University of Bari
Department of Computer Science

LACAM Laboratory
Machine Learning

Learning and Exploiting Deep Tractable Probabilistic Models

Antonio Vergari

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

19th December 2017 - Université Paris Sud

Outline

A couple recent topics from my research as a PhD.
A little excursus on ***bridging deep and probabilistic models*** to leverage both exact and efficient probabilistic inference and rich and compositional representations ***towards automating density estimation*** over ***hybrid domains***.

Focusing on ***Sum-Product Networks*** (**SPNs**) **[Poon and Domingos 2011]** as they can be pivotal for both. Talking about **what** SPNs can offer, **how** they can be exploited and **why** you may want to use them.

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

$\Longrightarrow$ e.g., *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:

$\Longrightarrow$ e.g., *complete evidence* (EVA), *marginals* (MAR), *conditionals* (CON), *Most Probable Explanaition* (MPE) and MAP *assignments*,...

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

The density estimation pipeline

1. decide a **parametric form** for the estimator
   - ⊗ a parametric form for individual RVs *(e.g., are counts of vehicles poisson distributed?)*
   - ⊗ the dependency structure parametric form *(e.g., are jams influenced by salary growth?)*
2. **fit the estimator** to the data *(e.g., optimize data likelihood)*
   - ⊗ fit model structure
   - ⊗ fit model parameters
3. perform **inference *ad libitum***
   - ⊗ several kinds of probabilistic queries *(e.g., how likely is to see ?)*
   - ⊗ compute statistics, metrics, descriptors *(e.g., mutual information)*
   - ⊗ make sense of the data and the model (**interpretability**) *(e.g., what has been learned?)*
   - ⊗ ...
4. **(*re*-)use knowledge** in other tasks *(e.g., can representations learned for traffic counts be used to predict where to build a city mall?)*

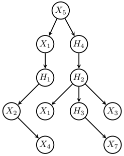
Tractable Probabilistic Models (TPMs)

Classical Probabilistic Graphical Models (PGMs) like ***Bayesian Networks*** (**BNs**) and ***Markov Networks*** (**MNs**) are highly expressive but exact inference is in general *NP-hard*.

Tractable Probabilistic Models (**TPMs**) are density estimators for which some kind of **inference is *exact* and *tractable*** , i.e. *polynomial* in the number of RVs:

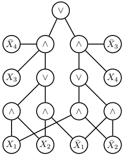
$\rightarrow$ e.g., ***bounded tree-width PGMs***, ***computational graphs*** and ***neural autoregressive models***

BN trees



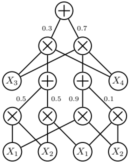
✓EVI, ✓SAM, ✓MAR, ✓CON, ✓MPE, ✓Z

d-DNNFs



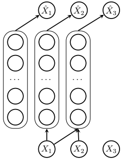
✓EVI, ✗SAM, ✓MAR, ✓CON, ✓MPE, ✓Z

SPNs



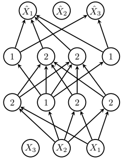
✓EVI, ✓SAM, ✓MAR, ✓CON, ✗MPE, ✓Z

NADEs



✓EVI, ✓SAM, ✗MAR, ✗CON, ✗MPE, ✗Z

MADEs



✓EVI, ✓SAM, ✗MAR, ✗CON, ✗MPE, ✗Z

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**

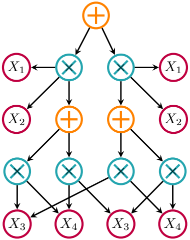
$$\phi_n(\mathbf{x}) = p(\mathbf{x}_{|\mathsf{sc}(n)})$$

- ⊕ 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 denotes a **mixture** over its children distributions

$$S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$$



SPNs: exact and tractable inferences

Let $\mathbf{S}^{\oplus}$ (resp. $\mathbf{S}^{\otimes}$) be the set of all sum (resp. product) nodes in an SPN S, then

- S is ***complete*** iff $\forall n \in \mathbf{S}^{\oplus}, \forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$
- S is ***decomposable*** iff $\forall n \in \mathbf{S}^{\otimes}, \forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$

If S is complete and decomposable, then it is ***valid*** and allows for the efficient computation of a network polynomial:

$\Rightarrow$ ***evidence***, ***marginals***, ***Z*** in time linear to $|S|$[1,2]

An SPN S is ***selective***[3], iff

$\forall \mathbf{x}^i \sim \mathbf{X}, \forall n \in \mathbf{S}^{\oplus} : |\{c \,|\, c \in \mathsf{ch}(n) : S_c(\mathbf{x}^i) > 0\}| \leq 1$

$\Rightarrow$ ***MPE inference***, ***assignments*** in time linear to $|S|$[4]

---

[1] Darwiche, *Modeling and Reasoning with Bayesian Networks*, 2009

[2] Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

[3] Peharz, Gens, et al., "Learning Selective Sum-Product Networks", 2014

[4] Choi and Darwiche, "On Relaxing Determinism in Arithmetic Circuits", 2017

Building SPNs...

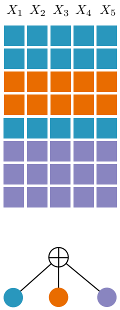
	X_1	X_2	X_3	X_4	X_5
1					
2					
3					
4					
5					
6					
7					
8					

Learning both structure and parameters of SPNs with algorithmic variants of **LearnSPN**

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

Building SPNs...

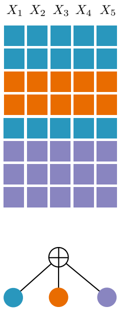


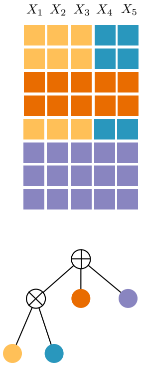
Looking for sub-population in the data—***clustering***—to introduce sum nodes...

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

Building SPNs...





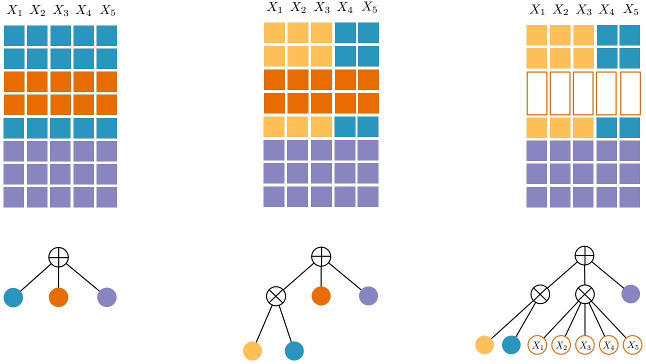
...seeking ***statistical independence*** among RVs to factorize into product nodes

---

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

Building SPNs...



...learning smaller estimators as a ***a recursive data crawler***

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

...and building upon SPNs

SPNs as ***recursive hierarchical decomposition*** of larger models into smaller ones.
Tackling inference and learning complexity by pushing it down towards the leaves

$\Longrightarrow$ computing ***mode***, mean, variance,...efficiently [5]

$\Longrightarrow$ ***delegating encoding and decoding*** to leaf distributions [6]

The Sum-Product Theorem [7] hints at generalizations over other semi-rings

$\Longrightarrow$ e.g., composing kernel machines [8]

SPNs as **divide-et-impera** machines ***gluing and orchestrating inference*** among different (possibly heterogeneous) models.

$\Longrightarrow$ performing ***MPE inference over autoencoders*** from different domains [9]

---

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

[6] Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

[7] Friesen and Domingos, "The Sum-Product Theorem: A Foundation for Learning Tractable Models", 2016

[8] Gens and Domingos, "Compositional Kernel Machines", 2017

[9] Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

Exploiting SPNs *more than once*

Learn one SPN S generatively from data $\{\mathbf{x}^i \sim \mathbf{X}\}_{i=1}^{m}$ to estimate $p(\mathbf{X})$ and then exploit it—*without retraining* it—by ***interpreting it as a neural network***:

- as a **feature extractor** for Representation Learning (RL)

  $\Longrightarrow$ *sum, product nodes or scope aggregations as filters* [10]

- as an ***autoencoder*** mapping back and forth embeddings

  $\Longrightarrow$ *Sum-Product Autoencoding* [11]

