Università degli Studi di Bari
Dipartimento di Informatica

LACAM
Machine Learning

Learning Sum-Product Networks

Nicola Di Mauro **Antonio Vergari**

September 2016

The need for SPN

Sum-Product Networks (SPNs) are a type of probabilistic model[1]

- for Pobabilistic Graphical Models (PGMs) there exist multi-purpose inference tools
  - the computational effort scales unproportional to the complexity of the graph
  - solution: using approximate inference

[1] H. Poon and P. Domingos, *Sum-Product Network: a New Deep Architecture*, UAI 2011

The need for SPNs

Why should you work on SPNs?

- exact tractable inference
- NN for which structure learning is easy

SPNs represent probability distributions and a corresponding exact inference machine for the represented distribution at the same time

Representation

Density estimation

(Different kinds of) Inference

Different kinds of queries:

- $p(\mathbf{X})$ (evidence)
- $p(\mathbf{E}), \mathbf{E} \subset \mathbf{X}$ (marginals)
- $p(\mathbf{Q}|\mathbf{E}), \mathbf{Q}, \mathbf{E} \subset \mathbf{X}, \mathbf{Q} \cap \mathbf{E} = \emptyset$ (conditionals)
- $\arg\max_{\mathbf{q} \sim \mathbf{Q}} p(\mathbf{q}|\mathbf{E})$ (MPE assignment)
- complex queries

Tractable Probabilistic Models

Sum-Product Networks

Scopes

Structural Properties

Inference

Complete evidence

Marginal inference

MPE inference

Interpretation

Interpretation

- probabilistic model
- deep feedforward neural network

Network Polynomials

Arithmetic Circuits

Differences with ACs:

- probabilistic semantics
  - learning
  - sampling
- no shared weights

SPNs as NNs (I)

SPNs are a particular kind of ***labelled constrained*** *and* ***fully probabilistic*** neural networks.

Labelled: each neuron is associated a *scope*
Constrained: completeness and decomposability determine network topology.
Fully probabilistic: each valid sub-SPN is still a valid-SPN.

SPNs provide a direct encoding of the input space into a deep architecture $\rightarrow$ ***visualizing representations*** (back) into the ***input space***.

A. Vergari et al. [2016]. "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*.

SPNs as NNs (II)

A classic MLP hidden layer computes the function:

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

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

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

and product layers computing:

