ENCODING AND DECODING REPRESENTATIONS WITH SUM- AND MAX-PRODUCT NETWORKS

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

Robert Peharz
Institute of Physiology (iDN)
Medical University of Graz, Austria
robert.peharz@medunigraz.at

Nicola Di Mauro, Floriana Esposito
Department of Computer Science
University of Bari, Italy
{nicola.dimauro,floriana.esposito}@uniba.it

ABSTRACT

Sum-Product networks (SPNs) are expressive deep architectures for representing probability distributions, yet allowing exact and efficient inference. SPNs have been successfully applied in several domains, however always as black-box distribution estimators. In this paper, we argue that due to their recursive definition, SPNs can also be naturally employed as hierarchical feature extractors and thus for unsupervised representation learning. Moreover, when converted into Max-Product Networks (MPNs), it is possible to decode such representations back into the original input space. In this way, MPNs can be interpreted as a kind of generative autoencoder, even if they were never trained to reconstruct the input data. We show how these learned representations, if visualized, indeed correspond to "meaningful parts" of the training data. They also yield a large improvement when used in structured prediction tasks. As shown in extensive experiments, SPN and MPN encoding and decoding schemes prove very competitive against the ones employing RBMs and other stacked autoencoder architectures.

1 INTRODUCTION

On a high level, the generative approach to machine learning can be described as follows: Given a set of samples $\mathcal{D}$, drawn (usually i.i.d.) from an unknown distribution p^* over random variables (RVs) $\mathbf{X}$, recover p^* from $\mathcal{D}$. To a certain extent, generative learning (GL) can be seen as the "kingclass" paradigm in machine learning. It is well known that an optimal predictor – given an additional loss function – can just be derived from p^*. For example, assuming that Y is a class variable and $\mathbf{X}$ are observed features, the classifier with minimal expected $0/1$-loss is given by $\operatorname{argmax}_y \; p^*(y, \mathbf{X})$.

It is therefore not surprising that GL and representation learning (RL) (Bengio et al., 2012) are highly related, as both aim at "formally understanding" data. GL can be described as a "black-box" approach, since we are usually interested in the capability of some model p_θ to capture the underlying distribution p^*. In RL, however, one may be interested in interpreting the "inner parts" of p_θ as abstract features of the original raw data. Both perspectives can be seen in the seminal RL approaches (Hinton & Salakhutdinov, 2006; Bengio et al., 2006), as the activations of generatively trained models are employed as data representations for initializing deep architectures.

As another simple example, consider a Bayes classifier, which estimates the joint distribution $p(Y, \mathbf{X})$ by using the class-prior $p(Y)$ and class-conditionals $p(\mathbf{X} \mid Y)$. In a purist GL view, we estimate $p(Y)$ and $p(\mathbf{X} \mid Y)$ to compute $p(Y, \mathbf{X}) = p(\mathbf{X} \mid Y)\, p(Y) \propto p(Y \mid \mathbf{X})$. In an RL approach, however, we would recognize that the parts of our model $p(\mathbf{X} \mid Y)$ (or also $p(Y \mid \mathbf{X})$) can be interpreted as a kind of soft one-hot encoding for Y, and would use them as *features* in a discriminative approach. The same argument holds for an unsupervised learning scenario, i.e. when Y is unobserved: we would deal with *latent mixture models* for which $p(\mathbf{X} \mid Y)$ are the mixture components. In summary, we

note that any generative model – depending on its structure and semantics – might be a potential feature extractor and thus a potential useful representation learner.

In this paper, we investigate a particular promising candidate for this approach, namely Sum-Product Networks (SPNs) (Poon & Domingos, 2011), recently proposed deep probabilistic networks, admitting exact but tractable inference for several kinds of probabilistic queries. SPNs have been successfully applied to computer vision (Gens & Domingos, 2012; Amer & Todorovic, 2015), speech (Peharz et al., 2014b; Zöhrer et al., 2015) and language modeling (Cheng et al., 2014). In these works, however, SPNs have been used only as black-box distribution estimators.

Here we exploit SPNs for RL. One way to interpret SPNs is as a hierarchically structured generalization of mixture models: they are nested arrangements of *factorized distributions* (product nodes) and *mixture distributions* (weighted sum nodes) defining distributions over subsets of $\mathbf{X}$. Due to this peculiarity, representations extracted from an SPN by evaluating the network nodes extend the idea of using mixture components as features, as in the motivating example above, in a recursive way. In Vergari et al. (2016) some initial approaches to *encode* embeddings via an SPN were proposed, showing how these model can constitute an interesting alternative or addition to other popular generative feature extractors such as RBMs (Hinton & Salakhutdinov, 2006; Marlin et al., 2010). The advantages of employing SPNs for RL are that one can "easily" learn both structure and parameters by leveraging the SPN's recursive probabilistic semantics (Gens & Domingos, 2013), rather than imposing an a-priori structure or using an ad-hoc weight learning algorithm, as usually done for other deep architectures. Rich hierarchical features can be obtained even by such a simple generative learning scheme. Indeed, in an SPN each node can be seen as a probabilistic *part-based* feature extractor. Visualizations of the filters learned by SPNs trained on images data confirm that these networks are able to learn meaningful representations at different levels of abstraction.

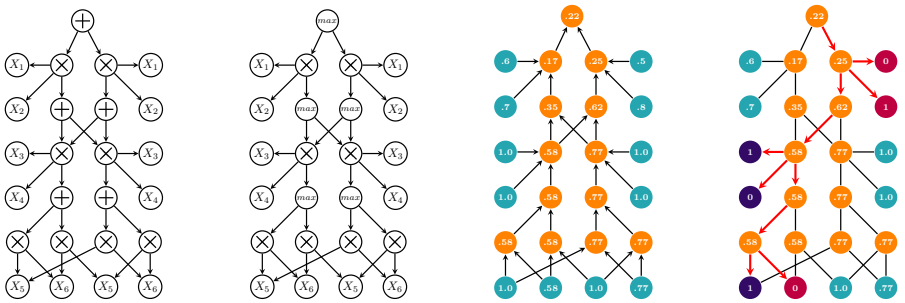
In this work we provide a way to *decode* the learned representations back to their original space by employing a Max-Product Network (MPN) (Poon & Domingos, 2011). Our decoding procedure leverages the Most Probable Explanation (MPE) (Darwiche, 2009) inference routine for SPNs and incorporates an imputation mechanism for missing components in a representation to be decoded. To a certain extent, an MPN can be exploited as a kind of generative autoencoder. We continue the work of Vergari et al. (2016) by adding other ways to leverage SPN representations, again for "free", i.e. without training the network with the aim to reconstruct its input. Additionally, we characterize conditions when MPNs can be considered perfect encoder-decoders under the proposed scheme. As a final contribution, we evaluate the meaningfulness and usefulness of SPN and MPN representations in an extensive set of structured output prediction tasks. Having devised a decoding procedure allows us to explore different learning scenarios, e.g. building embeddings for the input features, for the labels or for both. We demonstrate that these encoding and decoding schemes, "cheaply" obtained by a generative SPN, show surprisingly competitive performances when compared to those extracted from RBMs, probabilistic autoencoders (Germain et al., 2015) and deep autoencoders tailored for label embeddings (Wicker et al., 2016) in all the learning scenarios evaluated.

2 SUM-PRODUCT NETWORKS

Let RVs be denoted by upper-case letters, e.g. X, Y and let corresponding lower-case letters denote their values, e.g. $x \sim X$. Similarly, boldface notation denotes sets of RVs and their combined values, e.g. $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{x}$, $\mathbf{y}$. For $\mathbf{Y} \subseteq \mathbf{X}$ and a sample $\mathbf{x}$, we denote with $\mathbf{x}_{|\mathbf{Y}}$ the restriction of $\mathbf{x}$ to $\mathbf{Y}$.

An SPN S over a set of RVs $\mathbf{X}$ is a probabilistic model defined via a rooted DAG. The *leaves* of the graph (the SPN's inputs) are computationally tractable, possibly unnormalized distributions over a sub-set of $\mathbf{X}$. When n is a leaf of S, let ϕ_n denote its associated distribution. The *inner nodes* compute either *weighted sums* or *products* over their children. Let $\mathsf{ch}(n)$ be the set of children for a particular node n. For a sum node n and a child $c \in \mathsf{ch}(n)$, we associate a *nonnegative* weight w_{nc} with the outgoing sum-edge $n \to c$. The set of all sum weights in S (the network parameters) is denoted as $\mathbf{w}$. Furthermore, let $\mathbf{S}^{\oplus}$ (resp. $\mathbf{S}^{\otimes}$) be the set of all sum (resp. product) nodes in S.

Let S_n denote the sub-network rooted at node n and parametrized by $\mathbf{w}_n$. Each node n in S defines a probability distribution $p_{\mathbf{w}_n}$ over its scope by normalizing the output of S_n. Consequently, the distribution of S over all $\mathbf{X}$ is defined as the root normalized output. $S_n(\mathbf{x}_{|\mathsf{sc}(n)})$, or short-hand $S_n(\mathbf{x})$, indicates the output value of node n when $\mathbf{X} = \mathbf{x}$ is observed as the network input.



(a) (b) (c) (d)

Figure 1: A complete and decomposable SPN S with leaves over univariate distributions labeled by their scopes (1a); the MPN M obtained from S (1b); and its bottom-up evaluation to solve $\text{argmax}_{\mathbf{q}\sim\mathbf{Q}}\, p(\mathbf{q}, X_1 = 0, X_2 = 1, X_6 = 0)$ (1c) with $\mathbf{Q} = \{X_3, X_4, X_5\}$. Orange (resp. blue) for inner (resp. leaf) activations. A tree path highlighted by MPEAssignment in the top-down traversal of M (1d). The assignment for RVs $\mathbf{Q}$ (resp. $\mathbf{O} = \{X_1, X_2, X_6\}$) is the violet (resp. purple) leaves.

Let $\mathsf{sc}(*)$ denote the *scope*, a labeling function associating to each node n a subset of $\mathbf{X}$, i.e. $\mathsf{sc}(n) \subseteq \mathbf{X}$. For a leaf n, $\mathsf{sc}(n)$ is 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 gives rise to some fundamental properties of an SPN: S is *complete* if $\forall n \in \mathbf{S}^{\oplus}$ and $\forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$. S is *decomposable* if $\forall n \in \mathbf{S}^{\otimes}$ and $\forall c_1, c_2 \in \mathsf{ch}(n), c_1 \neq c_2 : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$ (an example in Figure 1a).

While inference in unconstrained SPNs is intractable, *marginalization* in complete and decomposable SPNs reduces to performing the marginalization task at the leaves and evaluating the inner nodes as usual (Poon & Domingos, 2011; Peharz et al., 2015). An SPN is *locally normalized* when it holds that $\forall n \in \mathbf{S}^{\oplus} : \sum_{c\in\mathsf{ch}(n)} w_{nc} = 1$ and all leaves represent normalized distributions. For complete, decomposable and locally normalized SPNs, the distributions of all nodes are already correctly normalized distributions. In the following, we only consider this class of SPNs.

