial affordance hierarchy: Spatial knowledge representation for planning in large-scale environments. In *ICAPS Workshop on Planning and Robotics*.

Rifai, S.; Vincent, P.; Muller, X.; Glorot, X.; and Bengio, Y. 2011. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*.

Rooshenas, A., and Lowd, D. 2014. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In *ICML*.

Vergari, A.; Peharz, R.; Di Mauro, N.; Molina, A.; Kersting, K.; and Esposito, F. 2017. Code and supplemental material for the present paper. github.com/arranger1044/spae.

Vergari, A.; Di Mauro, N.; and Esposito, F. 2015. Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning. In *ECML-PKDD*, 343–358.

Vergari, A.; Di Mauro, N.; and Esposito, F. 2016. Visualizing and understanding sum-product networks. *arXiv:1608.08266*.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR* 11:3371–3408.

Wicker, J.; Tyukin, A.; and Kramer, S. 2016. A nonlinear label compression and transformation method for multi-label classification using autoencoders. In *PAKDD*, 328–340.

Zhao, H., and Poupart, P. 2016. A unified approach for learning the parameters of sum-product networks. *arXiv:1601.00318*.

Zhao, H.; Melibari, M.; and Poupart, P. 2015. On the Relationship between Sum-Product Networks and Bayesian Networks. In *ICML*.

Zohrer, M.; Peharz, R.; and Pernkopf, F. 2015. Representation learning for single-channel source separation and bandwidth extension. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23(12):2398–2409.