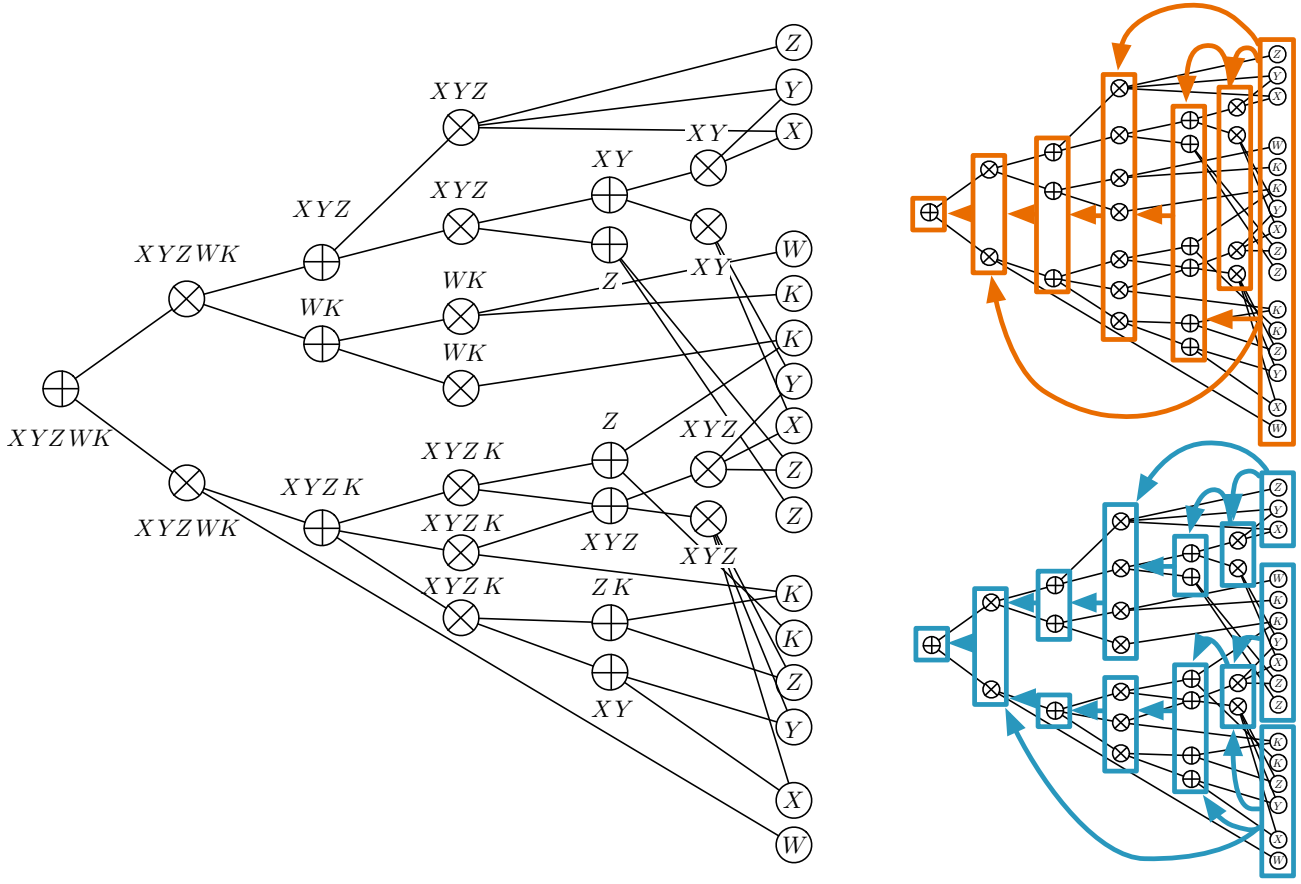
$$\mathbf{S}(\mathbf{x}) = \exp(\mathbf{P}\mathbf{x})$$

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

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

A. Vergari et al. [2016]. "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*.

SPNs as NNs (III)



SPNs as NNs (IV): filters

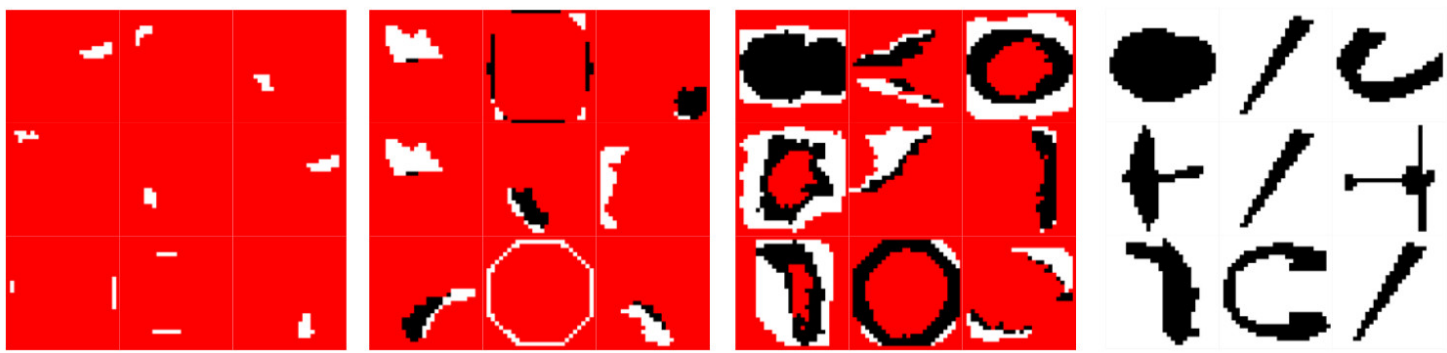
Learned features as images maximizing neuron activations [Erhan et al. 2009]:

$$\mathbf{x}^* = \underset{\mathbf{x}, ||\mathbf{x}||=\gamma}{\operatorname{argmax}} \, h_{ij}(\mathbf{x}; \boldsymbol{\theta}).$$

