University of Bari
Department of Computer Science

LACAM Laboratory
Machine Learning

Encoding and Decoding Representations with Sum-Product Networks

Antonio Vergari

joint work with:
Robert Peharz, Nicola Di Mauro, Alejandro Molina, Kristian Kersting and Floriana Esposito

November 14th - Bari - AIxIA 2017

Outline

Extending ***Sum-Product Networks*** (**SPNs**) [Poon and Domingos 2011] towards ***Representation Learning*** (**RL**) by equipping them with encoding and decoding routines. Learn a density estimator once and exploit it for predictive tasks later—without retraining—e.g. ***Multi-Label Classification*** (**MLC**).

Dealing with categorical—CAT ***embeddings***—and continuous representations—ACT ***embeddings***—by leveraging the ***probabilistic latent variable semantics*** and the ***neural network interpretation*** of SPNs, respectively.

Density estimation >

Sum-Product Networks >

MPE inference with SPNs >

CAT embeddings >

ACT embeddings >

CAT vs ACT embeddings >

MLC predictions >

Learn *once*, exploit *more than once*

The challenges in the arms race to ***deeply make sense of data*** lie into the ability to effectively make use of ***unlabeled data*** and to efficiently reason about it, i.e. to make ***inference*** about their configurations and relationships

$\Rightarrow$ *how to understand the flow of traffic in a city from historical records, traffic light sensors and camera recordings?*

Density estimation is the unsupervised task of learning an estimator for the joint probability distribution $p(\mathbf{X})$ from i.i.d. samples $\mathcal{D} = \{\mathbf{x}^i\}_{i=1}^m$ over random variables (RVs) $\mathbf{X}$ Given such an estimator, answer a wide range of probabilistic queries:

$\Rightarrow$ *complete evidence*, *marginals*, *conditionals*, *Most Probable Explanaition* (MPE),...

Learn once, exploit it several times philosophy to density estimation: learn one *tractable* probabilistic model in an unsupervised way from data, *then*:

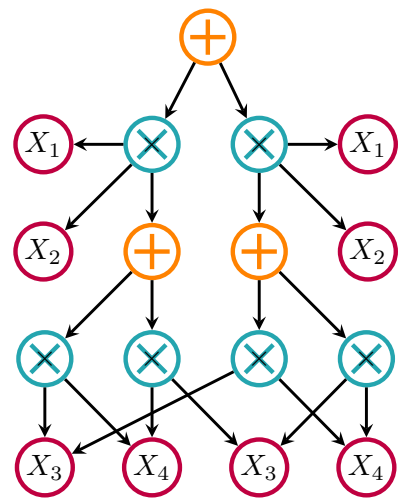
- $\oplus$ perform (several kinds of) ***inference ad libitum***
- $\oplus$ ***exploit it for predictive tasks*** later, without training again

Sum-Product Networks (SPNs)

A *Sum-Product Network* S over RVs $\mathbf{X}$ is defined via rooted weighted DAG consisting of distribution ***leaves*** (network inputs), ***sum*** and ***product*** nodes (inner nodes).

Each sub-network S_n defines an unnormalized probability distribution over the subset of RVs appearing in it, $\mathsf{sc}(n) \subseteq \mathbf{X}$.

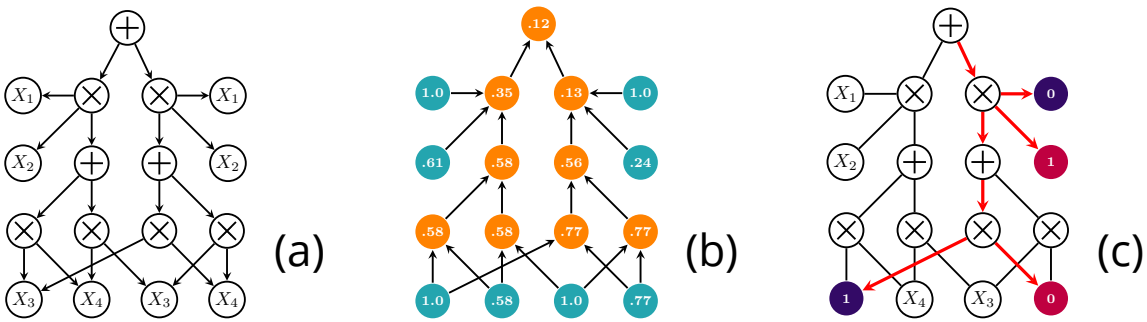
- A leaf n defines a **tractable distribution**
$$\phi_n(\mathbf{x}) = p(\mathbf{x}_{|\mathsf{sc}(n)})$$
- a product node n represents a **factorization over independent components**
$$S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$$
- a sum node n denote a **mixture** over its children distributions
$$S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$$



Inference with SPNs

An SPN S over $\mathbf{X}$ allows the ***exact*** computation of *complete evidence*, *marginal* and *conditional* queries for $p(\mathbf{X})$ in ***time linear*** in the network size (# edges, e.g. $|S|$).