While marginalization can be tackled in time linear in the network size, the problem of finding a *Most Probable Explanation* (MPE) is generally NP-hard in SPNs (Peharz et al., 2016). Given two sets of RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$ and $\mathbf{Q} \cap \mathbf{O} = \emptyset$, inferring an MPE assignment is defined as finding

$$\mathbf{x}^*_{|\mathbf{Q}} = \operatorname*{argmax}_{\mathbf{q}\sim\mathbf{Q}} p(\mathbf{o}, \mathbf{q}). \tag{1}$$

However, MPE can be solved exactly in *selective* SPNs (Peharz et al., 2014b; 2016), i.e. SPNs where it holds that for each sample $\mathbf{x}$ at most one child of each sum node is non-zero. MPE in selective SPNs is solved via MPEAssignment (Poon & Domingos, 2011), which evaluates the network twice. First one builds an MPN M from S by replacing each node $n \in \mathbf{S}^{\oplus}$ by a *max node* $n \in \mathbf{M}^{\mathsf{max}}$ computing $\max_{c\in\mathsf{ch}(n)} w_{nc} M_c(\mathbf{x})$ and each leaf distribution by a maximizing distribution (Peharz et al., 2016) (Figure 1b). One then computes $M(\mathbf{x}_{|\mathbf{O}})$ – the MPE probability of the query $p(\mathbf{x}_{|\mathbf{O}})$ – by evaluating M bottom-up (Figure 1c). Stage two consists of a top-down traversal of M. Starting from the root, one follows the maximal child branch for each max node and all child branches of a product node. Each partial input configuration determines a unique *tree path*. The MPE assignment $\mathbf{x}^*$ is obtained by collecting the MPE solutions (w.r.t. $\mathbf{Q}$) of the leaves in the path (Figure 1d). For selective SPNs, the corresponding MPNs compute precisely the same value for each node, since sums and maxes are equivalent when applied to all zeros but one nonnegative value. In the non-selective case, MPNs can be seen as a (lower-bounding) approximation of SPNs, and are thus also an interesting candidate for RL, as showed in the next sections. Furthermore, while MPEAssignment solution for general SPNs is not exact, it is still employable as a reasonable and common approximation (Peharz et al., 2016).

SPNs and MPNs can be interpreted as very peculiar deep Neural Networks (ANNs) that are *labeled*, *constrained* and *fully probabilistic* (Vergari et al., 2016). They are labeled networks because of the scope function, which enables a *direct encoding* of the input (Bengio et al., 2012). Their topology is constrained because of the completeness and decomposability properties, hence connections are sparse and not dense. Differently from other distribution estimators like NADEs (Larochelle &

Murray, 2011) and MADEs (Germain et al., 2015), they are fully probabilistic: not only the network outputs emit valid probability values, but each inner node as well, due to their recursive definition.

The semantics of SPNs enable the design of simple and yet surprisingly effective structure learning algorithms (Dennis & Ventura, 2012; Peharz et al., 2013; Gens & Domingos, 2013). Many recent attempts and variants (Rooshenas & Lowd, 2014; Adel et al., 2015; Vergari et al., 2015) build upon the currently most prominent algorithm LearnSPN, a greedy top-down SPN learner introduced in (Gens & Domingos, 2013). LearnSPN proceeds by recursively 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. To a certain extent, LearnSPN can be interpreted as a recursive *data crawler*, extracting peculiar features from a data matrix, which potentially only live in a particular data cluster and/or in a certain subset of RVs. This may be one of the few cases when the structure and parameters of an ANN can be learned without directly optimizing a loss function.

3 LEARNING REPRESENTATIONS WITH SPNS AND MPNS

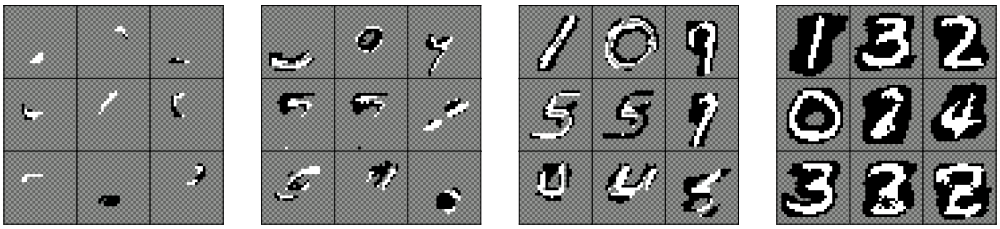
In this section we discuss how to exploit an SPN S or its corresponding MPN M for RL, after structure and parameters are generatively learned over $\mathbf{X}$, following Vergari et al. (2016). We are interested in *encoding* each sample $\mathbf{x}^i \sim \mathbf{X}$ into a continuous vector representation $\mathbf{e}^i$ in a new d-dimensional space, i.e. an *embedding* $\mathbf{e}^i \in \mathbf{E}_\mathbf{X} \subseteq \mathbb{R}^d$, where $\mathbf{e}^i = f_S(\mathbf{x}^i)$ (SPNs) or $\mathbf{e}^i = f_M(\mathbf{x}^i)$ (MPNs). We usually refer to SPNs, since most of the time similar consideration hold also for MPNs.

For ANNs, the common approach is to use the hidden neuron activations of the upper layers as the learned representations for f. As argued above, SPN nodes are particular interpretable due to their clear probabilistic semantics. Given an SPN S and a set of nodes $\mathbf{N} = \{n_j\}_{j=1}^d \subset \mathbf{S}$, we construct our embedding as $e^i_j = S_{n_j}(\mathbf{x}_{|\mathsf{sc}(n_j)}) = p_{\mathbf{w}_{n_j}}(\mathbf{x}_{|\mathsf{sc}(n_j)})$, where a reasonable selection criterion for $\mathbf{N}$ is given below. Each value represents the probability to see that sample according to a *marginal* distribution over a node scope. Thus, the so-constructed embedding is a point in the geometric space induced by a collection of proper *probability densities*.

SPN nodes can also be seen as *part-based filters* operating over sub-spaces given by the node scopes. Sum nodes can be interpreted as filters built by weighted averages over filters sharing the same scope, and product nodes can be seen as compositions of filters over non-overlapping scopes. From the perspective of the internal mechanisms of LearnSPN-like algorithms, each filter captures a different aspect of sub-population and sub-space of the data. Thereby, the scope information induces a hierarchy of filters at different *levels of abstraction*.

To confirm this interpretation, we visualize the features extracted from nodes in an SPN learned on image samples (Vergari et al., 2016). For ANNs, the feature filtered by a hidden neuron can be visualized as the image in the input space that maximally activates that neuron (Erhan et al., 2009). In SPNs this corresponds to solving MPE for the sub-SPN rooted at a particular node, and restricted to the node's scope. As stated in Section 2, we employ MPEAssignment as an approximation to this generally hard problem. Figure 2 shows some of the MPE solutions/filter activations for an SPN trained on a binarized version of MNIST (Larochelle et al., 2007) (see Appendix C for details). Note that they resemble part-based features at different levels of complexity: from small blobs (Figure 2a) to shape contours (Figures 2b and 2c), to full digits comprising background parts (Figure 2d). The missing background pixels, visualized in a checkerboard pattern, is due to those pixels being out of the scope for those nodes. This *pattern locality* is an SPN peculiarity: although also fully connected ANNs typically show locality (e.g. edge filters), the locality information is *explicitly* encoded in SPNs via the node scopes. This suggests that the scope information alone may already be able to convey a meaningful representation of "object parts", e.g. see the 'O' shapes in Figure 2. Also, note that filters appear qualitatively different from most classical ANNs, which motivates to combine SPN features with those from other deep architectures, an approach worth further investigation.

While in classical deep ANNs the layer depth is usually associated with the level of abstraction of its filters (Erhan et al., 2009; Zeiler & Fergus, 2014; Yosinski et al., 2014), note that this does not easily translate to SPNs. First, even rather simple models might yield extremely deep networks, when translated into SPNs. For example, when representing a hidden Markov model (HMM) as SPN Peharz et al. (2014b), the SPN's depth grows linearly in the length of the HMM. Thus, representations learned by SPNs are not easily arranged in a meaningful layered hierarchy, due to their constrained



(a) (b) (c) (d)

Figure 2: Visualizing features learned by an SPN trained on a binarized version of MNIST: 4 clusters of 9 images generated from randomly chosen nodes from different parts of the network but having similar scope lengths. The checkerboard pattern indicates pixels out of a node scope.

topology and how they are learned. Moreover, LearnSPN-like algorithms can introduce nodes with very different scope information at the same level of depth. This also occurs when compiling SPNs into a minimal layered structure (Vergari et al., 2016).

Therefore, we suggest that rather the scope length $|\mathsf{sc}(n)|$ of a node n should be associated with its level of abstraction. The filter activations in Figure 2 give confirming evidence for our conjecture. Thus, when the aim is to compress data into an abstract representation of at most d dimensions, one reasonable filter criterion for SPN/MPN representations would be to collect the d nodes with largest scopes. Clearly, the smaller we choose d, the smaller will be the theoretically achievable quality of the reconstructed data. In our experiments, we leverage SPN representations and decoding schemes by adopting full embeddings, comprising all nodes activations (all colored values in Figure 1c), and inner embeddings, dropping out the leaf information (only orange values in Figure 1c), according to the observation that the number of leaves is overabundant w.r.t. inner nodes in SPNs built by LearnSPN (Vergari et al., 2016). Note that, in both cases, the embedding size d is adaptively induced by the data, when building the SPN, without the need to fix or tune it beforehand.

4 Decoding representations

Now we tackle the task to revert SPN representations back to the input space, i.e. to find an inverse transformation $g\colon \mathbf{E}_{\mathbf{X}} \rightarrow \mathbf{X}$ such that $\mathbf{x}^i \approx \hat{\mathbf{x}}^i = g(f(\mathbf{x}^i))$. Being able to decode representations extends the ways one can exploit SPNs for RL to new learning scenarios for predictive tasks. For example, if one were to learn a classifier from features $\mathbf{X}$ to labels $\mathbf{Y}$, he could train the classifier to predict the *label embeddings* $\mathbf{E}_{\mathbf{Y}}$ rather than $\mathbf{Y}$ directly. Then, the predicted embeddings could be turned into the actual outputs by applying g for decoding. By disentangling dependencies over $\mathbf{Y}$ in the new space $\mathbf{E}_{\mathbf{Y}}$, one can obtain better predictors. Following this principle, label embeddings have been greatly employed in RL for structured output prediction. One common approach is to compress labels into a lower dimensional space, then a regressor is trained to predict such embeddings and the predictions are decoded (decompressed) by an inverse linear transformation Bhatia et al. (2015); Akata et al. (2013). The advantage of the decoding scheme we propose is that g does not need additional training to be learned, rather it is provided by an already learned SPN turned into MPN.

Let a *perfect encoder-decoder* be a pair (f, g) such that, for each $\mathbf{x} \sim \mathbf{X}$, its reconstruction is the exact same sample, i.e. $g(f(\mathbf{x})) = \mathbf{x}$. In our analysis, we focus on MPNs and characterize when they can be used as perfect encoder-decoders. In practice, autoencoders, for which f and g are learned from data, are usually not trained to be perfect encoder-decoders, as they often might learn trivial representation, such as the identity function. This seems not to be an issue for MPNs, since the learning phase is decoupled from the decoding one. We will also empirically confirm it in Section 5.