With SPNs, joint solution as an MPE assignment for all nodes (linear time):

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

.



$\rightarrow$ *scope length* ($|\mathsf{sc}(n)|$) correlates with feature abstraction level

---

A. Vergari et al. [2016]. "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*.

SPNs as BNs I

Zhao and Poupart

SPNs as BNs II

Peharz

Myths about SPNs

SPNs are PGMs. false

SPNs are convolutional NNs. false

SPNs

Learning

Learning SPNs

Parameter learning: estimate $\mathbf{w}$ from data considering an SPN as a latent RV model, or as a NN

Structure learning: build the network from data by assigning scores to tentative structures or by exploiting constraints

How to learn a “complete” SPN:

- handcrafted structure, then parameter learning [Gens and Domingos 2012; Poon and Domingos 2011]
- random structures, then parameter learning [Rashwan et al. 2016]
- structure learning, then parameter learning (fine tuning) [Zhao, Adel, et al. 2016]
- learn both weight and structure at the same time [Adel et al. 2015; Gens and Domingos 2013; Rooshenas and Lowd 2014; Antonio Vergari et al. 2015]...

Structure Learning

Score vs constraint based search. No closed form for likelihood scores, need heuristics [Rooshenas and Lowd 2014].
No need for it by exploiting the inner nodes probabilistic semantics

Learning graph vs tree structures: Easier to learn a tree SPN (sometimes SPT) with greedy approaches. Graph SPNs may be more compact and expressive efficient.

Top-down vs bottom-up approaches: iteratively cluster data matrix (top-down) or start by the marginal RVs (bottom-up)

LearnSPN [Gens and Domingos 2013] is a ***greedy***, ***top down***, ***constraint based*** learner for ***tree*** SPNs
$\rightarrow$ First principled top-down learner, inspired many algorithms and variations.
$\rightarrow$ Surprisingly simple and accurate.

LearnSPN (I)

Build a tree SPN by recursively split the data matrix:

- splitting columns into pairs by a greedy ***G Test***-based procedure with threshold ρ:

$$G(X_i, X_j) = 2 \sum_{x_i \sim X_i} \sum_{x_j \sim X_j} c(x_i, x_j) \cdot \log \frac{c(x_i, x_j) \cdot |T|}{c(x_i) c(x_j)}$$

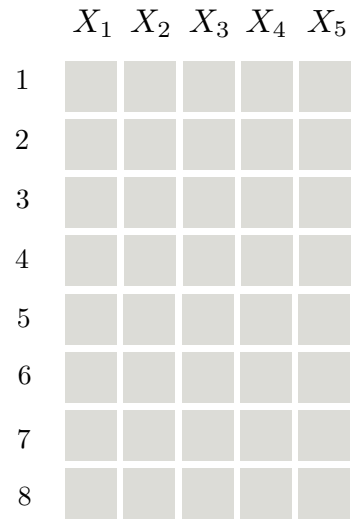
- clustering instances into $|C|$ sets with ***online Hard-EM*** with cluster penalty λ:

$$Pr(\mathbf{X}) = \sum_{C_i \in \mathbf{C}} \prod_{X_j \in \mathbf{X}} Pr(X_j | C_i) Pr(C_i)$$

