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

Antonio Vergari (*), Robert Peharz (**), Nicola Di Mauro (*), Floriana Esposito (*)
University of Bari, Italy (*), Medical University of Graz, Austria (**)
firstname.lastname@uniba.it (*), robert.peharz@medunigraz.at (**)

ABSTRACT

Sum-Product Networks (SPNs) are deep density estimators allowing exact and tractable inference. While up to now SPNs have been employed as black-box inference machines, we exploit them as feature extractors for unsupervised Representation Learning. Representations learned by SPNs are rich probabilistic and hierarchical part-based features. SPNs converted into Max-Product Networks (MPNs) provide a way to decode these representations back to the original input space. In extensive experiments, SPN and MPN encoding and decoding schemes prove highly competitive for Multi-Label Classification tasks.

1 INTRODUCTION AND MOTIVATION

Sum-Product Networks (SPNs) (Poon & Domingos, 2011) are deep generative models that, by decomposing a probability distribution into sums and products, allow *exact* and *tractable* computation of a range of queries such as the partition function, marginals and conditionals. SPNs have been successfully applied to computer vision (Gens & Domingos, 2012; Amer & Todorovic, 2015), speech (Peharz et al., 2014) and language modeling (Cheng et al., 2014), however always as "black box" inference machines. Here look at the inner workings of SPNs and exploit them as feature extractors for Representation Learning (RL), showing how they learn rich and hierarchical representations that can be effectively employed in several learning schemes for predictive tasks.

Generative models trained in an unsupervised way have been largely employed as feature extractors. This is the case of many seminal works with Restricted Boltzmann Machines (Ranzato & Hinton, 2010; Coates et al., 2011; Marlin et al., 2010) in which the extracted representations are then employed for supervised learning, e.g. fed into a Logistic Regressor (LR) or SVM classifier, or used to initialize another neural architecture (Hinton & Salakhutdinov, 2006). More recent examples of this two-stage learning approach for RL can be found in several computer vision works (Girshick et al., 2014; Simonyan & Zisserman, 2014; Kümmerer et al., 2014).

By having a mechanism to *decode* back the learned representation to the original space, one could employ such RL approaches to more complex tasks such as Multi-label classification (MLC): first by learning a model to predict label embeddings and then decoding this model predictions into the actual label values (Bhatia et al., 2015; Akata et al., 2013). The advantage would lie in learning an easier to predict target space (the one of label embeddings) instead of the original one.

By turning an SPN into a Max-Product Network (MPN), we show how Most Probable Explanation (MPE) inference (Poon & Domingos, 2011; Peharz et al., 2016) can be leveraged to build such a decoding procedure. In such a way, without training the network with the aim to reconstruct its input, we are able to use them as a kind of autoencoders in the aforementioned learning scenarios. Moreover, we extend this procedure to decode embeddings with missing components.We empirically evaluate the features extracted by SPNs and MPNs by visualizing them back into the input space and the proposed decoding procedure and its resilience by employing them for MLC when encoding the feature space, the target space, or both.

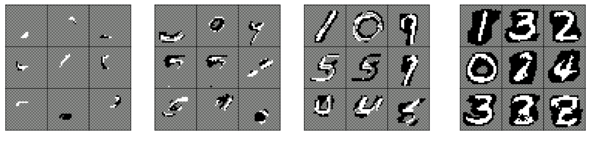
Figure 1: Visualizing features learned by an SPN trained on a binarized version of MNIST clustered by similar scope lengths.