- understanding learned representations

  $\Longrightarrow$ *visualizing filters in the input space*

Moreover the interpretation of SPNs as NNs enables

- efficient implementations running on GPUs
- structure learning as a constrained optimization problem

---

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

[11] Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

MPE inference with SPNs

Exact ***MPE inference***, e.g. computing for RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$, $\mathbf{Q} \cap \mathbf{O} = \emptyset$

$$\operatorname{argmax}_{\mathbf{q} \sim \mathbf{Q}} p(\mathbf{q}|\mathbf{O})$$

is NP-hard for a general SPN S over $\mathbf{X}$ but can be approximated in linear time in $|S|$ by the $\mathsf{MaxProdMPE}$ algorithm (but exact for selective SPNs)

Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

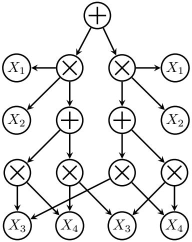
Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

MPE inference with SPNs

Exact ***MPE inference***, e.g. computing for RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$, $\mathbf{Q} \cap \mathbf{O} = \emptyset$

$$\text{argmax}_{\mathbf{q} \sim \mathbf{Q}}\, p(\mathbf{q}|\mathbf{O})$$

is NP-hard for a general SPN S over $\mathbf{X}$ but can be approximated in linear time in $|S|$ by the MaxProdMPE algorithm (but exact for selective SPNs)



E.g. to compute the MPE state of RVs $\mathbf{Q} = \{X_1, X_3\}$ given $\mathbf{O} = \{X_2, X_4\}$

$$\text{argmax}_{\mathbf{q} \sim \mathbf{Q}}\, p_S(\mathbf{q}, X_2 = 1, X_4 = 0),$$

MaxProdMPE first turns S into a ***Max-Product Network* (MPN)** M, by replacing sum nodes with *max nodes* and leaf distributions with *maximizing distributions*

Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

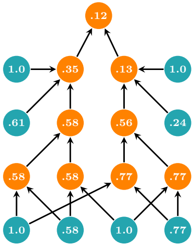
Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

MPE inference with SPNs

Exact ***MPE inference***, e.g. computing for RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$, $\mathbf{Q} \cap \mathbf{O} = \emptyset$

$$\text{argmax}_{\mathbf{q} \sim \mathbf{Q}}\, p(\mathbf{q}|\mathbf{O})$$

is NP-hard for a general SPN S over $\mathbf{X}$ but can be approximated in linear time in $|S|$ by the MaxProdMPE algorithm (but exact for selective SPNs)



...then M is evaluated ***bottom-up*** to compute $M(\mathbf{O})$ by propagating evidence from children to parents and marginalizing over query RVs $\mathbf{Q}$

Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

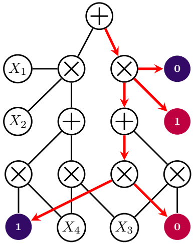
Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

MPE inference with SPNs

Exact ***MPE inference***, e.g. computing for RVs $\mathbf{Q}, \mathbf{O} \subset \mathbf{X}$, $\mathbf{Q} \cup \mathbf{O} = \mathbf{X}$, $\mathbf{Q} \cap \mathbf{O} = \emptyset$

$$\operatorname{argmax}_{\mathbf{q} \sim \mathbf{Q}} p(\mathbf{q}|\mathbf{O})$$

is NP-hard for a general SPN S over $\mathbf{X}$ but can be approximated in linear time in $|S|$ by the MaxProdMPE algorithm (but exact for selective SPNs)



A Viterbi-style step retrieves the query assignments to $\mathbf{Q}$ ***growing a tree path top-down***, starting from the root:

- following only the maximum activation children for a max node
- following all child branches for product nodes
- maximizing on leaf distributions over $\mathbf{Q}$

Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

Vergari, Peharz, et al., "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks", 2017

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 $g \colon \mathbf{E}_{\mathbf{X}} \to \mathbf{X}$ 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 $z_n \in \{0, \ldots, |\text{ch}(n)| - 1\}$.