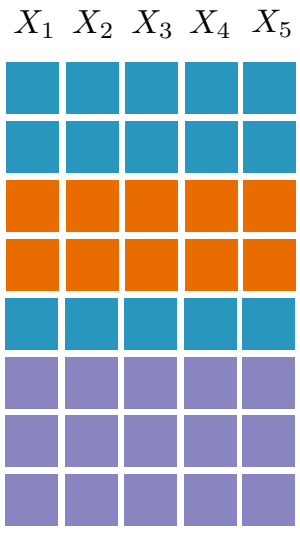
  weights are estimated as cluster proportions
- if there are less than m instances, put a ***naive factorization*** over leaves
- each univariate distribution get ***ML estimation*** smoothed by α

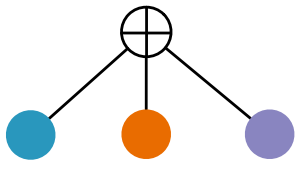
Hyperparameter space: $\{\rho, \lambda, m, \alpha\}$.

LearnSPN (II)

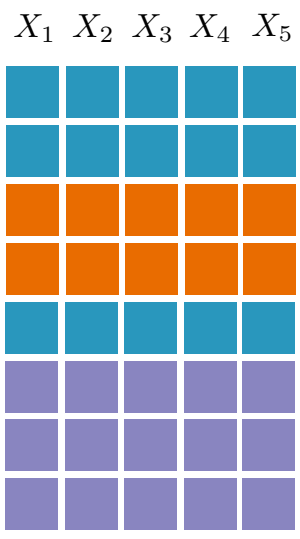


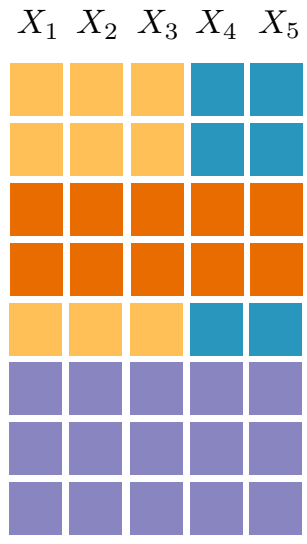
LearnSPN (II)

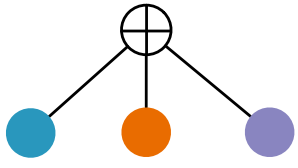


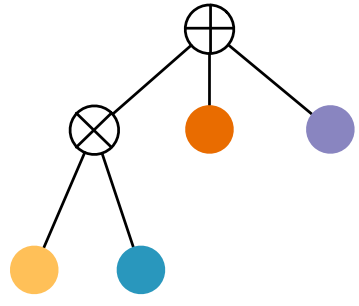


LearnSPN (II)