2 Encoding and decoding

An SPN S over a set of random variables (RVs) $\mathbf{X} = \{X_i\}_{i=1}^{p}$ is a computational graph defined by a rooted DAG. Let n be an input node of S, also called *leaf* node, it defines a tractable distribution ϕ_n over its *scope* $\mathsf{sc}(n) \subseteq \mathbf{X}$. If n is an *inner node*, it computes either a *weighted sum*, parametrized by a set of non-negative weights $\mathbf{w}_n$, or a *product* over its children, denoted as $\mathsf{ch}(n)$. We assume S to be *complete*, *decomposable* and normalized (Poon & Domingos, 2011; Peharz et al., 2015), hence the output value of its root, after $\mathbf{X} = \mathbf{x}$ is observed as the network input, is a valid evaluation for $p(\mathbf{X} = \mathbf{x})$. Moreover, for each node $n \in S$, S_n, the sub-network rooted at node n, defines a valid probability distribution $p_{\mathbf{w}_n}$ over the scope of n, defined as $\mathsf{sc}(n) = \cup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$. Such a probability is the output value (activation) of the node n, denoted as $S_n(\mathbf{x}_{|\mathsf{sc}(n)})$. An MPN M over $\mathbf{X}$ can be built from S by replacing each sum node in S by a *max node* in M and each leaf distribution by a maximizing distribution (Poon & Domingos, 2011; Peharz et al., 2016). SPNs and MPNs can be interpreted as peculiar neural networks (ANNs) in which nodes are *labelled* by the scope function, enabling a *direct encoding* of the input, connections are sparse due to their topology being *constrained* and each node retains a fully probabilistic semantics (Vergari et al., 2016).

We are interested in *encoding* a sample $\mathbf{x}^i \sim \mathbf{X}$ into a continuous d-dimensional *embedding* $\mathbf{e}^i \in \mathbf{E}_{\mathbf{X}} \subseteq \mathbb{R}^d$. To find a function $f_S : \mathbf{X} \rightarrow \mathbf{E}_{\mathbf{X}}$ we consider SPNs, but similar considerations hold also for MPNs. By interpreting an SPN S as an ANN, and given a set of its nodes $\mathcal{N} = \{n_j\}_{j=1}^{d}$, we construct our embedding as $e_j^i = S_{n_j}(\mathbf{x}_{|\mathsf{sc}(n_j)}) = p_{\mathbf{w}_{n_j}}(\mathbf{x}_{|\mathsf{sc}(n_j)})$. Each embedding component (feature) represents the probability to see that sample according to a *marginal* distribution over a node scope. SPN features can also be seen as *part-based filters* operating over sub-spaces given by the node scopes. This is clearly visible if such features are visualized in the input space. By solving an inverse problem similar to Erhan et al. (2009) by approximate MPE inference on SPNs learned on image samples, the features learned by an SPN appear as meaningful hierarchical representations at different levels of complexity. In Figure 1 blobs, shape contours and parts, associated to nodes extracted from an SPN learned on a binary version of MNIST (Vergari et al., 2016), are easily recognizable. These visualizations suggest that it is the scope information that correlates to the level of abstraction of a feature, rather than the simple depth of a node, as it happens for classical ANNs. In this work, to build $\mathcal{N}$, we consider either all the nodes in the network (full embeddings) or only the inner ones. In the second case we discard the potentially uninformative univariate distribution at the leaves.

For the decoding phase we need to find an inverse function $g\colon \mathbf{E}_{\mathbf{X}} \rightarrow \mathbf{X}$ such that $\mathbf{x}^i \approx \hat{\mathbf{x}}^i = g(f(\mathbf{x}^i))$. To do so we exploit an MPN M and devise g_M to mimic the algorithm to compute the MPE assignment of a probabilistic query over M. We note that when a sample $\mathbf{x}^i$ is fully observed, then the computation of $M(\mathbf{x}^i)$ activates only one maximal path in the network that can be traced back by a Viterbi-like procedure to a set of leaves whose scopes are a partition of $\mathbf{X}$ (Peharz et al., 2016). Therefore, if a full embedding $\mathbf{e}_M^i$ is available, and if we follow the same procedure to trace the maximal path and equip the reached leaves by a decoder procedure, then we can obtain $\hat{\mathbf{x}}^i$ (see Appendix). In practice, we are interested in decoding embeddings that have been predicted by some learned model. 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: $\tilde{\mathbf{x}}_{|\mathsf{sc}(n)}^i = \operatorname{argmin}_{\mathbf{u} \sim \mathsf{sc}(n)} D(\phi_n(\mathbf{u}) || \mathbf{e}_{M_n}^i)$. In our experiments we will employ an L_1 distance $|\phi_n(\mathbf{u}) - \mathbf{e}_{M_n}^i|$. While it is a cheap heuristics, it proved surprisingly effective in our experiments.