Given an MPN M, the encoder function f_M is given by collecting activations as illutrated in Section 3. Concerning the decoder, we propose a procedure for g_M that mimics the MPEassignment algorithm as presented in Section 2. Recall that MPEAssingment finds a solution in the input space (top-down phase) after probabilities, i.e. node activations, are evaluated (bottom-up phase). Consider Eq. 1 in the case in which $\mathbf{Q} = \emptyset$ and sample $\mathbf{x}^i$ is fully observed. If all the activations from the bottom-up phase are collected into an embedding $\mathbf{e}^i$, its components will exactly determine the top-down descending

phase, i.e. which branch to take when encountering a max node. As a consequence, the set of leaves in the traversed tree path completely *depend on* $\mathbf{e}^i$. This is also true if $\mathbf{e}^i$ components are not determined from a bottom-up phase but come from the "outside", e.g. they are predicted. In order to completely define g_M, each leaf node encoding ϕ_n reached in the top-down phase has to provide itself a decoder function g_{ϕ_n}, operating over its scope. Similarly to MPEAssignment, a fully decoded embedding is then constructed by collecting the reconstructions at the leaves according to each g_{ϕ_n} decoder.

In practice, we are interested in decoding embeddings that have been predicted by some learned model, i.e. the decoding phase is not applied to the embeddings obtained by directly evaluating a network. Nevertheless, it is important to determine under which circumstances these models behave as perfect encoder-decoders when transforming each instance to a new representation and back.

Proposition 1. *If for an MPN M over $\mathbf{X}$ there exist a perfect encoder-decoder for each leaf distribution ϕ_n and it holds for each max node $n \in M$ that there is only one child node $c \in \mathsf{ch}(n)$ for which $M_n(\mathbf{x}) = w_{nc} M_c(\mathbf{x})$, given $\mathbf{x} \sim \mathbf{X}$, then M is a perfect encoder-decoder.*

Proof. It is easy to demonstrate this by inductive reasoning. If M comprised only a leaf node, then it would be a perfect encoder-decoder by definition. If it were composed by a product node over child encoder-decoder MPNs, then each input could be reconstructed perfectly by the composition of the reconstruction of the child MPNs. Lastly, if it were composed by a max node over child perfect encoder-decoder MPNs $M_c, c = 1 \dots k$, then it would also be a perfect encoder-decoder since for each possible input, only one child component M_{c^*} would output a value s.t. $M_n = w_{nc^*} M_{c^*}$. $\square$

From Proposition 1 it follows immediately that deterministic MPNs can be perfect encoder-decoders.

Proposition 2. *An MPN M constructed from a* selective *SPN (Peharz et al., 2014a) S is a perfect encoder-decoder, provided that the leaves have perfect encoder-decoder functions.*

Thus, to complete our decoding procedure, we still have to cope with the leaf decoder functions. We define the decoded state for a leaf n as the configuration over its scope that minimizes some distance D over the leaf activation value and its encoded representation: $\tilde{\mathbf{x}}_{|\mathsf{sc}(n)} = \operatorname{argmin}_{\mathbf{u} \sim \mathsf{sc}(n)} D(\phi_n(\mathbf{u}) || f_{M_n}(\mathbf{x}))$. In our experiments we will employ simple L_1 distance $|\phi_n(\mathbf{u}) - f_{M_n}(\mathbf{x})|$. Unfortunately, decoding is ambiguous for most interesting leaf distributions, such as Gaussians. However, this approach works well for discrete data used in our experiments, as long as the state probabilities are mutually distinct. In future work, we will explore techniques to disambiguate decoding the leaves, e.g. by duplicating and splitting Gaussians. In Section 5, we empirically evaluate how good are the decoding schemes depicted here, since it is worth investigating how close to perfect encoder-decoders MPNs learned on real datasets can be.

In order to apply the proposed decoding procedure, a full embedding comprising all the node activations is required. In some real cases (e.g. data compression), only an *incomplete embedding*, comprising only activations from a subset of the network nodes, is available. For certain incomplete embeddings a full decoding, however, is still possible.

A *decodable incomplete embedding* $\mathbf{e}$ is an embedding such that for each missing activation $M_n(\mathbf{x}) \notin \mathbf{e}$ corresponding to a node $n \in \mathbf{M}$, all the activations $e_c = M_c(\mathbf{x})\ \forall c \in \mathsf{ch}(n)$ are in $\mathbf{e}$. For such an incomplete embedding, it is sufficient to evaluate the MPN by propagating the embedding activations bottom-up, evaluating parent nodes after their children. The missing embedding components are then reconstructed and the decoding phase for the now full embedding can proceed as before. If even this child information is missing, such a reconstruction is not possible in general. We argue that in such a case, the missing node activations can be reasonable imputed by their most probable value. When encountering a node n_j, whose corresponding embedding value e_j is not available, e_j is estimated as $\max_{\mathbf{u} \sim \mathsf{sc}(n_j)} M_{n_j}(\mathbf{u})$ by employing MPEassignment on the sub-networks rooted at n_j. Since the MPE activations can be precomputed for all nodes, the complexity of the whole procedure is still linear in the size of M. The pseudocode for the complete decoding procedure is listed in Appendix A.

In our experiments we evaluate the effectiveness and robustness of the decoding procedure both for complete (full) and incomplete predicted embeddings. In particular, for structured output prediction, we employ inner embeddings (cf. Section 3), where leaf values are imputed using MPEassignment as stated above. Moreover, we investigate its resilience when imputing missing at random embedding components either by just replacing them by their MPE value or by additionally evaluating the MPN bottom-up after the missing leaf activations have been imputed.

5 EXPERIMENTS

The research questions we are validating are: i) how good are learned MPNs at reconstructing their input when full/inner embeddings are decoded? ii) how meaningful are the representations learned by SPNs and MPNs and how useful is to predict these embeddings instead of the raw targets and then decoding them? iii) how resilient to missing components are the proposed decoding schemes?

Structured output prediction tasks like Multi-label Classification (MLC) offer a good experimental ground to answer the above questions. In MLC one is interested in predicting the target labels associated to a sample $\mathbf{x} \sim \mathbf{X}$ and represented as binary arrays: $\mathbf{y} \sim \mathbf{Y}$. Since there is no unique way to assess a classifier performance in MLC, we measure the JACCARD, HAMMING and EXACT MATCH 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).

For all experiments we use 10 standard benchmark datasets for MLC. To fairly compare all the algorithms in our experiments, we employ the binarized versions of all datasets already processed by Di Mauro et al. (2016) and divided in 5 folds. Detailed dataset statistics are reported in Appendix B.

We learn both the structure and weights of our SPN, and hence MPN, models on $\mathbf{X}$ and$\mathbf{Y}$ separately for each fold by employing LearnSPN-b (Vergari et al., 2015), a variant of LearnSPN (see Appendix C)[1]. Structural statistics, e.g. the number of inner nodes, for all the models are reported in Appendix D. Please refer to Tables 3 and 4 to determine the extracted embedding sizes.

5.1 RECONSTRUCTION PERFORMANCES

We want to determine how close to perfect encoder/decoders are MPNs learned from real data and equipped with our decoding schemes. In particular, we evaluate their decoding performances when the leaf activations are available (full embeddings), and the decoder employed is the L_1 distance, or when they are missing (inner embeddings) and therefore their MPE state is used.

First we turn each learned SPN into an MPN. Then, each model is asked to reconstruct both the training and test samples. Detailed results are reported in Tables 5 and 6, Appendix E.2. It can be observed that the L_1 leaf decoder proves to be a very reasonable approximation for binary RVs, scoring very high reconstructions for all the three measures. For the models over $\mathbf{Y}$, the MPE approximation scores surprisingly good reconstructions scoring $> 80\%$ EXACT MATCH on half datasets. In general, if the network is small enough, e.g., MPNs learned on the Flags dataset or on $\mathbf{Y}$ alone, it behaves as a perfect encoder-decoder for full embeddings. This demonstrates the efficacy of the proposed decoding schemes and shows how the presence of tied max node children activations impacts non-deterministic MPNs learned from data. We investigate if these potentially perfect reconstructions lead to banal representations in the following experiments.

5.2 STRUCTURED OUTPUT PREDICTION PERFORMANCES

We now focus on leveraging the representations learned by SPNs and MPNs in an unsupervised way for structured output prediction. In a fully supervised scenario one wants to build a classifier on the input RVs $\mathbf{X}$ to predict the output RVs $\mathbf{Y}$ directly ($\mathbf{X} \rightarrow \mathbf{Y}$). Instead, we can first encode both the input RVs $\mathbf{X}$ and/or the target RVs $\mathbf{Y}$ into different embedding spaces, $\mathbf{E_X}$, $\mathbf{E_Y}$, and build a predictive model on top of them. In order to do so, we explore different settings: we learn a classifier on the input embeddings instead of the raw features ($\mathbf{E_X} \rightarrow \mathbf{Y}$); alternatively, one can first train a regressor on the original input $\mathbf{X}$ to predict label embeddings ($\mathbf{X} \rightarrow \mathbf{E_Y}$), then decoding such predictions back to the original label space; finally, the same regressor can be trained on the input embeddings instead ($\mathbf{E_X} \rightarrow \mathbf{E_Y}$) and its predictions decoded as above.

As a *proxy* measure to assess the meaningfulness of the learned representations, we are considering their prediction performances. Given a predictive model, its improvement in performance in one of the above settings over the raw input/output case, $\mathbf{X} \rightarrow \mathbf{Y}$, determines how good the representations employed are. To highlight the ability of these representations to disentangle the dependencies underlying the RVs, we always train a simple linear model in all the settings. In particular, we employ an L_2-regularized logistic regressor, LR, (resp. a ridge regressor, RR) to predict each RV

[1] All the code employed for the experiments and visualizations will be made available

in $\mathbf{Y}$ (resp. component in $\mathbf{E}_\mathbf{Y}$) independently. Therefore, the most natural baseline to measure the aforementioned representation meaningfulness is to employ the same L_2-logistic regressor to the $\mathbf{X} \rightarrow \mathbf{Y}$ setting.

We now introduce other models as either encoders or encoder/decoders to plug into our unsupervised settings. The aim is to compare SPN/MPN representations against theirs w.r.t. the aforementioned baseline. Therefore we select them as generative models for which inference can be exactly and tractably computed. For the $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{Y}$ setting, we compare to RBMs (Smolensky, 1986) as highly expressive generative models, for which, while the joint likelihood is intractable for RBMs, exact conditionals of the latent RVs can be computed exactly and have been proven to be very predictive features (Larochelle & Bengio, 2008; Marlin et al., 2010; Larochelle et al., 2010). To evaluate different embedding sizes, we consider RBMs having 500, 1000 and 5000 hidden units (h). A natural competitor for all settings are MADEs (Germain et al., 2015), because they are deep autoencoders which are also tractable probabilistic models. We employ MADEs comprising 3 layers and 500 and 1000 (resp. 200 and 500) hidden units per layer for the $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{Y}$ (resp. $\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$) setting.

Additionally, we add to the comparison MANIAC (Wicker et al., 2016) a non-probabilistic autoencoder model tailored to MLC. In MANIAC, stacked autoencoders are trained to reconstruct $\mathbf{Y}$ by compressing each label into a new representation, which, in turn, is used to train a base model exactly as in our $\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$ setting. We employ architectures up to 4 hidden layers with different compression factors. Finally, we employ a max-margin CRF Finley & Joachims (2008), $\mathsf{CRF}_{\mathsf{SSVM}}$, in the $\mathbf{X} \rightarrow \mathbf{Y}$ setting that considers a dependency structure on the label space in the form of a Chow-Liu tree. In this way we are able to frame the performances of all the models in the unsupervised setting against a *fully* supervised and *discriminative* method on the same datasets.