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

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

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:

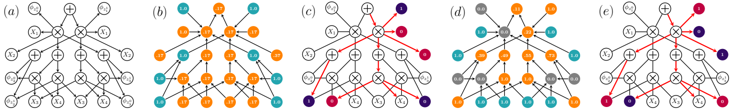
$$g_S(\tilde{\mathbf{z}}^i) = g_{\text{CAT}}(\tilde{\mathbf{z}}^i) \triangleq \tilde{\mathbf{x}}^i = \text{argmax}_{\mathbf{x}^i}\, p(\mathbf{x}^i \,|\, \tilde{\mathbf{z}}^i). \tag{2}$$

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

$\Longrightarrow$ *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:

$$z_j^i = \mathrm{argmax}_{k\in\{0,\dots,|\mathsf{ch}(n_j)|\}} w_{n_j c_k} M_{c_k}(\mathbf{x}^i), \qquad (3)$$

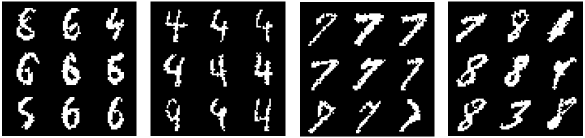
for each $Z_j \in \mathbf{Z}_S^\theta$, where $\mathbf{Z}_S^\theta$ are the LVs associated only to the max nodes in θ.

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

CAT **embeddings (III)**

CAT embeddings are ***compact* and *linear* representations of trees**, the **induced trees** in S [12].

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 [13].



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*.

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

[13] 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*** [14].

$\Rightarrow$ *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 $f_S(\mathbf{x}^i) = \mathbf{e}^i \in \mathbf{E_X} \subseteq \mathbb{R}^d$ by collecting the activations of nodes in $\mathbf{N}$, i.e.

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

.

[14] **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 $\mathbf{e}^i_M$ 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 $(f_{\mathsf{CAT}}, g_{\mathsf{CAT}})$ and $(f_{\mathsf{ACT}}, g_{\mathsf{ACT}})$—and a sample $\mathbf{x}^i \sim \mathbf{X}$, it holds that:

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

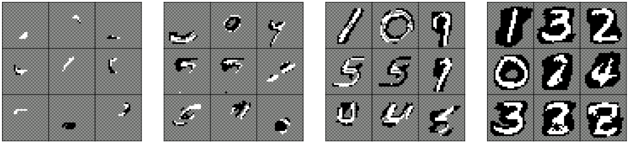
$\Rightarrow$ *different embeddings, but* **equivalent reconstructions***!*

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).

$\Longrightarrow$ *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***.

$\Longrightarrow$ 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.

$\Longrightarrow$ *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 $e^i_j \notin \mathbf{e}^i$ (resp. $z^i_j \notin \mathbf{z}^i$) corresponds to a node n_j activation (resp. LV Z_j state), then it can be imputed by employing MaxProdMPE on the sub-network M_{n_j}.

$\Rightarrow$ *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**

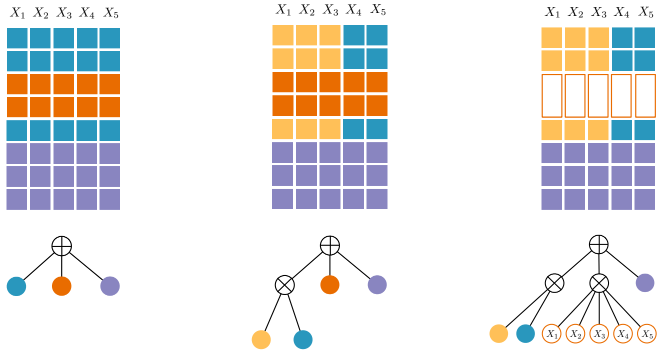
Measuring the average relative improvement for for the JACcard, HAMming and EXAct match scores over ***10 standard MLC benchmark datasets***.