When not all components of $\mathbf{e}^i$ are available, it would be possible to decode it back *completely* if the missing components are associated to nodes whose children components are available. The imputation can be done by evaluating these nodes in M by a forward pass. Otherwise, we suggest missing activation of a node n to be imputed by applying MPE inference to estimate $M_n(\mathbf{x}^i)$. For example, when dealing with inner embeddings we impute leaf node values by their MPE state.

3 Experiments

We evaluate the feature extracted unsupervisedly by SPNs/MPNs and the proposed decoding schemes on MLC tasks: for each sample $\mathbf{x} \sim \mathbf{X}$ one has to predict binary label array $\mathbf{y} \sim \mathbf{Y}$. In the simplest case one would learn a predictive model p, to predict the labels given the original features, $\mathbf{X} \overset{\mathsf{p}}{\Rightarrow} \mathbf{Y}$. We define three more learning scenarios in which to employ embeddings: I) learning p on the features encoded by a model m, $(\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{p}}{\Rightarrow} \mathbf{Y}$; II) learning p to predict the label space $\mathbf{E_Y}$ encoded by a model n, then use n to decode them back to actual $\mathbf{Y}$: $(\mathbf{X} \overset{\mathsf{p}}{\Rightarrow} \mathbf{E_Y}) \xrightarrow{\mathsf{n}} \mathbf{Y}$; III) combining the previous two approaches: $((\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{p}}{\Rightarrow} \mathbf{E_Y}) \xrightarrow{\mathsf{n}} \mathbf{Y}$. We employ for p linear predictors to show 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 is a ridge regressor (RR).

Concerning models m, we employ RBMs with up to 5000 hidden units, tractable probabilistic autoencoders as MADEs (Germain et al., 2015) up to 3 layers deep, and SPNs with inner embeddings. For the n models, we exploit the corresponding routines for MADEs, and up to 3 layers of non-probabilistic autoencoders (AEs) like stacked AEs for label embeddings (SDA) (Wicker et al., 2016), contractive AEs (CAE) (Rifai et al., 2011) and denoising AEs (DAE) (Vincent et al., 2010). Lastly, we use MPNs with either full or inner embeddings (in which case we compute the leaf MPE state).

We learn all models independently for $\mathbf{X}$ and $\mathbf{Y}$. We use LearnSPN-b (Vergari et al., 2015) to learn both the structure and weights of our SPNs, and hence MPNs. We employ ten standard benchmark datasets for MLC (Di Mauro et al., 2016) and perform a 5-fold cross validation. We score all the models by 3 differently forgiving metrics: jaccard, hamming and exact match scores (Dembczyński et al., 2012) and we report the average relative improvement of each model w.r.t. the $\mathbf{X} \overset{\text{LR}}{\Rightarrow} \mathbf{Y}$ baseline. In Table 1 we also report the performance of a fully supervised method for MLC such as max-margin structured SVMs employing CRFs (SSVM). For all settings and measures the MPN and SPN embeddings prove to be highly competitive against all other models. Either inner or full embeddings achieve the highest improvements in many cases. Overall, the proposed decoding procedure outperforms several of the autoencoder architectures used; employing MPE inference to cope with missing leaf activations helps expecially for the jaccard score.