In Appendix E.1 we report all the choices made to learn and tune the involved models. For SPNs, and hence MPNs, we do not need to define a handcrafted structure a priori like for all the competitors above. Consequently, for RBMs, MADEs, MANIAC it is needed to learn and cross-validate several models with different capacities to obtain properly sized embeddings. On the other hand, the size of embeddings extracted from SPNs/MPNs is adaptively determined by data, as stated in Section 3. The learned embedding sizes are reported in Tables 3 and 4 in Appendix D.

5.2.1 Results and discussion

Detailed average fold metrics and their average ranks for all datasets are reported in Tables 9, 7, 8 in Appendix E.3.1. In Table 1, instead, we report the aggregated scores over all datasets $d \in \mathcal{D}$ for each method f in the form of the average relative improvement w.r.t. the LR baseline: $\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \frac{\mathsf{score}_f(d)-\mathsf{score}_{\mathsf{LR}}(d)}{\mathsf{score}_{\mathsf{LR}}(d)} \cdot 100$. The best models for each setting and score are in bold, the higher their improvement, the better.

In summary, SPN and MPN embeddings proved to be highly competitive and even superior to all other models in the three settings and for all the scores. Even the fully supervised and discriminative $\mathsf{CRF}_{\mathsf{SSVM}}$ performance are comparable to the best SPN/MPN JACCARD (resp. HAMMING) score in the $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$ (resp. $\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$) setting, while reporting a largely worse EXACT MATCH improvement than our models in the $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$ setting.

In particular, the setting $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{Y}$ has proven to be hard for many models. This likely indicates that the dependencies on the $\mathbf{X}$ might not contribute much to the $\mathbf{Y}$ prediction (Dembczyński et al., 2012). Representations from SPNs, even with smaller embeddings than RBMs and MADEs (see Table 3), yield the largest improvements. In the $\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$ setting, disentangling the relationships among the $\mathbf{Y}$ gives all models a performance boost. This is not the case for MADEs on some datasets, probably due to their reconstruction power being traded off to their generalization power as generative models. MPNs, on the other hand, consistently exploit the label representation space and do not provide overfitted reconstructions. This answers our question about the meaningfulness of MPN representations suggesting that their tendency to be perfect encoder/decoders does not damage their representation performances.

Concerning two decoding schemes we proposed, operating on incomplete (inner) embeddings not only performs comparably to the full case, but also scores the best results on some datasets for JACCARD and EXACT MATCH scores. This aspect can be seen in the $\mathbf{E}_\mathbf{X} \rightarrow \mathbf{E}_\mathbf{Y}$ setting as well.

Additionally, to better understand the role of our decoding procedures in the $\mathbf{X} \rightarrow \mathbf{E_Y}$ and $\mathbf{E_X} \rightarrow \mathbf{E_Y}$ settings, we run a new set of experiments in which the decoding phase for $\mathbf{E_Y}$ is performed by a nearest neighbor ($k = 5$) model on the basis of the training labelled embeddings. In these settings, therefore, we can add to the comparison even RBM-encoded representations. Even in this scenario, MPN embeddings are the best or very competitive. As expected, the non-linear kNN predictor performs better than our full decoding on several datasets for the JACCARD and EXACT MATCH scores, but less well for the inner variant. This is likely due to the smaller inner embedding sizes and highlights the goodness of the proposed decoding approach in presence of missing values and, more in general, for maximizing the HAMMING score.

All in all, with these structured output prediction tasks we gathered empirical confirmation of the meaningfulness and practical usefulness of SPN and MPN embeddings. The reported large improvements over the three scores cannot be due to SPN/MPN larger embedding sizes. In fact, their sizes are always comparable or smaller than RBM, MADE, and MANIAC ones since the latter max capacities have been chosen after SPNs have been learned (Tables 3 and 4, Appendix D). It is also not possible to state that these representation higher predictive performances are correlated to the SPN ability to better model the data distributions, at least we look at the model likelihoods. Indeed, MADE log-likelihoods have proven to be higher that SPN ones on many datasets and comparable on the rest. We argue that the reason behind these results lies in the hierarchical part-based representations SPNs provide. Each embedding component is responsible for capturing only the significant feature portions according to its corresponding node scope, as shown in Section 3. The meaningfulness of these components as features has to be found in the structure learning performed by LearnSPN-b (Section 2): while its hierarchical co-clustering chooses to split the data into sub-populations in an unsupervised way to determine a reasonable distribution estimation, it highlights meaningful ways to discriminate among them.

5.3 Resilience to missing components

Lastly, we evaluate the resilience of the decoding procedure proposed when label embedding components are missing at random in the $\mathbf{X} \rightarrow \mathbf{E_Y}$ setting. We want to compare the two imputation schemes presented in Section 4: either employing MPEAssignment to retrieve the most probable activation or evaluating the MPN bottom-up to compute the missing predicted components.

For all datasets, for each label embedding that has been predicted, we remove at random a percentage of components varying from 0 (full embedding) to 90%, by increments of 10%. If leaves activations are missing, their MPE activation is considered. After the full embedding has been reconstructed, the decoding phase proceeds as before. Figure 3 shows how the two strategies perform differently for the EXACT MATCH score. The re-evaluation scheme is much more resilient one among the two, being able to maintain comparable scores to the full embedding case up to 30% missing components, then decaying less faster than the MPE based one. The proposed decoding scheme is therefore proved to be not only surprisingly effective but also quite robust. Similar, but less prominent, behaviors are reported for the JACCARD and HAMMING scores in the Appendix.

6 Conclusion

In this work we investigated SPNs and MPNs under a RL lens. We suggested an interpretation of MPNs as generative autoencoders by providing a decoding procedure that leverages approximate MPE inference. We characterize when these networks can lead to perfect reconstructions of their inputs, linking this property to determinism. When empirically evaluated in an extensive comparison for MLC, SPN and MPN representations ranked as one of the most predictive features and MPN reconstructions proved to be surprisingly effective. Encouraged by these results, we plan to explore new learning schemes directly exploiting these models learned representatons, and not optimizing their likelihood scores only. For instance, a differentiable procedure for MPE inference would allow SPNs and MPNs to be trained directly to reconstruct or denoise their input, bridging the gap even more between these networks, autoencoders and other ANNs and opening the path to hybridize them.

Table 1: Average relative test set improvement in scores w.r.t the LR baseline (values are percentages). For each setting, best results in bold. Results for the 5-NN decoding are shown in the last two row groups.

$\mathbf{E_X}$	$\mathbf{E_Y}$	predictor	decoder	JACCARD	HAMMING	EXACT
$\mathbf{X}$	$\mathbf{Y}$	LR	-	0.00	0.00	0.00
$\mathbf{X}$	$\mathbf{Y}$	CRF_{SSVM}	-	+15.83	+9.94	+103.90
$\text{RBM}_{h=500}$	$\mathbf{Y}$	LR	-	-1.16	-2.28	-14.13
$\text{RBM}_{h=1000}$	$\mathbf{Y}$	LR	-	+0.90	-0.85	-7.19
$\text{RBM}_{h=5000}$	$\mathbf{Y}$	LR	-	+1.46	+0.20	-1.62
$\text{MADE}_{h=500}$	$\mathbf{Y}$	LR	-	+1.15	+0.00	-7.04
$\text{MADE}_{h=1000}$	$\mathbf{Y}$	LR	-	+2.57	**+0.60**	+2.99
$\text{SPN}_{\text{inner}}$	$\mathbf{Y}$	LR	-	**+3.54**	+0.50	**+17.18**
$\mathbf{X}$	$\text{MADE}_{h=200}$	RR	MADE	-30.76	+7.10	-29.71
$\mathbf{X}$	$\text{MADE}_{h=500}$	RR	MADE	-30.42	+7.04	-28.02
$\mathbf{X}$	$\text{MANIAC}_{\text{RR}}$	RR	MANIAC	+5.96	+5.07	+95.78
$\mathbf{X}$	MPN_{full}	RR	MPN	+11.65	**+10.45**	+96.30
$\mathbf{X}$	$\text{MPN}_{\text{inner}}$	RR	MPN	**+15.19**	+7.61	**+98.58**
$\text{MADE}_{h=500}$	$\text{MADE}_{h=200}$	RR	MADE	-28.14	+7.10	-28.00
$\text{MADE}_{h=500}$	$\text{MADE}_{h=500}$	RR	MADE	-27.81	+6.93	-27.14
$\text{MADE}_{h=1000}$	$\text{MADE}_{h=200}$	RR	MADE	-27.80	+6.96	-29.03
$\text{MADE}_{h=1000}$	$\text{MADE}_{h=500}$	RR	MADE	-27.15	+6.94	-25.14
$\text{SPN}_{\text{inner}}$	MPN_{full}	RR	MPN	+14.52	**+9.97**	+106.62
$\text{SPN}_{\text{inner}}$	$\text{MPN}_{\text{inner}}$	RR	MPN	**+15.98**	+7.50	**+106.65**
$\mathbf{X}$	$\text{RBM}_{h=100}$	RR	5-NN	-7.13	+6.00	+6.60
$\mathbf{X}$	$\text{RBM}_{h=200}$	RR	5-NN	-4.25	+6.82	+22.59
$\mathbf{X}$	$\text{RBM}_{h=500}$	RR	5-NN	+6.93	+8.34	+59.19
$\mathbf{X}$	$\text{MADE}_{h=200}$	RR	5-NN	+11.17	+7.37	+82.72
$\mathbf{X}$	$\text{MADE}_{h=500}$	RR	5-NN	+14.57	+7.38	+88.62
$\mathbf{X}$	MPN_{full}	RR	5-NN	**+27.10**	**+8.90**	**+133.02**
$\mathbf{X}$	$\text{MPN}_{\text{inner}}$	RR	5-NN	+21.94	+7.92	+107.00
$\text{MADE}_{h=500}$	$\text{MADE}_{h=200}$	RR	5-NN	+9.48	+7.30	+81.78
$\text{MADE}_{h=500}$	$\text{MADE}_{h=500}$	RR	5-NN	+12.77	+7.12	+85.78
$\text{MADE}_{h=1000}$	$\text{MADE}_{h=200}$	RR	5-NN	+11.89	+7.44	+84.00
$\text{MADE}_{h=1000}$	$\text{MADE}_{h=500}$	RR	5-NN	+13.12	+7.24	+90.14
$\text{SPN}_{\text{inner}}$	MPN_{full}	RR	5-NN	**+25.41**	**+8.25**	**+129.60**
$\text{SPN}_{\text{inner}}$	$\text{MPN}_{\text{inner}}$	RR	5-NN	+21.45	+7.65	+109.79

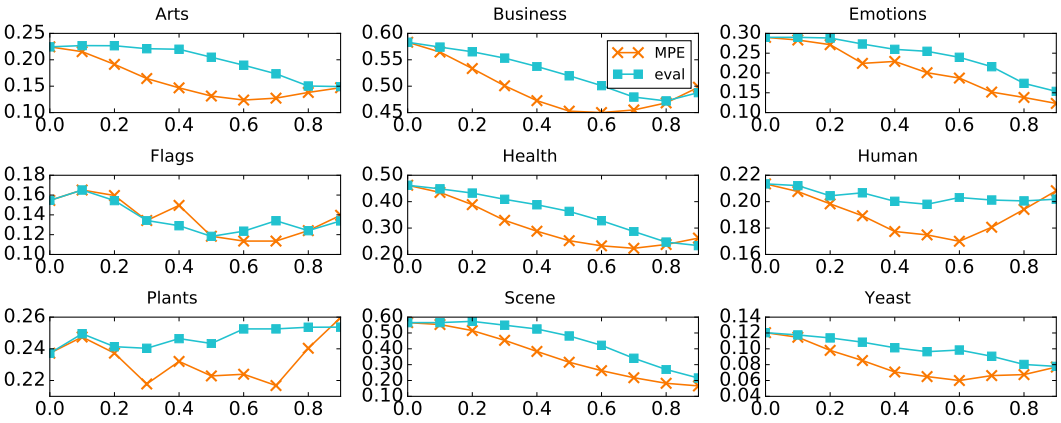


Figure 3: Average test EXACT MATCH scores (y axis) obtained by imputing different percentages of missing random embedding components (x axis) for the $\mathbf{X} \rightarrow \mathbf{E_Y}$ setting on all datasets by employing MPE inference (orange crosses) or the bottom-up evaluation imputation schemes (blue squares). Results for Cal dataset are not reported since they are all zeros (see Table 9).