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...



remember 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

Automating density estimation

1. decide a **parametric form** for the estimator
   - ⊗ a parametric form for individual RVs
   - ⊗ the dependency structure parametric form $\Rightarrow$ ***SPN structure***
2. **fit the estimator** to the data $\Rightarrow$ ***LearnSPN***
   - ⊗ fit model structure
   - ⊗ fit model parameters
3. perform **inference *ad libitum***
   - ⊗ several kinds of probabilistic queries $\Rightarrow$ ***EVI, MAR, CON,...***
   - ⊗ compute statistics, metrics, descriptors
   - ⊗ make sense of the data and the model (**interpretability**) $\Rightarrow$ ***visualizing filters***
   - ⊗ ...
4. **(*re-*)use knowledge** in other tasks $\Rightarrow$ ***SPAE embeddings and routines***

Mixed Sum-Product Networks (MSPNs)

Relieving practitioners from imposing parametric forms for RVs and their interactions

$\Longrightarrow$ *general strong assumptions—e.g., gaussianity—may not hold in practice*

$\Longrightarrow$ *specific knowledge over hybrid domains is often beyond users' possibilities*

LearnSPN and its variants are tailored towards specific parametric assumptions—gaussian [15], multinomial [16] or poisson data [17]

Mixed Sum-Product Networks (**MSPNs**) combine SPNs with **piecewise polynomial approximations** to provide a density estimator **without making parametric form assumptions** when

- $\oplus$ seeking ***RV dependencies*** (column splits)
- $\oplus$ determining ***instance clustering*** (row splits)
- $\oplus$ modeling ***univariate distributions*** (leaf growing)

---

[15] Jaini, Rashwan, et al., "Online Algorithms for Sum-Product Networks with Continuous Variables", 2016

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

[17] Molina, Natarajan, et al., "Poisson Sum-Product Networks: A Deep Architecture for Tractable Multivariate Poisson Distributions", 2017

LearnMSPN: decomposing RVs I

Looking for RV dependency through an empirical estimator for Rényi's Maximum Correlation Coefficient [Rényi 1959], the ***Randomized Dependency Coefficient*** (**RDC**) [Lopez-Paz, Hennig, et al. 2013].

RVs X_i and X_j are independent iff for two samples $\mathcal{D}_{X_i} = \{x_i^m | x_i^m \sim X_i\}_{m=1}^{M}$ and $\mathcal{D}_{X_j} = \{x_j^m | x_j^m \sim X_j\}_{m=1}^{M}$ RDC$(\mathcal{D}_{X_i}, \mathcal{D}_{X_j}) \approx 0$.

I. Preserve marginal structure by going through the ***empirical cdf***:

$$\mathcal{C}_{X_i} = \left\{ \frac{1}{M} \sum_{r=1}^{M} \mathbb{1}\{v_i^r \leq v_i^m\} \middle| v_i^m \in \mathcal{D}_{X_i} \right\}_{m=1}^{M}$$

II. Randomly project to a ***k-dimensional gaussian space***, and then apply a ***non-linearity*** σ.

$$\phi(\mathcal{C}_{X_i}) = \sigma(\mathbf{w} \cdot \mathcal{C}_{X_i}^T + b), (\mathbf{w}, b) \sim \mathcal{N}(\mathbf{0}_k, s\mathbf{I}_{k \times k})$$

Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

LearnMSPN: decomposing RVs II

III. The RDC is the largest canonical correlation analysis (CCA) coefficient

$$\mathrm{RDC}(\mathcal{D}_{X_i}, \mathcal{D}_{X_j}) = \sup_{\boldsymbol{\beta},\boldsymbol{\gamma}} \rho(\boldsymbol{\beta}^T \phi(\mathcal{C}_{X_i}), \boldsymbol{\gamma}^T \phi(\mathcal{C}_{X_j})).$$

where ρ^2 is the solution of the eigenproblem for the CCA over $\phi(\mathcal{C}_{X_i})$ and $\phi(\mathcal{C}_{X_j})$:

$$\begin{pmatrix} 0 & \Sigma_{ii}^{-1}\Sigma_{ij} \\ \Sigma_{jj}^{-1}\Sigma_{ji} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\gamma} \end{pmatrix} = \rho^2 \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\gamma} \end{pmatrix},$$

where the covariance block matrices involved are:

$$\Sigma_{ij} = \mathrm{cov}(\phi(\mathcal{C}_{X_i}), \phi(\mathcal{C}_{X_j})), \Sigma_{ji} = \mathrm{cov}(\phi(\mathcal{C}_{X_j}), \phi(\mathcal{C}_{X_i})),$$

$$\Sigma_{ii} = \mathrm{cov}(\phi(\mathcal{C}_{X_i}), \phi(\mathcal{C}_{X_i})), \Sigma_{jj} = \mathrm{cov}(\phi(\mathcal{C}_{X_j}), \phi(\mathcal{C}_{X_j})).$$

Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

LearnMSPN: clustering

Clustering hybrid data highly depends on the metric space employed:

$\Longrightarrow$ *e.g. K-Means relies on gaussianity*

We employ the RDC pipeline to project into a homogeneous space in which clusters may be more easily separable.
Given a samples $\mathcal{D}_{\mathbf{X}}$ over RVs $\mathbf{X}$ we:

1. compute $\mathcal{E} = \{\mathcal{C}(\mathcal{D}_{X_i}) | \mathcal{D}_{X_i}\}_{i=1}^{n}$. via the ***empirical copula transform***
2. then project all features into a new ***k-dimensional*** ***non-linear*** space
3. finally, we apply clustering—e.g. safely K-Means—to obtain c clusters

$\Longrightarrow$ $c = 2$ *for deeper SPNs* **[Vergari, Di Mauro, et al. 2015]**

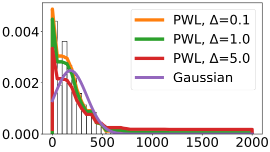
Comparable to employing the ***Gower distance***—if one can make parametric assumptions

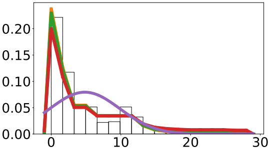
Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

LearnMSPN: leaf distribution modeling

Approximate univariate leaf probability mass or density functions with ***piecewise polynomials***

$\Longrightarrow$ *unwrapping the whole MSPN polynomial for symbolic evaluation*





Degree 0 approximations : piecewise constants, i.e. ***histograms***

$\Longrightarrow$ adaptive bins by fitting an irregular histogram by optimizing a penalized log-likelihood **[Rozenholc, Mildenberger, et al. 2010]**

Degree 1 approximations : piecewise linear

$\Longrightarrow$ reframing it into a supervised task, fitting a piecewise linear model via ***unimodal isotonic regression*** **[Frisen 1986]**

Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

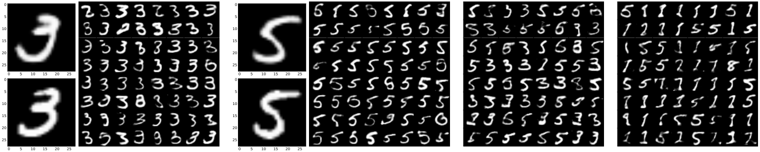
MSPNs: inference over hybrid domains

Toy ***symbol grounding*** with MSPN: embed MNIST digits into a 16-d continuous space $\mathbf{X}$ and augment them with binary codes $\mathbf{Y}$ for semantic features:

(i) a vertical stroke, (ii) a circle, (iii) a left curvy stroke,

(iv) a right curvy stroke, (v) a horizontal stroke, (vi) a double curve stroke

$\Longrightarrow$ *the code for 3 is therefore:* $\mathbf{y}_3 = (0, 0, 1, 0, 1, 1)$



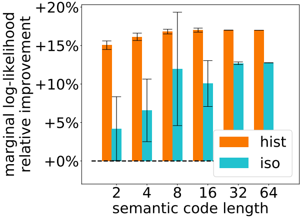
(0,0,1,0,1,1) ***(0,0,1,1,1,0)*** ***(0,0,1,1,1,1)*** ***(1,0,1,1,1,0)***

Model $p(\mathbf{X}, \mathbf{Y}_c, c)$ with an MSPN and perform: Easily perform ***MPE and conditional sampling*** from $p(\mathbf{X}|\mathbf{y}_c)$ over **existing class codes** $\mathbf{y}_c$ and **invented ones**.