Additionally, we evaluate the resilience of our decoding procedure to handle missing data more in detail for scenario II). We increase the percentage of missing components from 0 (full embeddings) to 90 and we employ two strategies: either impute a node activation by its MPE value only or evaluate the MPN forward before performing MPE with the updated activations (eval). Figure 2 depicts the exact match score performances for these increasingly missing embeddings for 9 datasets (on one all models score 0). Evaluating the MPN again before imputing a node value not only consistently achieves the best performances than not doing it, but shows a quite slow performance decrease up to 40% missing components, on the majority of datasets. This suggests that the redundancy of activations not present in the maximal decoding path can still help the decoding procedure.

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.

	JAC	HAM	EXA
$\mathbf{X} \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	0.00	0.00	0.00
$\mathbf{X} \overset{\text{SSVM}}{\Longrightarrow} \mathbf{Y}$	+15.83	+9.94	+103.90
$(\mathbf{X} \xrightarrow{\text{RBM}} \mathbf{E_X}) \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	+1.46	+0.20	-1.62
$(\mathbf{X} \xrightarrow{\text{MADE}} \mathbf{E_X}) \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	+2.57	**+0.60**	+2.99
$(\mathbf{X} \xrightarrow{\text{CAE}} \mathbf{E_X}) \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	-0.15	-0.69	+4.13
$(\mathbf{X} \xrightarrow{\text{DAE}} \mathbf{E_X}) \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	+0.70	-0.05	+4.17
$(\mathbf{X} \xrightarrow{\text{SPN}_\text{inner}} \mathbf{E_X}) \overset{\text{LR}}{\Longrightarrow} \mathbf{Y}$	**+3.54**	+0.50	**+17.18**
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MADE}} \mathbf{Y}$	-30.42	+7.04	-28.02
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{SDA}} \mathbf{Y}$	+5.96	+5.07	+95.78
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{CAE}} \mathbf{Y}$	+7.60	+9.53	+78.81
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{DAE}} \mathbf{Y}$	+13.39	+9.78	**+102.22**
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MPN}_\text{full}} \mathbf{Y}$	+11.65	**+10.45**	+96.30
$(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MPN}_\text{inner}} \mathbf{Y}$	**+15.19**	+7.61	+98.58
$((\mathbf{X} \xrightarrow{\text{MADE}} \mathbf{E_X}) \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MADE}} \mathbf{Y}$	-27.15	+6.94	-25.14
$((\mathbf{X} \xrightarrow{\text{CAE}} \mathbf{E_X}) \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{CAE}} \mathbf{Y}$	+5.21	+9.27	+79.20
$((\mathbf{X} \xrightarrow{\text{DAE}} \mathbf{E_X}) \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{DAE}} \mathbf{Y}$	+13.97	+9.88	+98.25
$((\mathbf{X} \xrightarrow{\text{SPN}_\text{full}} \mathbf{E_X}) \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MPN}_\text{full}} \mathbf{Y}$	+14.52	**+9.97**	+106.62
$((\mathbf{X} \xrightarrow{\text{SPN}_\text{full}} \mathbf{E_X}) \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MPN}_\text{inner}} \mathbf{Y}$	**+15.98**	+7.50	**+106.65**

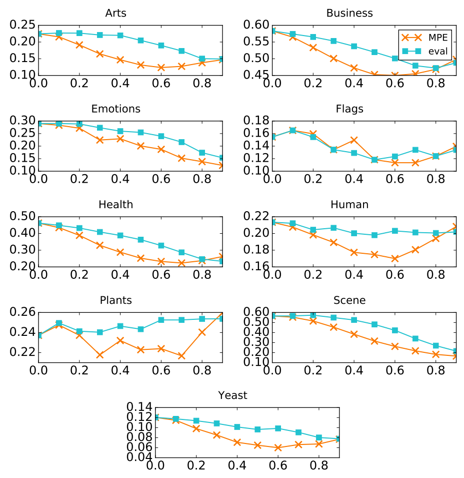

Figure 2: exact match scores (y axis) while increasing the percentage of missing components (x axis) for the $(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E_Y}) \xrightarrow{\text{MPN}_\text{full}} \mathbf{Y}$ setting, on 9 datasets, while imputing the missing embedding components only by MPE inference or re-evaluating the network before (eval).