REFERENCES

T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of Sum-Product Networks via an SVD-based algorithm. In *UAI*, pp. 32–41, 2015.

Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for attribute-based classification. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 819–826, June 2013.

M. Amer and S. Todorovic. Sum product networks for activity recognition. *TPAMI*, 38(4):800–813, 2015.

Alessandro Antonucci, Giorgio Corani, Denis Deratani Mauá, and Sandra Gabaglio. An ensemble of bayesian networks for multilabel classification. In *Proceedings of IJCAI*, pp. 1220–1225, 2013.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, pp. 153–160, 2006.

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

Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *NIPS 28*, pp. 730–738. 2015.

A. Cano, J.M. Luna, E.L. Gibaja, and S. Ventura. LAIM discretization for multi-label data. *Information Sciences*, 330:370–384, 2016.

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

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

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

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

N. Di Mauro, A. Vergari, and F. Esposito. Multi-label classification with cutset networks. In *Probabilistic Graphical Models*, pp. 147–158, 2016.

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

Thomas Finley and Thorsten Joachims. Training structural svms when exact inference is intractable. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 304–311, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

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

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

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

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Xiangnan Kong, Michael K. Ng, and Zhi-Hua Zhou. Transductive multilabel learning via label set propagation. *IEEE Trans. Knowl. Data Eng.*, 25(3):704–719, 2013.

H. Larochelle and Y. Bengio. Classification Using Discriminative Restricted Boltzmann Machines. In *ICML*, pp. 536–543, 2008.

H. Larochelle and I. Murray. The Neural Autoregressive Distribution Estimator. In *AISTATS*, pp. 29–37, 2011.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. In *ICML*, pp. 473–480, 2007.

H. Larochelle, Y. Bengio, and J.P. Turian. Tractable Multivariate Binary Density Estimation and the Restricted Boltzmann Forest. *Neural Computation*, 22(9):2285–2307, 2010.

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

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

R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. In *Workshop on Learning Tractable Probabilistic Models*. 2014a.

R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP*, pp. 3699–3703, 2014b.

R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. *The Journal of Machine Learning Research*, 2015.

R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *CoRR abs/1601.06180*, 2016. (Accepted for publication in TPAMI).

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

A. Rooshenas and D. Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In *ICML*, pp. 710–718, 2014.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

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

A. Vergari, N. Di Mauro, and F. Esposito. Visualizing and understanding sum-product networks. *CoRR abs/1608.08266*, 2016.

J. Wicker, A. Tyukin, and S. Kramer. *A Nonlinear Label Compression and Transformation Method for Multi-label Classification Using Autoencoders*, pp. 328–340. 2016.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pp. 818–833, 2014.

M. Zöhrer, R. Peharz, and F. Pernkopf. Representation learning for single-channel source separation and bandwidth extension. *IEEE/ACM TASLP*, 23(12):2398–2409, 2015.

A DECODING ALGORITHM

Algorithm 1 lists the pseudocode for our decoding procedure as illustrated in Section 4.

Algorithm 1 decodeEmbedding(M, $\mathbf{e}$, a)

Input: an MPN M over $\mathbf{X}$, an embedding $\mathbf{e} \in \mathbb{R}^d$ and a map $a : \mathbf{M}^{\mathbf{e}} \subseteq \mathbf{M} \to \{1, \ldots, d\}$
Output: a sample $\tilde{\mathbf{x}} \sim \mathbf{X}$ decoded from $\mathbf{e}$, according to M
$\tilde{\mathbf{x}} \leftarrow \mathbf{0}_{|\mathbf{X}|}$
$\mathcal{Q} \curvearrowleft \mathsf{root}(M)$ ▷ top-down traversal of M by using a queue $\mathcal{Q}$
while not $\mathsf{empty}(\mathcal{Q})$ **do**
  $n \curvearrowleft \mathcal{Q}$ ▷ process current node
  if $n \in \mathbf{M}^{\mathsf{max}}$ **then** ▷ max node
    $c_{\mathsf{max}} \leftarrow \mathrm{argmax}_{c \in \mathsf{ch}(n)} w_{nc} v_c$ such that $v_c \leftarrow e_{a(c)}$ **if** $c \in \mathbf{M}^{\mathbf{e}}$ **else** $\max_{\mathbf{u} \sim \mathsf{sc}(c)} M_c(\mathbf{u})$
    $\mathcal{Q} \curvearrowleft c_{\mathsf{max}}$
  else if $n \in \mathbf{M}^{\otimes}$ **then** ▷ product node
    $\forall c \in \mathsf{ch}(n) : \mathcal{Q} \curvearrowleft c$
  else ▷ leaf node
    if $n \in \mathbf{M}^{\mathbf{e}}$ **then**
      $\tilde{\mathbf{x}}_{\mathsf{sc}(n)} \leftarrow \mathrm{argmin}_{\mathbf{u} \sim (\mathsf{sc}(n))} D(\phi_n(\mathbf{u}) || e_{a(n)})$
    else ▷ MPEAssignment (inner embedding)
      $\tilde{\mathbf{x}}_{|\mathsf{sc}(n)} \leftarrow \mathrm{argmax}_{\mathbf{u} \sim \mathsf{sc}(n)} M_n(\mathbf{u})$
return $\tilde{\mathbf{x}}$

B DATASETS

The 10 datasets employed come from the freely accessible MULAN[2], MEKA[3], and LABIC[4] repositories. They are real world standard benchmarks for MLC from text, image, sound and biological domains. Subsets of them have been also used in Dembczyński et al. (2012); Antonucci et al. (2013); Kong et al. (2013). They have been binarized as in (Di Mauro et al., 2016) by implementing the Label-Attribute Interdependence Maximization (LAIM) (Cano et al., 2016) discretization method[5].

Table 2 reports the information about the adopted datasets, where N, M and L represent the number of attributes, instances, and possible labels respectively. They are divided into five standard folds. Furthermore, for each dataset $\mathcal{D} = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^M$ the following statistics are also reported: *label cardinality*: $\mathsf{card}(\mathcal{D}) = \frac{1}{M} \sum_{i=1}^M \sum_{j=1}^L y_j^i$, *label density*: $\mathsf{dens}(\mathcal{D}) = \frac{\mathsf{card}(\mathcal{D})}{L}$ and *distinct labels*: $\mathsf{dist}(\mathcal{D}) = |\{\mathbf{y} | \exists (\mathbf{x}^i, \mathbf{y}) \in \mathcal{D}\}|$.

Table 2: Dataset descriptions: number of attributes (N), instances (N), and labels (L).

	domain	N	M	L	card	dens	dist
Arts	text	500	7484	26	1.653	0.063	599
Business	text	500	11214	30	1.598	0.053	233
Cal	music	68	502	174	26.043	0.149	502
Emotions	music	72	593	6	1.868	0.311	27
Flags	images	19	194	7	3.391	0.484	54
Health	text	500	9205	32	1.644	0.051	335
Human	biology	440	3106	14	1.185	0.084	85
Plant	biology	440	978	12	1.078	0.089	32
Scene	images	294	2407	6	1.073	0.178	15
Yeast	biology	103	2417	14	4.237	0.302	198

[2] http://mulan.sourceforge.net/.
[3] http://meka.sourceforge.net/.
[4] http://computer.njnu.edu.cn/Lab/LABIC/LABIC_Software.html.
[5] The processed versions are freely available at https://github.com/nicoladimauro/dcsn.

C LEARNING SPNS

To learn the structure and weights of our SPNs (and hence MPNs), we employ LearnSPN-b Vergari et al. (2015), a variant of LearnSPN. LearnSPN-b splits the data matrix slices always into two, while performing row clustering or checking for RVs independence. With the purpose of slowing down the greedy hierarchical clustering processes, it has proven to obtain simpler and deeper networks without limiting their expressiveness as density estimators. Based on the datasets statistics reported above in Appendix B, we define the same ranges for LearnSPN-b hyperparameters both when we learn our SPNs for the $\mathbf{X}$ and the $\mathbf{Y}$. We set the G-test independence test threshold to 5, we limit the minimum number of instances in a slice to split to 10 and we performed a grid search for the best leaf distribution Laplace smoothing value in $\{0.1, 0.2, 0.5, 1.0, 2.0\}$. We perform all computations in the $\log$ space to avoid numerical issues.

For the SPN learned on the binarized version of MNIST in Section 3 we set the G-test independence test threshold to 20 and the instance threshold to 50 in order to reduce the network size. We then applied the same grid search as above for the leaf Laplace smoothing coefficient.

D SPN MODEL STATISTICS

Statistics for the reference SPN models learned with LearnSPN-b on the $\mathbf{X}$ RVs only are reported in Table 3. Their average (and standard deviations) values over the dataset folds provide information about the network topology and quality: how many nodes are in there (edges + 1), how are they divided into leaves and sum and products and their max depth (as the longest path from the root). The same statistics are reported for the SPNs over RVs $\mathbf{Y}$, then turned in MPNs, in Table 4.

Table 3: Statistics for the SPN models learned by LearnSPN-b on the $\mathbf{X}$ RVs on the ten datasets. Average and standard deviation values across the five folds reported.

	edges	depth	leaves	inner	sum	prod	scopes
Arts	9241.8	20.2	7412.6	1830.2	605.4	1224.8	1053.6
	±175.4	±1.1	±151.7	±56.2	±19.5	±36.8	±18.6
Business	8569.6	23.4	7029.0	1541.6	507.6	1034.0	971.4
	±228.8	±1.7	±170.7	±73.7	±24.7	±49.1	±22.6
Cal	263.0	7.0	219.8	44.2	14.6	29.6	82.6
	±17.0	±0.0	±18.5	±3.6	±1.1	±2.5	±1.1
Emotions	985.8	13.4	724.6	262.2	87.2	175	147.4
	±36.4	±0.9	±20.2	±20.2	±6.9	±13.3	±4.7
Flags	74.0	7.0	54.6	20.4	6.8	13.6	25.6
	±3.9	±0.0	±1.5	±2.5	±1.7	±0.1	±0.5
Health	7209.2	22.2	5917.0	1293.2	427.8	865.4	899.8
	±249.3	±1.1	±247.4	±21.4	±6.4	±15.0	±7.9
Human	15356.6	19.0	11828.6	3529.0	1170.6	2358.4	1479.2
	±228.9	±1.4	±133.8	±98.7	±32.0	±66.8	±28.8
Plant	3493.8	13.8	2741.8	753.0	247.4	505.6	681.8
	±58.6	±1.1	±42.1	±32.8	±10.7	±22.15	±8.9
Scene	14814.6	15.8	11542.6	3273.0	1089.8	2183.2	1025.6
	±169.1	±1.1	±122.9	±59.9	±20.0	±40.0	±21.8
Yeast	2215.0	18.2	1611.2	604.8	199.6	405.2	262.2
	±96.1	±1.1	±72.4	±28.3	±9.4	±19.0	±3.9

The length of the embeddings extracted from such models is the number of inner nodes from Table 3 for the inner embeddings over $\mathbf{X}$. For the embeddings over RVs $\mathbf{Y}$, their length in the full setting shall be considered as the number of all nodes from Table 4.