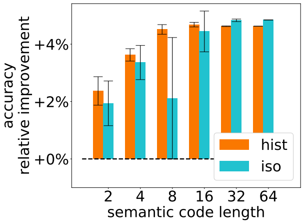
Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

MSPNs: privileged information learning

Efficient marginalization in MSPNs allows to leverage additional RVs at training time as ***privileged information***



a



b

Randomly increasing the semantic codes $\mathbf{Y}_c$ helps modeling both the ***marginal likelihood*** $p(\mathbf{X})$ and the ***predictive accuracy*** on the class c at test time

$\Longrightarrow$ *towards stacking density estimators*

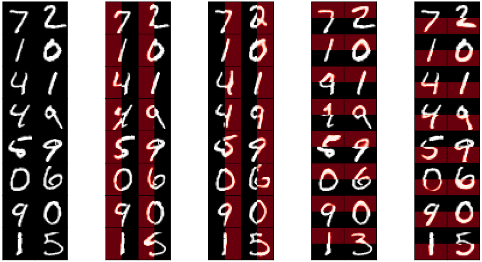
Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

MSPNs: orchestrating inference

Split RVs into two halves—$\mathbf{X}_l$, $\mathbf{X}_r$, $\mathbf{X}_u$, and $\mathbf{X}_d$—and learn one autoencoder f on each RV set *independently*. They act as different domains.

Learn one MSPN M_{ud} to model $P(f_u(\mathbf{X}_u), f_d(\mathbf{X}_d))$ (resp. M_{lr} and $P(f_l(\mathbf{X}_l), f_r(\mathbf{X}_r))$).
Given one half test image, **predict** the other half.

$\Rightarrow$ *M_{ud} fills and glues the embedding spaces of f_u and f_d*



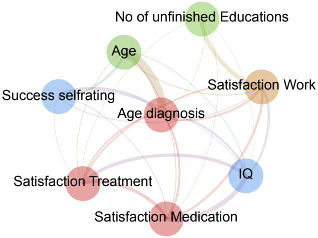
Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

MSPNs: hybrid measures computation

Recall, an MSPN encodes a polynomial over leaf piecewise polynomials.

$\Longrightarrow$ *managing complexity by divide-et-impera representations!*

Employing a symbolic solver to evaluate the overall network polynomial to easily compute information-theoretic measures for hybrid domains,
e.g. ***hybrid mutual information***



Molina, Vergari, et al., "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains", 2017

Automating density estimation

1. decide a **parametric form** for the estimator ⇒ ***MSPN***
   - a parametric form for individual RVs
   - the dependency structure parametric form
2. **fit the estimator** to the data ⇒ ***LearnMSPN***
   - fit model structure
   - fit model parameters
3. perform **inference *ad libitum***
   - several kinds of probabilistic queries ⇒ ***EVI, MAR, CON,*** ***hybrid queries***
   - compute statistics, metrics, descriptors ⇒ ***hybrid MI***
   - make sense of the data and the model (**interpretability**) ⇒ ***visualizing filters***
   - ...
4. **(*re*-)use knowledge** in other tasks ⇒ ***SPAE embeddings*** ***privileged information,***

What is still missing?

Points from 1 to possibly 4 are just ***the inner loop of optimization***!
Still many hyperparameter to tune and value choices to automatize

$\Longrightarrow$ *e.g., dependency threshold, smoothing factor, ...*

Proposal: automating hyperparameter selection a-là gray-box ***AutoML***

$\Longrightarrow$ *CV grid search, bayesian optimization, ...*

With SPNs we can learn the structure, **but also we have to learn the structure**!
Learn(M)SPN is too greedy and requires a top structure to be learned before leaf models

$\Longrightarrow$ *no end-to-end joint learning of structures ...*

Proposal: reframe ***structure learning as constrained optimization*** and use sgd

While piecewise polynomials are flexible enough to approximate several distributions, they may **lack the interpretability of known parametric forms**.
Proposal: also ***infer the parametric form*** of marginal distributions ex-post

$\Longrightarrow$ *is it gaussian, logit, poisson? ...*

In a nutshell

SPNs as deep tractable probabilistic models can be effectively learned as accurate and flexible density estimators—even on mixed domains—and at the same time being exploited to provide new feature representations for predictive tasks.