Exact ***MPE inference***, however, is NP-hard in general SPNs [1] but can be answered in exactly in linear time in ***selective SPNs*** by the MaxProdMPE algorithm.



MaxProdMPE turns S into a ***Max-Product Network*** (**MPN**) M *(a)*, then evaluates M **bottom-up** propagating evidence and marginalizing over query RVs *(b)*. A Viterbi-style step retrieves the query RV assignments ***growing a tree path* top-down** —following max sum node child branches and all product node children *(c)*.

---

[1] **Peharz, Gens, et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016**

Sum-Product Autoencoding (SPAE)

Given an SPN S—***unsupervisedly learned*** to estimate $p(\mathbf{X})$ we want to ***encode*** a sample $\mathbf{x}^i \sim \mathbf{X}$ as an *embedding* $\mathbf{e}^i$ in a new d-dimensional space $\mathbf{E}_\mathbf{X} \subseteq \mathbb{R}^d$

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

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

$$g_S(\mathbf{e}^i) = \tilde{\mathbf{x}}^i \approx \mathbf{x}^i.$$

Embeddings over $\mathbf{X}$ can be later used in ***predictive tasks*** as features:

$\Rightarrow$ *e.g.* **to predict** *a RV* Y

or as the output of a predictive model p whose target space is $\mathbf{E}_\mathbf{X}$

$\Rightarrow$ *e.g.* **to disentangle** *label dependencies* $\mathbf{Y}$ *in MLC*

We equip S with f_S and g_S by exploiting MPE inference routines

$\Rightarrow$ *dealing with categorical and continuous representations*

$\Rightarrow$ *dealing with* **partial embeddings**

CAT **embeddings (I)**

Given an SPN S over $\mathbf{X}$, to each sum node $n \in \mathbf{S}^{\oplus}$ is associated a ***categorical latent variable*** (**LV**) Z_n having values $z_n \in \{0, \ldots, |\mathsf{ch}(n)| - 1\}$.

It would be natural to encode $\mathbf{x}^i$ through the LVs in S, i.e. $\mathbf{E}_{\mathbf{X}} = \mathbf{Z}_S$ ($d = |\mathbf{S}^{\oplus}|$):

$$f_S(\mathbf{x}^i) = f_{\mathsf{CAT}}(\mathbf{x}^i) \triangleq \tilde{\mathbf{z}}^i = \mathrm{argmax}_{\mathbf{z}^i}\, p(\mathbf{z}^i \mid \mathbf{x}^i), \tag{1}$$

i.e. $\mathbf{x}^i$ is encoded as the *categorical* vector $\tilde{\mathbf{z}}^i$ comprising the ***MPE state*** for $\mathbf{Z}_S$.

Analogously, the decoding of $\tilde{\mathbf{z}}^i$ through g_S can be defined as:

$$g_S(\tilde{\mathbf{z}}^i) = g_{\mathsf{CAT}}(\tilde{\mathbf{z}}^i) \triangleq \tilde{\mathbf{x}}^i = \mathrm{argmax}_{\mathbf{x}^i}\, p(\mathbf{x}^i \mid \tilde{\mathbf{z}}^i). \tag{2}$$