REFERENCES

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.

Mohamed Amer and Sinisa Todorovic. Sum product networks for activity recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 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.

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.

Alberto Cano, José María Luna, Eva L. Gibaja, and Sebastián Ventura. LAIM discretization for multi-label data. *Information Sciences*, 330:370–384, 2016.

Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Adam Chai. Language modeling with Sum-Product Networks. In *INTERSPEECH 2014*, pp. 2098–2102, 2014.

Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single layer networks in unsupervised feature learning. *AISTATS 2011*, 2011.

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

N. Di Mauro, A. Vergari, and F. Esposito. Multi-label classification with cutset networks. In *International Conference on Probabilistic Graphical Models (PGM)*, 2016.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network. *ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.*, 2009.

Robert Gens and Pedro Domingos. Discriminative Learning of Sum-Product Networks. In *Advances in Neural Information Processing Systems 25*, pp. 3239–3247, 2012.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: masked autoencoder for distribution estimation. *CoRR*, abs/1502.03509, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

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

Matthias Kümmerer, Lucas Theis, and Matthias Bethge. Deep gaze i: Boosting saliency prediction with feature maps trained on imagenet. *arXiv preprint arXiv:1411.1045*, 2014.

Benjamin M Marlin, Kevin Swersky, Bo Chen, and Nando D Freitas. Inductive Principles for Restricted Boltzmann Machine Learning. In *AISTATS 2010*, pp. 509–516, 2010.

Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP2014*, 2014.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. *The Journal of Machine Learning Research*, 2015.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro M. Domingos. On the latent variable interpretation in sum-product networks. *CoRR*, abs/1601.06180, 2016. URL http://arxiv.org/abs/1601.06180.

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

M. Ranzato and Geoffrey E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2551–2558. IEEE, 2010.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the Twenty-eight International Conference on Machine Learning, ICML*, 2011.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

A. Vergari, N. Di Mauro, and F. Esposito. Visualizing and understanding sum-product networks. *preprint arXiv*, 2016. URL `https://arxiv.org/abs/1608.08266`.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning. In *ECML-PKDD 2015*, 2015.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1756006.1953039`.

Jörg Wicker, Andrey Tyukin, and Stefan Kramer. *A Nonlinear Label Compression and Transformation Method for Multi-label Classification Using Autoencoders*, pp. 328–340. Springer International Publishing, Cham, 2016. ISBN 978-3-319-31753-3. doi: 10.1007/978-3-319-31753-3_27. URL `http://dx.doi.org/10.1007/978-3-319-31753-3_27`.

A Decoding algorithm

Algorithm 1 lists the pseudocode for our decoding procedure as illustrated in Section 2.

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

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

B Datasets

The 10 datasets employed come from the freely accessible MULAN[1], MEKA[2], and LABIC[3] 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[4].

Table 1 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 1: Dataset descriptions: number of attributes (N), instances (M), 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

[1] `http://mulan.sourceforge.net/`.

[2] `http://meka.sourceforge.net/`.

[3] `http://computer.njnu.edu.cn/Lab/LABIC/LABIC_Software.html`.

[4] 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 Figure 1 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 2. 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 3.

Table 2: 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 2 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 3.