...additional future works

- Bayesian Sum-Product Networks
- ***SPNify*** other (non-probabilistic) models: autoencoders, Gibbs samplers, GPs,...
- demistify some folklore: *"SPNs are not NNs"*, *"SPNs are not as expressive as NNs"*,...

awesome-spns

Star or fork on github for more references to the SPN literature:

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

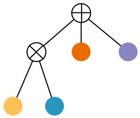
References I

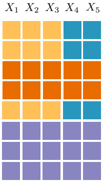
- Choi, Arthur and Adnan Darwiche (2017). "On Relaxing Determinism in Arithmetic Circuits". In: *Proceedings of ICML*, pp. 825–833.
- Darwiche, Adnan (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge.
- Friesen, Abram L. and Pedro M. Domingos (2016). "The Sum-Product Theorem: A Foundation for Learning Tractable Models". In: *CoRR* abs/1611.03553.
- Frisen, M. (1986). "Unimodal Regression". In: *Journal of the Royal Statistical Society. Series D (The Statistician)* 35.4, pp. 479–485. ISSN: 00390526, 14679884. URL: `http://www.jstor.org/stable/2987804`.
- Gens, Robert and Pedro Domingos (2013). "Learning the Structure of Sum-Product Networks". In: *Proceedings of the ICML 2013*, pp. 873–880.
- — (2017). "Compositional Kernel Machines". In: *Proceedings of the 5th International Conference on Learning Representations*.
- Jaini, Priyank et al. (2016). "Online Algorithms for Sum-Product Networks with Continuous Variables". In: *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings*, pp. 228–239. URL: `http://jmlr.org/proceedings/papers/v52/jaini16.html`.
- Lopez-Paz, David, Philipp Hennig, and Prof. Bernhard Schölkopf (2013). "The Randomized Dependence Coefficient". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al., pp. 1–9.
- Marlin, Benjamin M et al. (2010). "Inductive Principles for Restricted Boltzmann Machine Learning". In: *AISTATS 2010*, pp. 509–516.
- Molina, Alejandro, Sriraam Natarajan, and Kristian Kersting (2017). "Poisson Sum-Product Networks: A Deep Architecture for Tractable Multivariate Poisson Distributions". In: *AAAI*. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14530`.

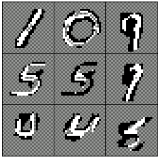
References II

- Molina, Alejandro et al. (2017). "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains". In: *AAAI*.
- Peharz, Robert, Robert Gens, and Pedro Domingos (2014). "Learning Selective Sum-Product Networks". In: *Workshop on Learning Tractable Probabilistic Models*. LTPM.
- 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*.
- Rényi, Alfréd (1959). "On measures of dependence". In: *Acta mathematica hungarica* 10.3-4, pp. 441–451.
- 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*.
- Rozenholc, Yves, Thoralf Mildenberger, and Ursula Gather (2010). "Combining regular and irregular histograms by penalized likelihood". In: *Computational Statistics & Data Analysis* 54.12, pp. 3313–3323.
- Vergari, A. et al. (2017). "Sum-Product Autoencoding: Encoding and Decoding Representations using Sum-Product Networks". In:
- 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

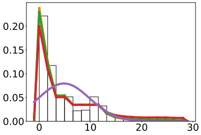


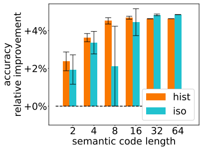




$$\mathbf{S}^{\oplus}(\mathbf{x}) = \log(\mathbf{W}\mathbf{x})$$

$$\mathbf{S}^{\otimes}(\mathbf{x}) = \exp(\mathbf{P}\mathbf{x})$$





SPNs as NNs

SPNs as *sparse*, *constrained* NNs with a *fully probabilistic* semantics and allowing for direct encoding through the scope function.

A classic MLP hidden layer computes first a linear and then a non-linear mapping:

$$h(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

SPNs can be reframed as *DAGs* of MLPs, each sum layer of s nodes computing:

$$\mathbf{S}^{\oplus}(\mathbf{x}) = \log(\mathbf{W}\mathbf{x})$$

and similarly for product layers:

$$\mathbf{S}^{\otimes}(\mathbf{x}) = \exp(\mathbf{P}\mathbf{x})$$

where $\mathbf{W} \in \mathbb{R}_{+}^{s \times r}$ and $\mathbf{P} \in \{0, 1\}^{s \times r}$ are the weight connection matrices:

$$\mathbf{W}_{(ij)} = \begin{cases} w_{ij} & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{P}_{(ij)} = \begin{cases} 1 & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases}$$

Vergari, Di Mauro, et al., “Visualizing and Understanding Sum-Product Networks”, 2016