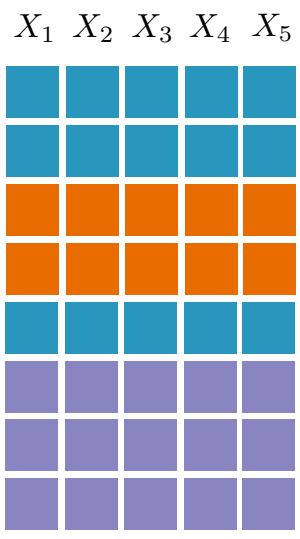


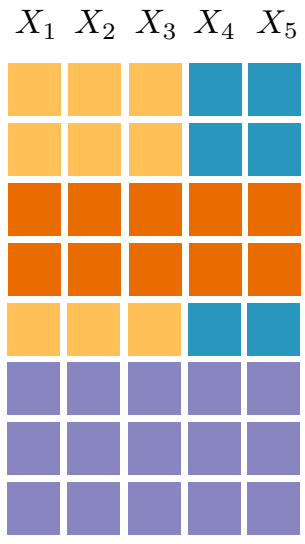


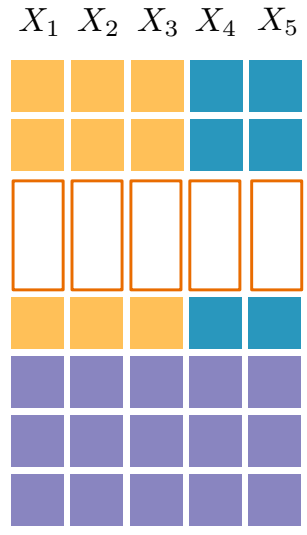


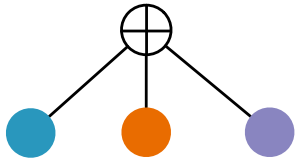


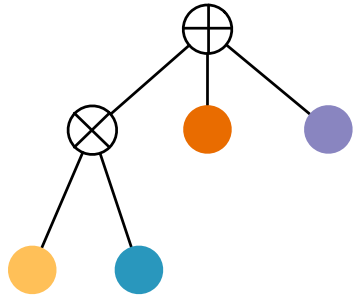
LearnSPN (II)

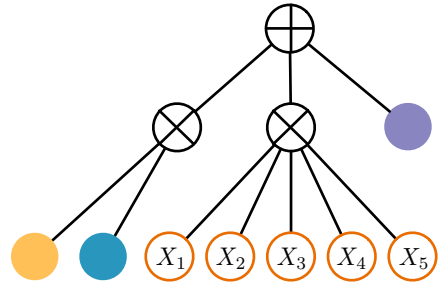












Tweaking LearnSPN

LearnSPN performs two interleaved greedy ***hierarchical divisive clustering*** processes (co-clutering on the data matrix).

Fast and simple. But both processes never look back and are committed to the choices they take $\rightarrow$ *slow down the two processes*

Online EM does not need to specify the number of clusters k in advance. But overcomplex structures are learned by exploding the number of sum node children $\rightarrow$ *look for deeper networks*

Tractable leaf estimation. But too strong naive factorization independence assumptions, hard to regularize $\rightarrow$ *learn tree distributions as leaves*

ML estimations are effective. But they are not robust to noise, they can overfit the training set easily $\rightarrow$ *learn bagged sum nodes*

Why Structure Quality Matters

Tractable inference is guaranteed *if the network size is polynomial* in $|\mathbf{X}|$.

Network ***size*** influences inference complexity: smaller networks, faster inference!
$\rightarrow$ Comparing network sizes is better than comparing inference times

Network ***depth*** influences *expressive efficiency [Martens and Medabalimi 2014; Zhao, Melibari, et al. 2015]*.

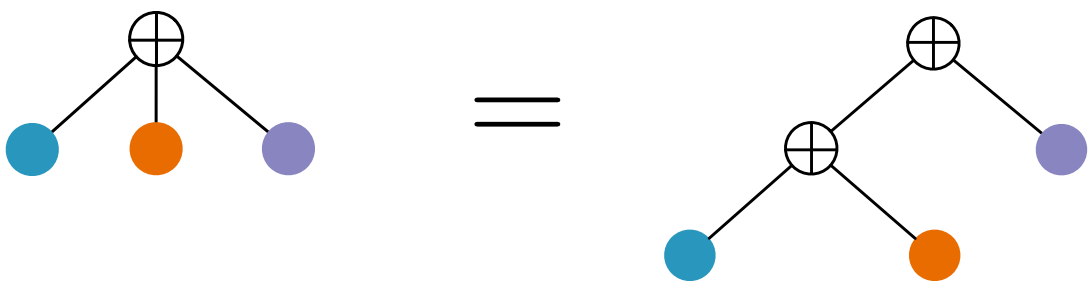
Structural simplicity as a bias: overcomplex networks may not generalize well.

Structure quality desiderata: **smaller** but **accurate**, ***deeper*** but not wider, SPNs.

LearnSPN-b

Observation: each clustering process benefits from the other one improvements/highly suffers from other's mistakes.