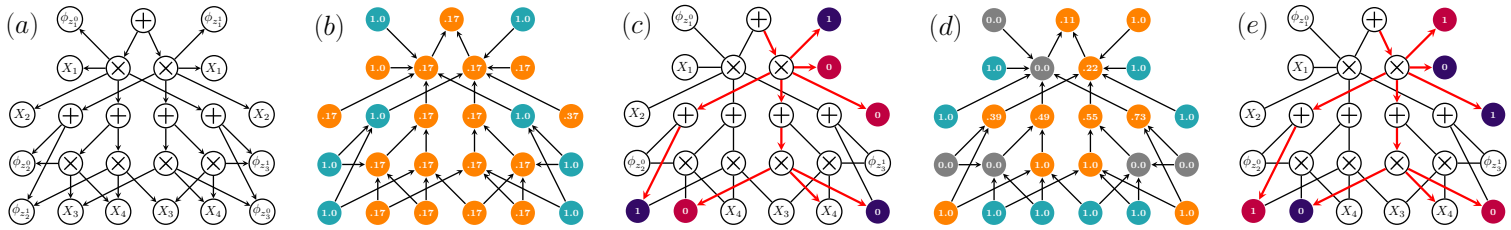
However, this requires performing MPE inference over the ***joint probability distribution*** over $\mathbf{V} = (\mathbf{X}, \mathbf{Z}_S)$

$\Rightarrow$ *we need to deal with an* **augmented SPN** [a] $\overline{S}$ *over* $\mathbf{V}$

---

[a] Peharz, Gens, et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

CAT **embeddings (II)**



To solve both Eq. (1) and Eq. (2), has to be run MaxProdMPE twice on the augmented MPN $\overline{M}$. Since each application of MaxProdMPE involves a bottom-up and a backtracking pass, we need in total 4 passes over $\overline{M}$.

$\Rightarrow$ $\overline{M}$ *is selective, hence MPE inference is exact!*

Materializing $\overline{M}$ scales quadratically, thus we directly use M, evaluating $M(\mathbf{x}^i)$ in a bottom-up pass once and then ***growing a tree path*** θ while collecting the states:

$$z_j^i = \text{argmax}_{k \in \{0,\ldots,|\text{ch}(n_j)|\}} \, w_{n_j c_k} M_{c_k}(\mathbf{x}^i), \tag{3}$$

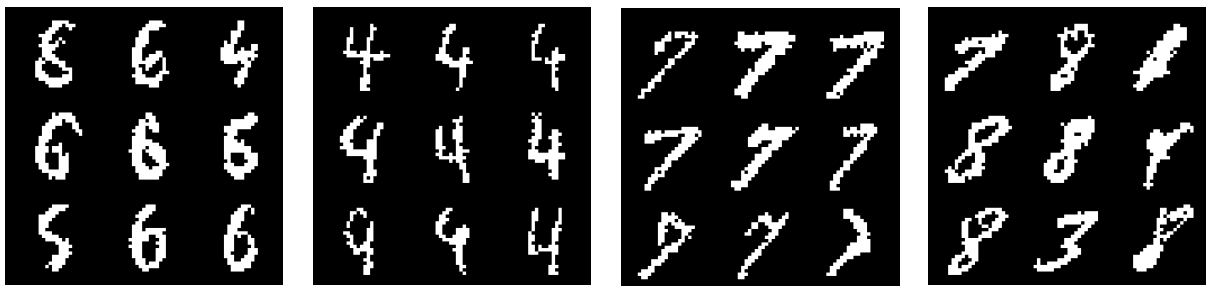
for each $Z_j \in \mathbf{Z}_S^\theta$, where $\mathbf{Z}_S^\theta$ are the LVs associated only to the max nodes in θ.

$\Rightarrow$ CAT *embeddings are very* **sparse** *!*

CAT **embeddings (III)**

CAT embeddings are ***compact* and *linear* representations of trees**, the **induced trees** in S [2].

We can interpret the semantics of CAT embeddings by visualizing ***the latent factors of variations*** encoded in $\mathbf{Z}_S$ through the *clusters* of samples sharing the same representations [3].



For an SPN learned on MNIST, samples sharing the same CAT encoding—even if belonging to different classes, clearly share ***stylistic aspects*** like *orientation* and *stroke*.

---

[2] Zhao, Melibari, et al., "On the Relationship between Sum-Product Networks and Bayesian Networks", 2015

[3] Vergari, Di Mauro, et al., "Visualizing and Understanding Sum-Product Networks", 2016

ACT **embeddings (I)**

SPNs be interpreted as deep neural networks with sparse ***constrained topology*** in which neurons ***labeled*** by the scope function sc—enabling a *direct encoding* of the input—retaining a ***fully probabilistic semantics*** [4].