Table 3: 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
	±28.5	±1.1	±21.6	±10.8	±3.9	±7.0	±3.5
Business	414.0	18.6	292.6	122.4	40.2	82.2	65.8
	±18.0	±0.9	±18.1	±5.7	±1.8	±3.9	±2.5
Cal	1840.4	12.6	1428.0	413.4	137.8	275.6	293.6
	±51.2	±0.9	±25.8	±29.6	±9.8	±19.7	±7.8
Emotions	39.2	7.0	24.6	15.6	5.2	10.4	11.2
	±4.5	±0.0	±2.2	±2.5	±1.7	±0.1	±0.8
Flags	25.2	5.4	17.8	8.4	2.8	5.6	9.6
	±4.2	±0.9	±1.8	±2.5	±0.8	±1.7	±0.5
Health	504.2	17.4	355.0	150.2	49.2	101.0	76.4
	±21.6	±1.7	±17.5	±7.6	±2.4	±5.3	±2.1
Human	118.2	14.2	85.2	34.0	11.0	23.0	25.0
	±8.2	±1.1	±5.4	±3.8	±1.6	±2.2	±1.6
Plant	80.0	14.6	57.0	24.0	8.0	16.0	20
	±8.2	±2.2	±6.2	±2.1	±0.7	±1.4	±0.7
Scene	38.4	9.0	24.4	15.0	5.0	10.0	11.0
	±0.5	±0.0	±0.5	±0.0	±0.0	±0.0	±0.0
Yeast	382.4	14.6	241.2	142.2	46.6	95.6	46.4
	±33.4	±0.9	±22.6	±12.8	±4.1	±8.8	±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 $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$[5] 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.

[5]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[6].

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 SAE 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.

We were not able to properly learn SAEs 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 them on the $(\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{LR}}{\Rightarrow} \mathbf{Y}$ and hence $((\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{RR}}{\Rightarrow} \mathbf{E_Y}) \xrightarrow{\mathsf{n}} \mathbf{Y}$ settings because the learned representations were not available through MEKA.

E.1.5 Learning contractive and denoising autoencoders

For both CAE and DAE models, for all settings, we learned architectures with up to three layers for both the encoder and the decoder network.

In particular, we looked at architectures with layers each one made of 500 hidden units for the $(\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{LR}}{\Rightarrow} \mathbf{Y}$ setting, and 200 hidden units in the case of $(\mathbf{X} \overset{\mathsf{RR}}{\Rightarrow} \mathbf{E_Y}) \xrightarrow{\mathsf{n}} \mathbf{Y}$. For the last setting, $((\mathbf{X} \xrightarrow{\mathsf{m}} \mathbf{E_X}) \overset{\mathsf{RR}}{\Rightarrow} \mathbf{E_Y}) \xrightarrow{\mathsf{n}} \mathbf{Y}$ we exploited the former for the m models and the latter as the n models. We modeled the size of the representation layer by performing a grid search among network layers of size compressed by a factor in $\{0.7, 0.8, 0.9\}$.

To train them, we employed Adam as the optimized for gradient descent up to 1000 epochs, stopping it earlier if no improvement was found after 50 epochs on validation data. We looked for the batch size in $\{20, 100\}$. For CAEs we searched for the contractive coefficient in $\{0.001, 0.01, 0.1\}$ while we explored the space of $0.1, 0.2, 0.3$ as the percentages of flipping a bit (noise) in the input representation for DAEs.

E.2 More results for missing embedding components

Figures 1 and 2 report the resilience of the decoding scheme for missing at random embedding components for the jaccard and hamming measures, respectively.

[6]`https://github.com/arranger1044/MADE`.