Table 4: Statistics for the SPN models learned by LearnSPN-b on the $\mathbf{Y}$ RVs on the ten datasets. Average and standard deviation values across the five folds reported.

	edges	depth	leaves	inner	sum	prod	scopes
Arts	495.0	17.8	340.6	155.4	50.2	105.2	74.4
	$\pm$28.5	$\pm$1.1	$\pm$21.6	$\pm$10.8	$\pm$3.9	$\pm$7.0	$\pm$3.5
Business	414.0	18.6	292.6	122.4	40.2	82.2	65.8
	$\pm$18.0	$\pm$0.9	$\pm$18.1	$\pm$5.7	$\pm$1.8	$\pm$3.9	$\pm$2.5
Cal	1840.4	12.6	1428.0	413.4	137.8	275.6	293.6
	$\pm$51.2	$\pm$0.9	$\pm$25.8	$\pm$29.6	$\pm$9.8	$\pm$19.7	$\pm$7.8
Emotions	39.2	7.0	24.6	15.6	5.2	10.4	11.2
	$\pm$4.5	$\pm$0.0	$\pm$2.2	$\pm$2.5	$\pm$1.7	$\pm$0.1	$\pm$0.8
Flags	25.2	5.4	17.8	8.4	2.8	5.6	9.6
	$\pm$4.2	$\pm$0.9	$\pm$1.8	$\pm$2.5	$\pm$0.8	$\pm$1.7	$\pm$0.5
Health	504.2	17.4	355.0	150.2	49.2	101.0	76.4
	$\pm$21.6	$\pm$1.7	$\pm$17.5	$\pm$7.6	$\pm$2.4	$\pm$5.3	$\pm$2.1
Human	118.2	14.2	85.2	34.0	11.0	23.0	25.0
	$\pm$8.2	$\pm$1.1	$\pm$5.4	$\pm$3.8	$\pm$1.6	$\pm$2.2	$\pm$1.6
Plant	80.0	14.6	57.0	24.0	8.0	16.0	20
	$\pm$8.2	$\pm$2.2	$\pm$6.2	$\pm$2.1	$\pm$0.7	$\pm$1.4	$\pm$0.7
Scene	38.4	9.0	24.4	15.0	5.0	10.0	11.0
	$\pm$0.5	$\pm$0.0	$\pm$0.5	$\pm$0.0	$\pm$0.0	$\pm$0.0	$\pm$0.0
Yeast	382.4	14.6	241.2	142.2	46.6	95.6	46.4
	$\pm$33.4	$\pm$0.9	$\pm$22.6	$\pm$12.8	$\pm$4.1	$\pm$8.8	$\pm$4.2

E MORE EXPERIMENT DETAILS AND RESULTS

E.1 TRAINING DETAILS

E.1.1 LEARNING LINEAR PREDICTORS

We learn to predict each target feature independently from the others, both when we employ the L_2-regularized logistic regressor (LR) to predict RV $\mathbf{Y}$ directly and when we use a ridge regressor (RR) to predict the label embeddings.

To select the best value for the regularization parameter we will perform a grid search for LR in the space $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and for RR in the space $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$[6] for each experiment.

E.1.2 LEARNING RBMS

Concerning RBMs, we train them on the $\mathbf{X}$ alone (or on the $\mathbf{Y}$ alone for the kNN experiments) by using the Persistent Constrastive Divergence (PCD) Marlin et al. (2010) algorithm, leveraging the implementation available in scikit-learn. 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 let the number of epochs range in $\{10, 20, 30\}$ since no early stopping criterion was available. We then select the best models according to their pseudo-log likelihoods. To generate embeddings from RBMs, 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 and MPN representations.

E.1.3 LEARNING MADES

For MADEs, following the experimentation reported in (Germain et al., 2015), we employ adadelta to schedule the learning rate during training and fix its decay rate at 0.95; we set the max number of worsening iterations on the validation set to 30 as for RBMs and we employed a batch size of 100 samples. We initialize the weights by employing an SVD-based init scheme.

[6]We leverage the python implementations for LR and RR from the scikit-learn package (`http://scikit-learn.org/`). Note that in scikit-learn the grid parameter for LR has to be interpreted as an inverse regularization coefficient.

Other hyperparameters are optimized by a log-likelihood-wise grid search. The gradient dumping coefficient is searched in $\{10^{-5}, 10^{-7}, 10^{-9}\}$, and we employ once the shuffling of mask and orders. Both ReLus and softplus functions are explored as the non-linearities employed for each hidden neuron. We employ a MADE openly available implementation, ported to python3[7].

We learn architectures of three hidden layers comprising 500 and 1000 (resp. 200 and 500) hidden neurons each for the $\mathbf{X}$ (resp. $\mathbf{Y}$). For each reference model, we extract $\mathbf{E_X}$ embeddings by evaluating all the hidden layer activations ($d = 1500$ and $d = 3000$); for the $\mathbf{E_Y}$ case, however, only the last hidden layer embeddings are actually exploited for the prediction ($d = 200$ and $d = 500$).

E.1.4 LEARNING MANIAC MODELS

Following the experiments in Wicker et al. (2016), we perform a grid search for the following hyperparameters: the number of layers is chosen in $\{2, 3, 4\}$ and the compression factor $\beta \in \{0.7, 0.8, 0.9\}$. We employ the Java implementation freely available in MEKA. For the RF version of MANIAC we build a random forest comprising 100 trees as it has been used in Wicker et al. (2016) (see Appendix E.3.2 for the results of such a model).

We were not able to properly learn MANIAC for one dataset, Cal, for all measures, as a numerical error in MEKA prevented the model evaluation, thereby we removed it in the result Table.

We were also not able to train MANIAC on the $\mathbf{E_X} \rightarrow \mathbf{Y}$ and hence $\mathbf{E_X} \rightarrow \mathbf{E_Y}$ settings because the learned representations were not available through MEKA.

E.2 RECONSTRUCTION ERRORS

In this Section we provide the detailed results for the reconstructions of the input for our SPNs turned into MPNs for each train and test portion of each dataset, averaged by fold. Table 5 (resp. Table 6) reports the results for architectures trained on the $\mathbf{X}$ (resp. $\mathbf{Y}$) and asked to reconstruct their inputs w.r.t these RVs.

Table 5: Average train and test JACcard, HAMming and EXAct match scores for the reconstruction of the original $\mathbf{X}$ representations through our SPN models, turned into MPNs, on each dataset.

		score	Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast
train	full	JAC	99.34	79.76	99.94	99.43	100.00	99.53	99.26	99.52	99.44	99.75
		HAM	99.98	99.65	99.97	99.74	100.00	99.99	99.83	99.93	99.86	99.86
		EXA	93.95	77.50	98.35	83.47	100.00	96.92	52.39	75.89	56.69	87.67
	inner	JAC	39.94	49.18	95.03	81.40	68.09	52.11	60.61	54.96	73.49	89.47
		HAM	99.08	99.35	97.62	90.32	89.74	99.45	89.85	92.70	87.74	93.79
		EXA	13.84	28.03	97.37	01.56	08.50	26.90	00.00	00.00	00.00	00.57
test	full	JAC	99.41	99.72	99.95	99.48	100.00	99.65	99.33	99.60	99.44	99.78
		HAM	99.98	99.99	99.98	99.76	100.00	99.99	99.85	99.94	99.76	99.87
		EXA	94.64	97.19	99.00	83.81	100.00	97.61	55.11	78.62	56.37	88.20
	inner	JAC	37.97	48.03	94.56	79.35	66.88	51.08	59.01	99.44	71.74	88.98
		HAM	99.02	99.31	97.37	89.09	89.34	99.42	89.31	99.76	86.74	93.49
		EXA	13.20	27.84	29.08	01.85	07.76	26.31	00.00	56.37	00.00	00.49

E.3 OTHER RESULTS FOR MLC

E.3.1 JACCARD HAMMING AND EXACT MATCH MEASURES

In this Section we report the additional results for the JACCARD and HAMMING measures in Table 7 and Table 8 respectively. Figures 4 and 5 report the resilience of the decoding scheme for missing at random embedding components for the JACCARD and HAMMING measures, respectively. We employ a euclidean 5-nearest neighbor classifier to perform the decoding step on all our models. These results are reported in the table last rows.

[7] https://github.com/arranger1044/MADE.

Table 6: Average train and test JACcard, HAMming and EXAct match scores for the reconstruction of the original $\mathbf{Y}$ representations through our SPN models, turned into MPNs, on each dataset.

		score	Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast
train	full	JAC	99.94	99.88	98.72	100.00	100.00	99.96	100.00	100.00	100.00	99.95
		HAM	99.99	99.98	99.80	100.00	100.00	99.99	100.00	100.00	100.00	99.98
		EXA	99.80	99.67	72.75	100.00	100.00	99.87	100.00	100.00	100.00	99.73
	inner	JAC	88.76	92.19	55.94	78.52	70.47	93.25	90.35	89.44	96.31	95.29
		HAM	98.75	99.34	92.41	91.52	81.82	99.41	98.55	98.42	98.76	98.32
		EXA	75.77	82.44	00.00	53.41	23.96	84.06	82.30	85.91	92.64	80.89
test	full	JAC	99.93	99.86	98.89	100.00	100.00	99.97	100.00	100.00	100.00	99.93
		HAM	99.99	99.98	99.82	100.00	100.00	99.99	100.00	100.00	100.00	99.97
		EXA	99.75	99.62	76.50	100.00	100.00	99.89	100.00	100.00	100.00	99.62
	inner	JAC	88.42	92.13	51.98	77.89	70.56	93.11	90.39	89.32	99.95	94.81
		HAM	98.69	99.33	91.52	91.15	81.90	99.39	98.55	98.41	99.98	98.12
		EXA	75.46	82.34	00.00	52.12	23.90	83.87	82.38	85.79	99.73	79.39

Table 7: Average test set JACCARD scores. For each setting, best result for a dataset in bold and average ranks in the last column. Results for the 5-NN decoding are shown in the last two row groups.

		Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast	*RANK*
$\rightarrow \mathbf{Y}$	LR	28.48	49.92	17.43	55.78	48.66	41.11	29.44	32.70	65.43	38.59	-
	CRF_{SSVM}	33.61	73.86	19.98	54.48	56.40	62.10	28.96	31.34	66.15	45.47	-
$\mathbf{E_X} \rightarrow \mathbf{Y}$ $\mathbf{X}$	$\text{RBM}_{h=500}$	26.45	47.38	17.52	**58.11**	51.90	38.11	26.69	33.14	71.61	36.70	4.0
	$\text{RBM}_{h=1000}$	27.85	47.81	17.40	57.94	51.64	39.64	29.44	**33.53**	71.73	37.44	3.4
	$\text{RBM}_{h=5000}$	29.16	48.59	17.51	56.91	50.07	40.73	30.30	32.52	69.61	39.25	3.4
	$\text{MADE}_{h=500}$	27.81	46.90	**17.95**	55.92	47.30	41.35	27.84	30.60	68.82	39.58	4.5
	$\text{MADE}_{h=1000}$	29.71	48.50	17.86	55.66	**54.03**	42.84	28.07	31.53	71.49	**40.79**	3.0
	$\text{SPN}_{\text{inner}}$	**31.63**	**53.29**	17.02	56.84	45.24	**43.88**	**31.51**	32.23	**71.87**	39.70	***2.7***
$\mathbf{X} \rightarrow \mathbf{E_Y}$	$\text{MADE}_{h=200}$	5.03	68.56	20.05	29.21	47.97	40.14	2.58	11.31	12.89	42.73	4.5
	$\text{MADE}_{h=500}$	5.08	68.60	20.04	30.02	48.95	38.78	2.47	11.12	15.37	42.82	4.3
	$\text{MANIAC}_{\text{RR}}$	**39.96**	**73.43**	-	49.41	56.51	**60.72**	**33.19**	31.37	54.52	**49.35**	***2.0***
	MPN_{full}	29.30	**73.43**	20.30	**54.30**	**58.18**	57.80	25.86	29.39	61.20	46.83	2.1
	$\text{MPN}_{\text{inner}}$	35.72	70.53	**20.77**	52.08	55.86	55.31	27.61	**33.07**	**69.60**	47.08	***2.0***
$\mathbf{E_X} \rightarrow \mathbf{E_Y}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	6.72	68.40	20.20	34.19	48.02	39.27	3.79	12.58	16.69	42.33	4.6
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	6.79	68.39	20.19	33.82	48.76	39.22	3.92	12.57	17.92	42.39	4.2
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	8.37	68.55	19.81	31.65	48.01	39.22	5.59	11.84	16.95	42.37	4.8
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	8.65	68.41	19.83	33.11	48.44	39.50	5.96	11.34	17.61	42.64	3.9
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{full}}$	33.47	**73.88**	19.52	**54.48**	**57.70**	**60.20**	**28.67**	29.37	63.64	**46.50**	1.8
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{inner}}$	**37.64**	69.98	**20.52**	52.50	56.56	59.28	27.82	**33.24**	**65.20**	46.05	***1.6***
$\mathbf{X} \rightarrow \mathbf{E_{Y}}_{\text{5-NN}}$	$\text{RBM}_{h=100}$	17.59	51.20	20.73	43.26	52.22	32.41	24.03	25.48	70.08	44.41	5.8
	$\text{RBM}_{h=200}$	17.07	47.73	21.85	**53.73**	57.95	38.14	26.52	23.45	60.71	43.91	4.7
	$\text{RBM}_{h=500}$	16.76	46.96	21.64	51.31	**59.19**	36.46	**39.16**	**44.61**	71.07	43.19	3.8
	$\text{MADE}_{h=200}$	35.23	64.91	**22.07**	47.20	54.46	55.29	30.37	28.25	65.52	42.72	4.5
	$\text{MADE}_{h=500}$	37.36	69.04	21.51	47.55	56.79	56.90	32.47	28.66	62.52	45.93	3.8
	MPN_{full}	**45.24**	**73.51**	21.09	52.96	54.08	**61.56**	39.05	38.40	**74.22**	48.07	***2.2***
	$\text{MPN}_{\text{inner}}$	43.11	72.86	20.96	50.79	51.13	59.44	35.50	33.76	73.47	**48.24**	3.2
$\mathbf{E_X} \rightarrow \mathbf{E_{Y}}_{\text{5-NN}}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	34.67	64.42	21.79	48.10	53.96	53.40	29.61	24.94	67.93	42.96	5.0
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	36.57	66.95	20.80	48.60	54.21	58.02	30.82	27.58	64.73	45.63	3.9
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	35.22	64.91	**21.93**	49.24	**55.15**	55.29	29.97	27.70	68.27	43.41	3.6
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	37.04	67.57	20.97	47.15	53.36	58.43	32.27	26.64	65.55	45.48	3.9
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{full}}$	**46.38**	**73.90**	20.56	**53.04**	52.16	**63.81**	**36.36**	**36.86**	**70.27**	**47.90**	***1.8***
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{inner}}$	44.57	73.04	20.28	50.94	50.84	62.29	34.34	33.28	69.37	47.69	2.8

E.3.2 MORE MANIAC RESULTS

In addition to the ridge regressor (RR) employed as the base model in our previous experiments, we also evaluate a much complex regressor as a random forest (RF) in conjunction with MANIAC, as

Table 8: Average test set HAMMING scores. For each setting, best result for a dataset in bold and average ranks in the last column. Results for the 5-NN decoding are shown in the last two row groups.

		Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast	*RANK*
$\rightarrow \mathbf{Y}$	LR	86.67	92.13	65.25	76.70	68.41	92.24	84.72	86.95	87.69	65.17	-
	CRF_{SSVM}	94.93	97.67	84.28	80.91	71.59	96.96	92.13	91.53	91.09	79.22	-
$\mathbf{E_X} \rightarrow \mathbf{Y}$ $\mathbf{X}$	$\text{RBM}_{h=500}$	82.79	91.43	63.13	**78.27**	66.95	90.18	80.36	83.87	89.72	61.51	5.0
	$\text{RBM}_{h=1000}$	84.82	91.41	64.09	78.02	**68.34**	90.69	83.66	85.36	89.92	63.04	3.7
	$\text{RBM}_{h=5000}$	85.90	92.31	64.95	77.82	68.11	91.21	85.64	86.72	89.37	65.51	3.0
	$\text{MADE}_{h=500}$	83.90	92.38	64.61	77.37	67.66	91.18	83.25	85.33	89.17	69.40	4.2
	$\text{MADE}_{h=1000}$	84.82	93.00	**65.04**	77.03	68.16	91.48	83.68	85.56	90.14	**70.52**	***2.5***
	$\text{SPN}_{\text{inner}}$	**86.10**	**94.12**	62.25	77.60	66.87	**91.76**	**87.39**	**86.92**	**90.20**	67.54	***2.5***
$\mathbf{X} \rightarrow \mathbf{E_Y}$	$\text{MADE}_{h=200}$	93.80	97.17	86.14	73.97	67.11	95.81	91.54	91.09	82.86	77.98	3.2
	$\text{MADE}_{h=500}$	93.80	97.17	86.06	74.08	67.03	95.82	91.54	91.09	82.51	77.82	3.4
	$\text{MANIAC}_{\text{RR}}$	94.27	97.51	-	78.55	70.06	96.64	89.95	88.68	85.88	77.54	3.2
	MPN_{full}	**94.80**	**97.62**	**86.25**	**80.69**	**73.20**	**96.81**	**92.09**	**91.69**	**91.14**	**79.34**	***1.0***
	$\text{MPN}_{\text{inner}}$	92.26	97.28	85.62	77.71	70.35	95.78	89.44	89.35	89.67	74.47	3.8
$\mathbf{E_X} \rightarrow \mathbf{E_Y}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	93.83	97.17	**86.15**	74.39	66.43	95.85	**91.53**	**91.17**	82.78	77.80	3.3
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	93.83	97.17	86.07	74.33	66.20	95.87	**91.53**	91.14	82.50	77.62	4.0
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	93.86	97.18	**86.15**	73.13	66.73	95.85	**91.53**	91.07	83.02	77.89	3.1
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	93.86	97.18	86.05	73.27	67.11	95.58	**91.53**	91.08	82.80	77.73	3.6
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{full}}$	**94.93**	**97.68**	86.01	**79.99**	**73.36**	**96.96**	91.16	91.00	**89.77**	**78.94**	***2.2***
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{inner}}$	92.78	97.21	85.27	77.66	70.94	96.27	89.34	89.18	88.20	74.16	4.0
$\mathbf{X} \rightarrow \mathbf{E_{Y}}_{\text{5-NN}}$	$\text{RBM}_{h=100}$	91.34	95.38	84.83	73.35	68.59	94.12	85.82	86.35	89.83	78.30	5.7
	$\text{RBM}_{h=200}$	91.28	95.06	84.64	78.21	71.57	94.67	86.86	86.37	86.52	78.35	5.1
	$\text{RBM}_{h=500}$	91.18	95.00	84.53	78.22	**73.13**	94.57	**90.64**	**90.45**	90.15	78.40	3.8
	$\text{MADE}_{h=200}$	92.84	96.62	85.16	76.05	70.62	95.98	88.83	87.41	88.17	77.43	4.4
	$\text{MADE}_{h=500}$	92.80	97.05	**85.50**	76.78	71.16	96.23	89.31	87.51	86.96	76.01	4.0
	MPN_{full}	**94.04**	**97.60**	84.96	**79.42**	69.22	**96.78**	90.56	89.38	**91.15**	**78.88**	***1.9***
	$\text{MPN}_{\text{inner}}$	93.62	97.52	85.01	78.69	66.28	96.49	89.82	88.36	90.75	78.09	3.1
$\mathbf{E_X} \rightarrow \mathbf{E_{Y}}_{\text{5-NN}}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	92.62	96.56	85.22	76.02	70.77	95.81	88.97	86.85	89.02	76.84	4.6
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	92.62	96.86	**85.35**	76.55	69.88	96.32	89.02	87.23	87.84	75.87	4.2
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	92.78	96.62	85.04	76.73	**70.82**	95.98	88.91	87.38	89.17	76.53	3.9
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	92.67	96.94	**85.35**	77.03	69.68	96.37	89.29	87.51	88.20	75.68	3.5
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{full}}$	**94.16**	**97.64**	84.81	**79.28**	67.08	**96.95**	**89.90**	**89.06**	**89.66**	**78.54**	***1.8***
	$\text{SPN}_{\text{inner}} \rightarrow \text{MPN}_{\text{inner}}$	93.82	97.53	84.78	78.39	66.58	96.75	89.39	88.33	89.25	77.66	2.8

suggested in (Wicker et al., 2016). The rationale behind this is that a linear model, such as RR, could be at disadvantage on a compressed representation space, like those learned by MANIAC. Results for the JACCARD, HAMMING and EXACT MATCH scores are reported in Table 10 along with our previous results of MPN embeddings employing RR, for the $\mathbf{X} \rightarrow \mathbf{E_Y}$. The performance of a linear models on our embeddings is favorably comparable to that of a non-linear one on MANIAC embeddings, proving the efficacy of MPN as feature extractors.

Table 9: Average test set EXACT MATCH. For each setting, best result for a dataset in bold and average ranks in the last column. Results for the 5-NN decoding are shown in the last two row groups.

		Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast	*RANK*
$\to \mathbf{Y}$	LR	7.00	27.31	0.00	23.78	9.81	14.14	10.11	19.23	46.36	7.20	-
	CRF_{SSVM}	25.33	58.68	0.00	30.18	14.98	49.40	22.54	24.34	60.74	10.72	-
$\mathbf{X} \to \mathbf{Y}$	$\text{RBM}_{h=500}$	6.37	23.44	0.00	**27.65**	6.15	11.09	5.37	13.70	55.09	6.87	4.5
	$\text{RBM}_{h=1000}$	6.02	24.49	0.00	26.81	7.71	12.29	8.47	15.64	56.00	6.87	3.6
	$\text{RBM}_{h=5000}$	6.52	24.59	0.00	25.80	8.76	13.26	11.01	17.90	54.09	6.62	3.3
	$\text{MADE}_{h=500}$	5.90	24.37	0.00	23.95	7.70	15.53	8.40	14.82	55.25	6.82	4.6
	$\text{MADE}_{h=1000}$	7.79	22.15	0.00	24.45	**9.76**	17.24	8.53	16.35	**59.78**	**8.06**	2.7
	$\text{SPN}_{\text{inner}}$	**10.37**	**30.03**	0.00	24.62	8.70	**19.88**	**15.03**	**18.41**	56.95	6.95	***1.9***
$\mathbf{X} \to \mathbf{E_Y}$	$\text{MADE}_{h=200}$	3.24	53.25	0.00	10.11	1.58	28.78	1.96	6.55	9.80	3.93	4.4
	$\text{MADE}_{h=500}$	3.30	53.23	0.00	10.28	3.63	27.53	1.88	5.93	11.30	4.10	4.3
	$\text{MANIAC}_{\text{RR}}$	**25.70**	56.51	-	22.11	14.51	45.23	23.08	24.75	45.37	**12.41**	2.1
	MPN_{full}	22.45	**58.32**	0.00	**29.51**	**15.46**	**46.27**	21.34	23.72	56.54	12.04	***2.0***
	$\text{MPN}_{\text{inner}}$	25.18	54.50	0.00	25.97	13.44	38.79	**23.66**	**31.29**	**66.51**	12.04	2.1
$\mathbf{E_X} \to \mathbf{E_Y}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	4.94	53.29	0.00	9.94	2.08	28.09	2.44	5.21	10.84	3.31	4.1
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	5.09	53.26	0.00	9.27	2.59	28.04	2.67	4.80	10.63	3.60	4.3
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	4.73	53.29	0.00	8.26	1.56	27.98	2.73	4.80	9.72	3.93	4.8
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	5.17	53.24	0.00	8.42	3.63	28.04	3.28	5.22	9.80	3.85	3.9
	$\text{SPN}_{\text{inner}} \to \text{MPN}_{\text{full}}$	25.97	**58.80**	0.00	**29.34**	**16.45**	**48.14**	22.02	24.04	55.80	**12.86**	***1.6***
	$\text{SPN}_{\text{inner}} \to \text{MPN}_{\text{inner}}$	**27.72**	53.96	0.00	26.14	14.47	43.41	**23.66**	**31.61**	**62.11**	12.20	1.7
$\mathbf{X} \to \mathbf{E_{Y\,5\text{-NN}}}$	$\text{RBM}_{h=100}$	10.06	20.11	0.00	13.83	9.30	14.36	8.88	18.81	66.97	11.34	5.4
	$\text{RBM}_{h=200}$	9.50	13.71	0.00	27.48	12.90	24.74	11.97	19.42	58.16	10.84	4.8
	$\text{RBM}_{h=500}$	8.51	12.18	0.00	23.26	**17.00**	21.84	33.03	**42.53**	67.96	10.54	3.8
	$\text{MADE}_{h=200}$	24.31	44.58	0.00	17.36	15.98	41.65	22.50	26.08	60.24	8.31	4.4
	$\text{MADE}_{h=500}$	25.08	50.82	0.00	17.19	14.93	43.55	24.50	26.08	56.66	8.68	4.3
	MPN_{full}	**34.46**	**57.49**	0.00	**25.46**	8.31	**47.78**	**33.16**	35.49	**69.75**	**14.52**	***2.0***
	$\text{MPN}_{\text{inner}}$	29.79	56.02	0.00	22.93	5.71	43.78	28.94	30.17	67.59	12.90	3.2
$\mathbf{E_X} \to \mathbf{E_{Y\,5\text{-NN}}}$	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=200}$	24.21	44.51	0.00	18.21	**15.92**	40.30	22.09	23.44	63.06	9.14	4.7
	$\text{MADE}_{h_{\mathbf{X}}=500,h_{\mathbf{Y}}=500}$	25.11	48.76	0.00	18.38	10.75	44.87	23.08	25.69	58.57	10.21	3.8
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=200}$	24.88	44.58	0.00	20.23	13.87	41.65	22.05	25.28	63.23	9.39	4.1
	$\text{MADE}_{h_{\mathbf{X}}=1000,h_{\mathbf{Y}}=500}$	25.53	49.79	0.00	16.35	14.43	44.85	24.53	24.66	59.45	9.51	3.8
	$\text{SPN}_{\text{inner}} \to \text{MPN}_{\text{full}}$	**35.98**	**57.79**	0.00	**25.46**	7.24	**50.13**	**28.75**	**34.25**	**63.60**	**14.81**	***1.6***
	$\text{SPN}_{\text{inner}} \to \text{MPN}_{\text{inner}}$	31.93	56.03	0.00	23.44	6.25	47.58	25.94	30.06	61.36	13.16	2.7

Table 10: Average test JACcard, HAMming and EXAct match scores for MANIAC employing a random forest as base model (RF) and our MPN models in the $\mathbf{X} \to \mathbf{E_Y}$ setting.

		Arts	Business	Cal	Emotions	Flags	Health	Human	Plant	Scene	Yeast
JAC	$\text{MANIAC}_{\text{RF}}$	**42.04**	72.61	-	52.81	53.62	**63.08**	**29.37**	31.22	62.26	**49.56**
	MPN_{full}	29.30	**73.43**	20.30	**54.30**	**58.18**	57.80	25.86	29.39	61.20	46.83
	$\text{MPN}_{\text{inner}}$	35.72	70.53	**20.77**	52.08	55.86	55.31	27.61	**33.07**	**69.60**	47.08
HAM	$\text{MANIAC}_{\text{RF}}$	93.99	97.49	-	76.53	72.10	**96.89**	90.16	89.10	88.06	77.63
	MPN_{full}	**94.80**	**97.62**	**86.25**	**80.69**	**73.20**	96.81	**92.09**	**91.69**	**91.14**	**79.34**
	$\text{MPN}_{\text{inner}}$	92.26	97.28	85.62	77.71	70.35	95.78	89.44	89.35	89.67	74.47
EXA	$\text{MANIAC}_{\text{RF}}$	**30.51**	55.47	-	24.61	10.93	**48.15**	18.57	23.52	54.45	**12.57**
	MPN_{full}	22.45	**58.32**	0.00	**29.51**	**15.46**	46.27	21.34	23.72	56.54	12.04
	$\text{MPN}_{\text{inner}}$	25.18	54.50	0.00	25.97	13.44	38.79	**23.66**	**31.29**	**66.51**	12.04

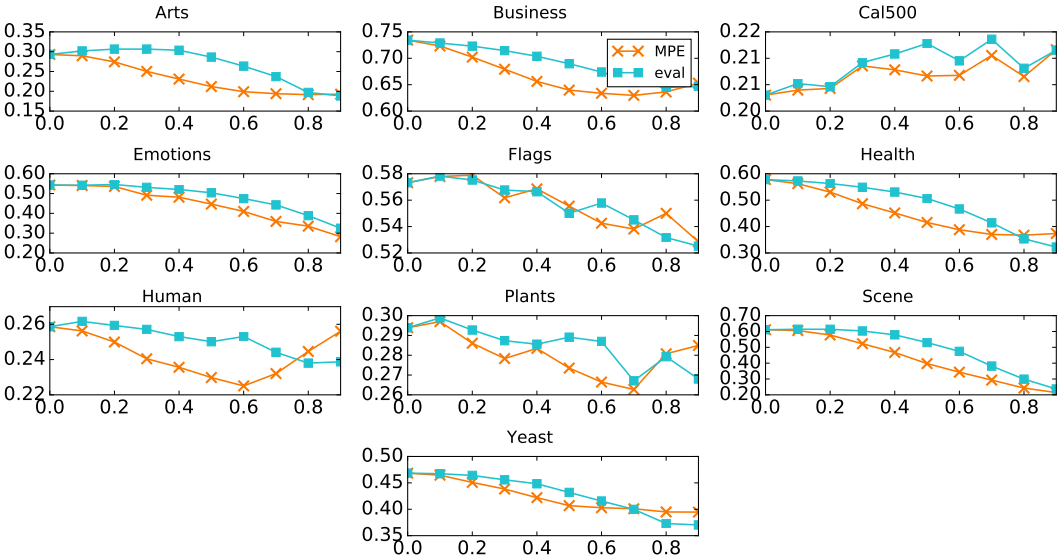


Figure 4: Average test JACCARD scores (y axis) obtained by imputing different percentages of missing random embedding components (x axis) for the $\mathbf{X} \rightarrow \mathbf{E}_{\mathbf{Y}}$ setting on all datasets by employing MPE inference (orange crosses) or the bottom-up evaluation imputation schemes (blue squares).

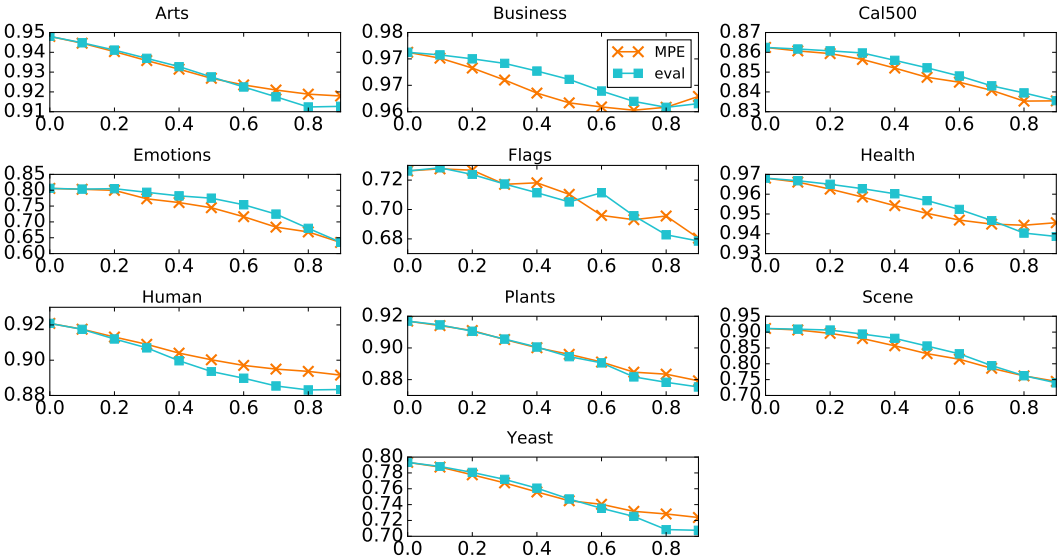


Figure 5: Average test HAMMING scores (y axis) obtained by imputing different percentages of missing random embedding components (x axis) for the $\mathbf{X} \rightarrow \mathbf{E}_{\mathbf{Y}}$ setting on all datasets by employing MPE inference (orange crosses) or the bottom-up evaluation imputation schemes (blue squares).