$\Rightarrow$ *each neuron activation, i.e.* $S_n(\mathbf{x})$*, is a valid probability*

Therefore, neuron ACT***ivations*** can be used as features to build embeddings, as it is common practice for neural networks and autoencoders **[Marlin, Swersky, et al. 2010; Rifai, Vincent, et al. 2011]**

$\Rightarrow$ *however* **representations are not arranged layer-wise** *!*

Let $\mathbf{N} = \{n_j\}_{j=1}^{d} \subseteq \mathbf{M}$ be a set of nodes in an MPN M, by a *certain criterion*. A sample $\mathbf{x}^i$ is encoded into a d-dimensional ***continuous*** embedding $f_S(\mathbf{x}^i) = \mathbf{e}^i \in \mathbf{E}_\mathbf{X} \subseteq \mathbb{R}^d$ by collecting the activations of nodes in $\mathbf{N}$, i.e.

$$e_j^i = M_{n_j}(\mathbf{x}^i)$$

.

[4] Vergari, Di Mauro, et al., "Visualizing and Understanding Sum-Product Networks", 2016

ACT **embeddings (II)**

We can note how ACT ***embeddings implicitly encode an induced tree*** : node activations $\mathbf{e}_M^i$ are sufficient to determine which max node child branch to follow, according to Eq. 3—recompute each hard decision again.

Therefore, we can build a decoder g_{ACT} that ***mimicks only the top-down pass*** of MaxProdMPE: growing the induced tree from the root by following the max sum node child branches—all product child nodes are followed as usual.

Given an SPN S over $\mathbf{X}$—equipped with $(f_{\text{CAT}}, g_{\text{CAT}})$ and $(f_{\text{ACT}}, g_{\text{ACT}})$—and a sample $\mathbf{x}^i \sim \mathbf{X}$, it holds that:

$$g_{\text{ACT}}(f_{\text{ACT}}(\mathbf{x}^i)) = g_{\text{CAT}}(f_{\text{CAT}}(\mathbf{x}^i)). \tag{4}$$

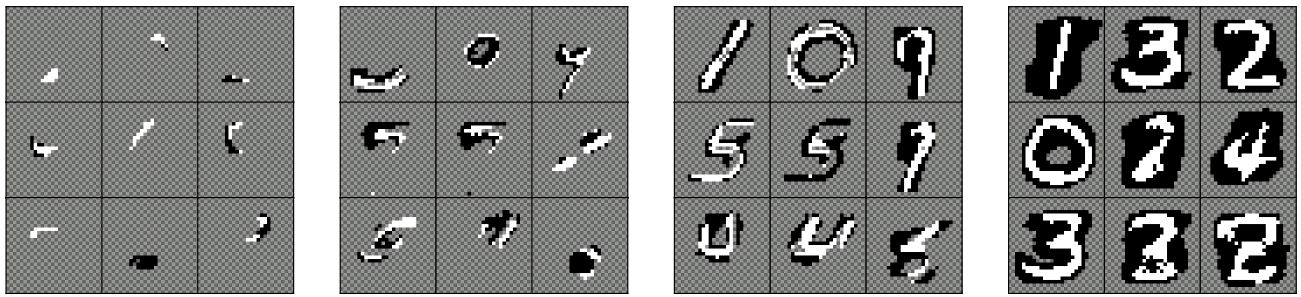
$\Rightarrow$ *different embeddings, but* **equivalent reconstructions** *!*

ACT **embeddings (III)**

Since ACT embeddings are points in the space induced by a collection of *distributions*, SPN nodes are ***part-based filters*** operating over different sub-spaces of RVs.

For an SPN S we can visualize the filter encoded by sub-network S_n rooted at node n by ***computing the mode*** of the distribution p_{S_n}:

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



E.g., on MNIST, differently complex local patterns emerge e.g. from small blobs to shape contours and finally full digits

$\Rightarrow$ a ***hierarchy of representations*** structured at levels of abstraction!

CAT **vs** ACT **embeddings**

Even if one can demonstrate that CAT and ACT embeddings can lead to the same reconstructions (see Eq. 4), however, ***they act differently when plugged in predictive tasks*** (both as feature and target representation spaces).

$\Rightarrow$ *exhaustive empirical evaluation for MLC*

When employed as features for a predictor (its input) ACT ***embeddings*** perform better than CAT ones due to their ***greater information content***.

$\Rightarrow$ CAT *embeddings are shared more frequently among samples*

Conversely, when employed to encode target RVs (a predictor's output) ***classification* for the CAT *case is easier*** than *regression* with ACT embeddings.