Idea: slow them down the processes by limiting the number of nodes to split to the minimum. LearnSPN-b, binary splitting $k = 2$.
$\rightarrow$ one hyperparameter less, λ.
$\rightarrow$ not committing to complex structures too early
$\rightarrow$ reducing node out fan increases the depth
$\rightarrow$ same expressive power as LearnSPN structures
$\rightarrow$ statistically same (or better) accuracy, smaller networks



LearnSPN-b: depth VS size

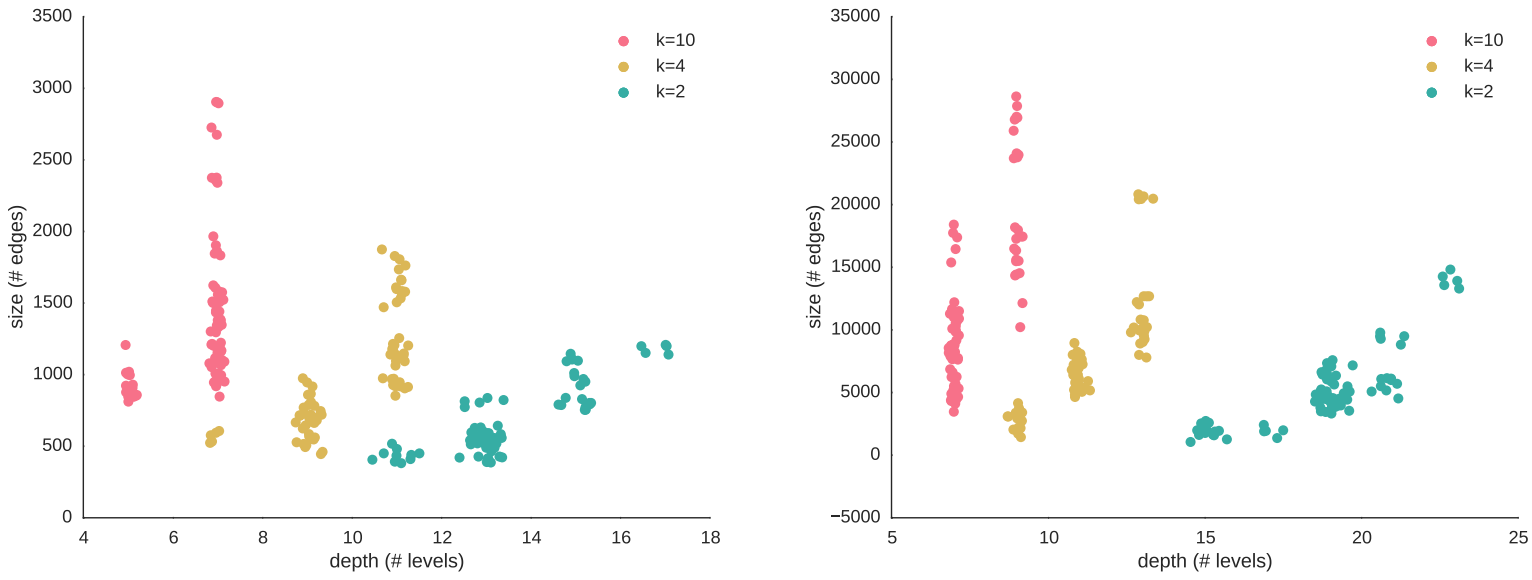


Figure : Network sizes VS depths while varying the max number of sum node children splits ($k \in \{10, 4, 2\}$). Each dot is an experiment in the grid search hyperparameter space performed by LearnSPN-b on NLTCS (left) and Plants (right).