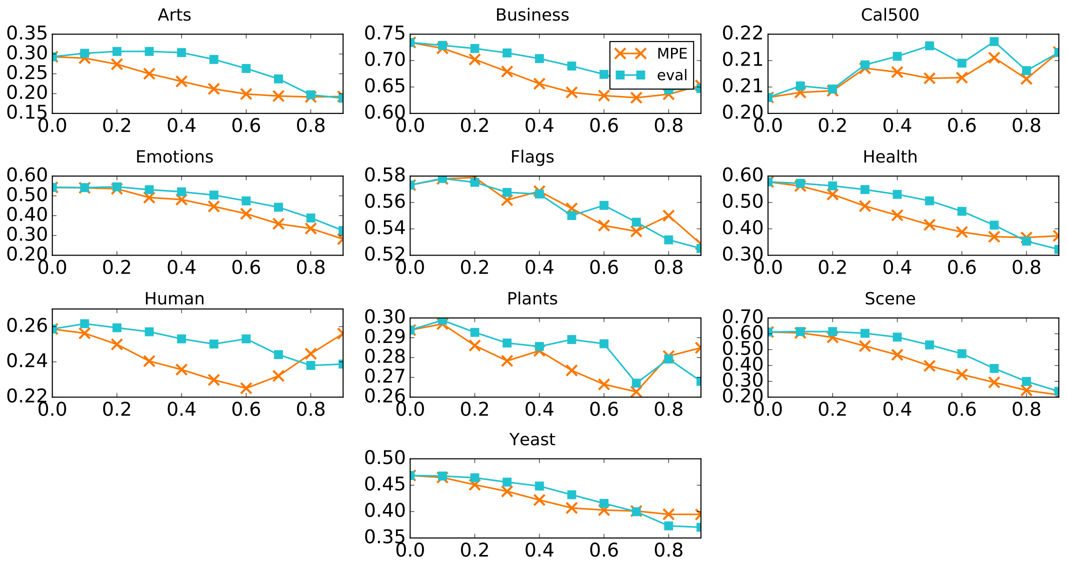

Figure 1: Average test jaccard scores (y axis) obtained by imputing different percentages of missing random embedding components (x axis) for the $(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E}_{\mathbf{Y}}) \xrightarrow{\text{n}} \mathbf{Y}$ setting on all datasets by employing MPE inference (orange crosses) or the bottom-up evaluation imputation schemes (blue squares).

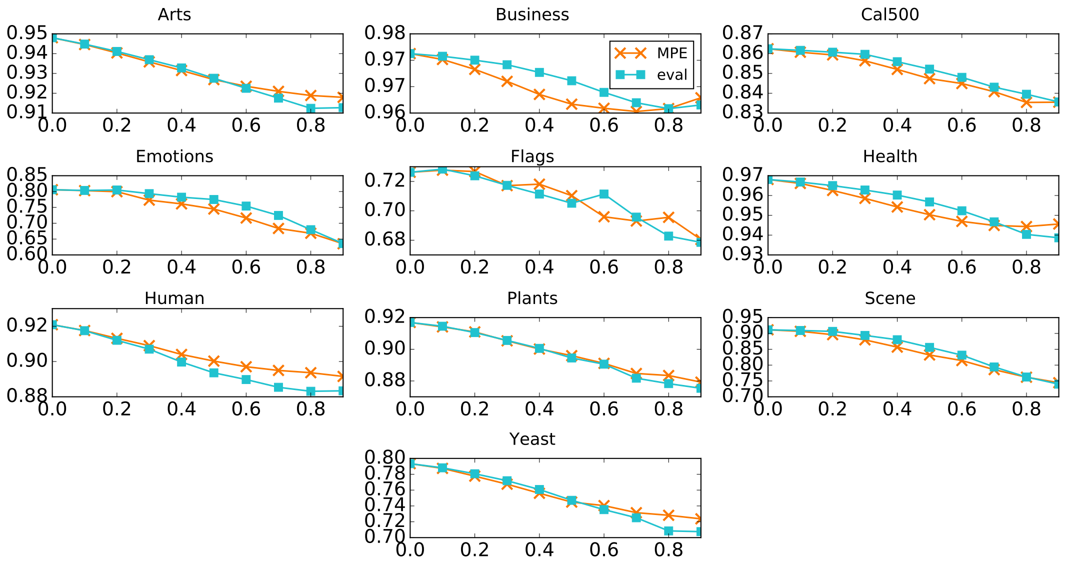

Figure 2: Average test hamming scores (y axis) obtained by imputing different percentages of missing random embedding components (x axis) for the $(\mathbf{X} \overset{\text{RR}}{\Longrightarrow} \mathbf{E}_{\mathbf{Y}}) \xrightarrow{\text{n}} \mathbf{Y}$ setting on all datasets by employing MPE inference (orange crosses) or the bottom-up evaluation imputation schemes (blue squares).