$\Rightarrow$ *simpler prediction task due to the sparsity*

Partial embedding decoding

Up to now we have considered only ***fully decodable embeddings***, i.e. embeddings comprising all the information required to materialize a ***complete and well-formed tree*** necessary to decode $\mathbf{e}$ into $\tilde{\mathbf{x}}$.
In some real cases, however, only incomplete or ***partial embeddings*** are available: some values e_j are corrupted, invalid or just missing.

$\Rightarrow$ *e.g., data compression*

SPAE routines offer a natural and efficient way to deal with such cases: MPE inference.

$\Rightarrow$ *treat* **missing embedding components as missing values**

In practice, if for an ACT (resp. CAT) embedding the component $e_j^i \notin \mathbf{e}^i$ (resp. $z_j^i \notin \mathbf{z}^i$) corresponds to a node n_j activation (resp. LV Z_j state), then it can be imputed by employing MaxProdMPE on the sub-network M_{n_j}.

$\Rightarrow$ *imputation for all missing components in one single pass*

MLC prediction tasks (I)

Evaluating SPAE on ***Multi-Label Classification*** (**MLC**): predicting the target labels—binary arrays—$\mathbf{y}^i \sim \mathbf{Y}$ associated to sample $\mathbf{x}^i \sim \mathbf{X}$.

Evaluating four ***different learning scenarios***:

- ⊕ no embedding at all (baseline)

$$\mathbf{X} \overset{p}{\Rightarrow} \mathbf{Y}$$

- ⊕ when embedding only input RVs $\mathbf{X}$

$$(\mathbf{X} \xrightarrow{f_r} \mathbf{E_X}) \overset{\text{LR}}{\Rightarrow} \mathbf{Y}$$

- ⊕ when embedding only target RVs $\mathbf{Y}$ (*requires decoding!*)

$$(\mathbf{X} \overset{p}{\Rightarrow} (\mathbf{Y} \xrightarrow{f_t} \mathbf{E_Y})) \xrightarrow{g_t} \mathbf{Y}$$

- ⊕ when embedding both RV sets $\mathbf{X}$, $\mathbf{Y}$

$$((\mathbf{X} \xrightarrow{f_r} \mathbf{E_X}) \overset{p}{\Rightarrow} (\mathbf{Y} \xrightarrow{f_t} \mathbf{E_Y})) \xrightarrow{g_t} \mathbf{Y}$$

MLC prediction tasks (II)

	$\mathbf{X} \overset{p}{\Rightarrow} \mathbf{Y}$	JAC	EXA
baseline	p: LR	0.00	0.00
	p: CRF_{SSVM}	+15.83	+103.90
scenario I	r: $\text{RBM}_{h \in \{500,1000,5000\}}$	+1.46	-1.62
	r: $\text{MADE}_{h \in \{500,1000\}}$	+2.57	+2.99
	r: $\text{CAE}_{\gamma \in \{0.7,0.8,0.9\}}$	-0.15	+4.13
	r: $\text{DAE}_{\gamma \in \{0.7,0.8,0.9\}}$	+0.70	+4.17
	r: SPAE_{ACT}	**+3.54**	**+17.18**
	r: SPAE_{CAT}	-11.90	-11.53
scenario II	t: $\text{MADE}_{h \in \{200,500\}}$, p: RR	-30.42	-28.02
	t: $\text{SAE}_{\gamma \in \{0.7,0.8,0.9\}}$, p: RR	+5.96	+95.78
	t: $\text{CAE}_{\gamma \in \{0.7,0.8,0.9\}}$, p: RR	+7.60	+78.81
	t: $\text{DAE}_{\gamma \in \{0.7,0.8,0.9\}}$, p: RR	+13.39	+102.22
	t: SPAE_{ACT}, p: RR	+15.19	+98.58
	t: SPAE_{CAT}, p: LR	**+24.07**	**+141.81**
scenario III	r, t: MADE, p: RR	-27.15	-25.14
	r, t: $\text{CAE}_{\gamma \in \{0.7,0.8,0.9\}}$, p: RR	+5.21	+79.20
	r, t: $\text{DAE}_{\gamma \in \{0.7,0.8,0.9\}}$, p: RR	+13.97	+98.25
	r: SPAE_{ACT}, t: SPAE_{ACT}, p: RR	+15.98	+106.65
	r: SPAE_{CAT}, t: SPAE_{CAT}, p: LR	+13.73	+107.05
	r: SPAE_{ACT}, t: SPAE_{CAT}, p: LR	**+25.47**	**+144.78**