LearnSPN-b: best ll VS size

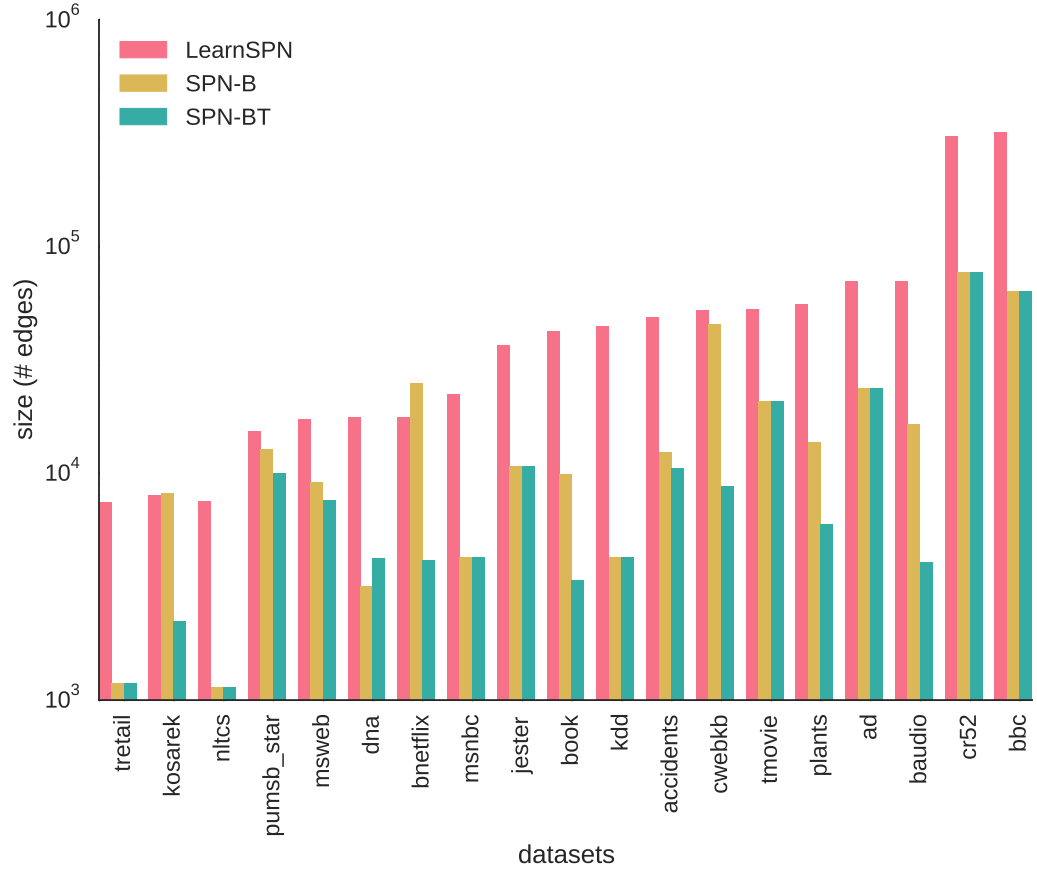


Figure : Comparing network sizes for the networks scoring the best log-likelihoods in the grid search as obtained by LearnSPN, LearnSPN-b and LearnSPN-bT for each dataset.

Other variations on LearnSPN

ACs modeling leaves by performing a greedy score search. ***ID-SPN*** best log-likelihood learner (but lots of hyperparameters).
Freely available in the Libra[2] toolkit. [Rooshenas and Lowd 2014]

Looking for **correlations instead of independencies** via matrix factorizations.
Splitting matrix rows and columns at the same time: SPN-SVD.
It can cope with continuous data [Adel et al. 2015]

Post-learning **mergining sub-SPNs** that model "similar" distributions.
Reducing network sizes [Rahman and Gogate 2016].

Learning Relational SPNs on **first order data** represented in Tractable Markov Logic (TML), LearnRSPN [Nath and Domingos 2015].