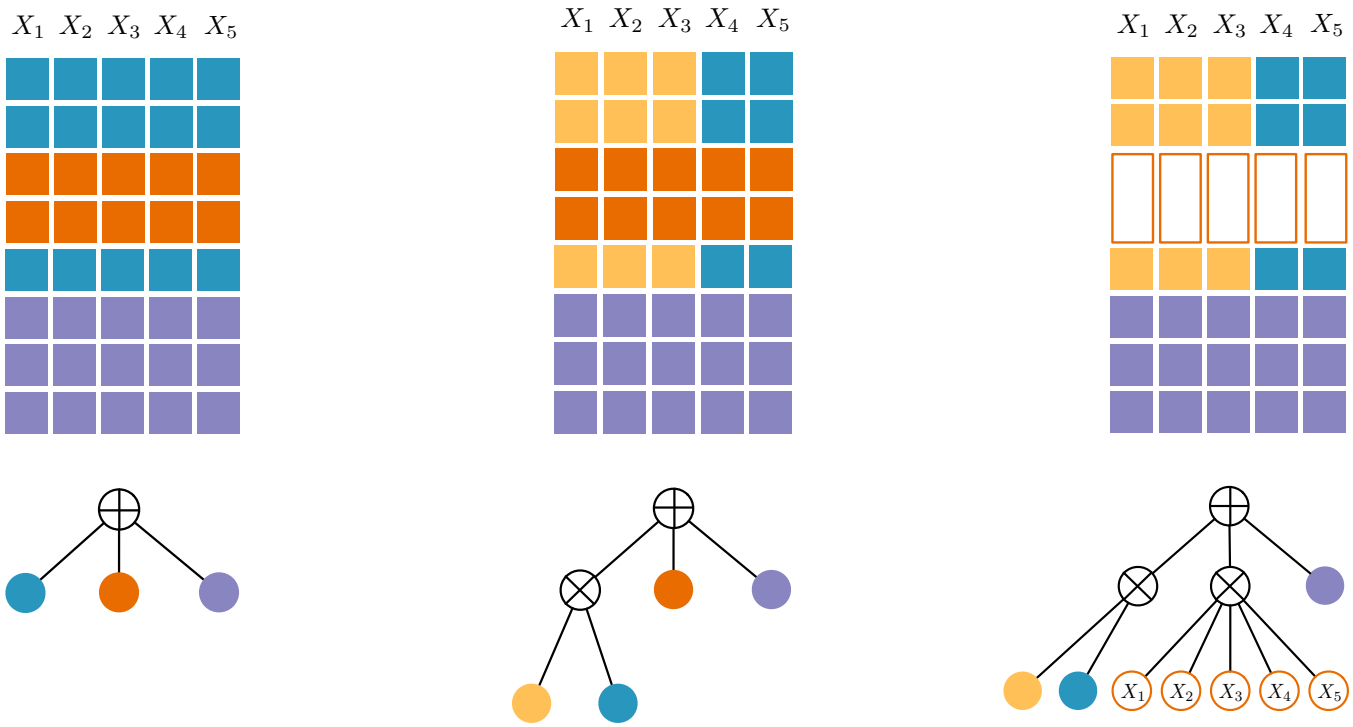
Measuring the average relative improvement for for the JACcard, HAMming and EXAct match scores over ***10 standard MLC benchmark datasets***.

In all scenarios we employ a **linear predictor**: a logistic (LR) or ridge regressor (RR) for classification or regression, respectively.

Both ACT and CAT are competitive, in all scenarios—for all scores—against:

- ⊕ RBMs
- ⊕ probabilistic autoencoders (MADEs)
- ⊕ deep stacked autoencoders (SAEs)
- ⊕ contractive autoencoders (CAEs)
- ⊕ denoising autoencoders (DAEs)

Why SPAE works for RL...



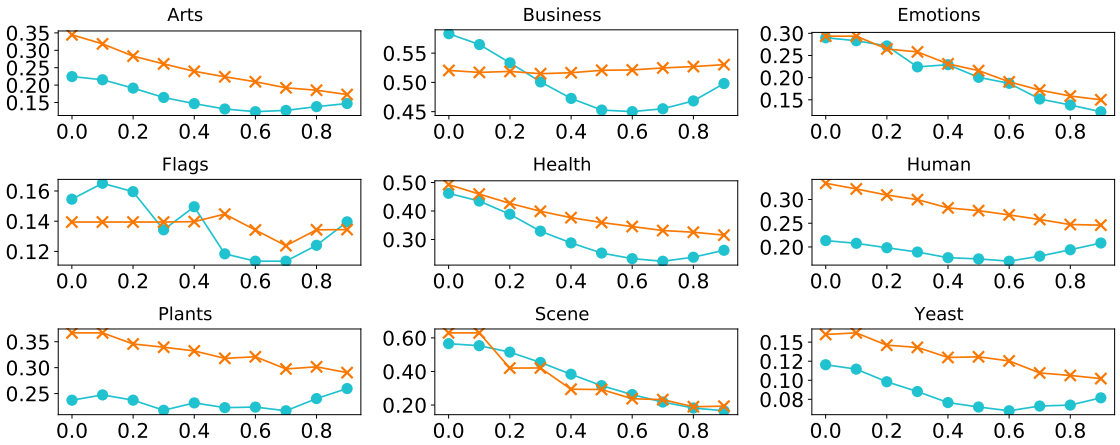
SPNs are built via *hierarchical co-clustering*, learning features as ***recursive data crawlers***!

Vergari, Di Mauro, et al., "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning", 2015

Gens and Domingos, "Learning the Structure of Sum-Product Networks", 2013

Partial embedding decoding

Evaluating the ***resilience to missing-at-random*** embedding components when decoding CAT and ACT label embeddings in scenario II—varying from 0 (full embedding) to 90% missing, by increments of 10



Both CAT and ACT decoding routines are quite robust

$\Rightarrow$ *degrading performances by less than* 50% *with* 50% *missing components*

CAT scores decay slower than ACT and are generally better

Conclusions

We extend the scope of SPNs towards Representation Learning (RL).
Compared to other classical probabilstic models (RBMs, MADEs) and autoencoders (CAEs, DAEs, SAE, etc.) for RL, SPNs in our SPAE routines can:

- ⊕ still be used as tractable models for exact inference on a ***wider range of queries***
- ⊕ provide ***rich***, ***hierarchical*** **and** ***part-based*** **features**
- ⊕ save *time and effort* in hyperparameter tuning since both their structure and weights can be learned in a "cheap" way—e.g., adaptive size

SPAE suggests several interesting avenues for future work:

- ⊕ explore embeddings based on other instances of Arithmetic Circuits **[Darwiche 2009]**
- ⊕ extracting ***structured representations***
- ⊕ perform ***differentiable MPE inference*** allowing SPNs—bridging the gap even more between SPNs, autoencoders and other neural networks.

Shameless plug

I Star or fork the ***awesome-spn*** repo for more references to the SPN literature:

`https://github.com/arranger1044/awesome-spn`

II Visit our poster for more on SPN structure learning tomorrow, *Main Track Session 1*

"Alternative Variable Splitting Methods to Learn Sum-Product Networks"

joint work with Nicola Di Mauro, Floriana Esposito and Fabrizio Ventola

III ***Are you looking for a post-doc*** to work on *probabilistic models*, *deep learning* and *machine learning*? Here is my cv:

`tinyurl.com/ycnhn7nu`

References I

- Darwiche, Adnan (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge.
- Gens, Robert and Pedro Domingos (2013). "Learning the Structure of Sum-Product Networks". In: *Proceedings of the ICML 2013*, pp. 873–880.
- Marlin, Benjamin M et al. (2010). "Inductive Principles for Restricted Boltzmann Machine Learning". In: *AISTATS 2010*, pp. 509–516.
- Peharz, Robert et al. (2016). "On the Latent Variable Interpretation in Sum-Product Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP, Issue 99. URL: `http://arxiv.org/abs/1601.06180`.
- Poon, Hoifung and Pedro Domingos (2011). "Sum-Product Networks: a New Deep Architecture". In: *UAI 2011*.
- Rifai, Salah et al. (2011). "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the Twenty-eight International Conference on Machine Learning, ICML*.
- Vergari, Antonio, Nicola Di Mauro, and Floriana Esposito (2015). "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning". In: *ECML-PKDD 2015*.
- — (2016). "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*. URL: `https://arxiv.org/abs/1608.08266`.
- Zhao, Han, Mazen Melibari, and Pascal Poupart (2015). "On the Relationship between Sum-Product Networks and Bayesian Networks". In: *ICML*.

Discuss