---

[2] http://libra.cs.uoregon.edu/

Other Tendencies in Structure Learning

Learning deterministic structures which enable closed form log-likelihood and weight estimation.
Selective SPNs, enabling efficient Stochastic Local Search [Peharz, Gens, and Domingos 2014].
Mixing latent and deterministic mixtures as sum nodes (a Cutset Network is an SPN!). [Rahman and Gogate 2016]

Learning DAGs structures instead of trees.
Substituting sub-structures with more complex ones by cloning mixtures. [Dennis and Ventura 2015]

Template learning for sequence models. Stochastic local search over well defined constrained structures. Dynamic SPNs [Melibari et al. 2016]$\longrightarrow$ *PGM'16!*

Parameter Learning

Non convex optimization, solvable with (online) iterative methods (e.g. SGD)

Classical approach: compute the gradient $\nabla_{\mathbf{w}} S(\mathbf{x})$
$\rightarrow$ use backpropagation (differential approach[3])

1. $\nabla_{S(\mathbf{x})} S(\mathbf{x}) \leftarrow 1$ start from the root
2. if n is a sum node, $\forall_{c \in \mathsf{ch}(n)}$:
   $\nabla_{S_c(\mathbf{x})} S(\mathbf{x}) \leftarrow \nabla_{S_c(\mathbf{x})} S(\mathbf{x}) + w_{nc} \nabla_{S_n(\mathbf{x})} S(\mathbf{x})$
3. if n is a product node, $\forall_{c \in \mathsf{ch}(n)}$:
   $\nabla_{S_c(\mathbf{x})} S(\mathbf{x}) \leftarrow \nabla_{S_c(\mathbf{x})} S(\mathbf{x}) + \nabla_{S_n(\mathbf{x})} S(\mathbf{x}) \prod_{k \in \mathsf{ch}(n) \setminus \{c\}} S_k(\mathbf{x})$

Issues:

- vanishing gradients: depth is a major problem for *soft* gradients
- hyperparameter choices
  - adaptive learning rate scheduling algos not employed yet!

[3] Adnan Darwiche [2003]. "A Differential Approach to Inference in Bayesian Networks". In: *J.ACM*.

“Hard” gradients

From SPN S to MPN M

- forward (bottom-up) prop $\mathbf{x}^i$
- backprop as MPE descent
- “count” the weights in the path $W_{\mathbf{x}^i}$
  $\nabla_{w_{pc}} \log M(\mathbf{x}) = \frac{\sharp\{w_{pc} \in W_{\mathbf{x}}\}}{w_{pc}}$

Gradient is “hard”, does not vanishes regardless network depth
Much fewer parameters get updated/per instance
$\rightarrow$ slower convergence

Hoifung Poon and Pedro Domingos [2011]. “Sum-Product Networks: a New Deep Architecture”. In: *UAI 2011*.

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

Hard/Soft Parameter Updating

	Δw_{pc}
Soft Gradient	
Generative ($\nabla_{w_{pc}} S(\mathbf{x})$)	$S_c(\mathbf{x}) \nabla_{S_p(\mathbf{x})} S(\mathbf{x})$
Discriminative ($\nabla_{w_{pc}} \log S(\mathbf{y}\|\mathbf{x})$)	$\frac{\nabla_{w_{pc}} S(\mathbf{y}\|\mathbf{x})}{S(\mathbf{y}\|\mathbf{x})} - \frac{\nabla_{w_{pc}} S(*\|\mathbf{x})}{S(*\|\mathbf{x})}$
Hard Gradient	
Generative ($\nabla_{w_{pc}} \log M(\mathbf{x})$)	$\frac{\sharp\{w_{pc} \in W_{\mathbf{x}}\}}{w_{pc}}$
Discriminative ($\nabla_{w_{pc}} \log M(\mathbf{y}\|\mathbf{x})$)	$\frac{\sharp\{w_{pc} \in W_{(\mathbf{y}\|\mathbf{x})}\} - \sharp\sharp\{w_{pc} \in W_{(\mathbf{1}\|\mathbf{x})}\}}{w_{pc}}$
Soft Posterior[4] ($p(H_p = c\|\mathbf{x})$)	$\propto \frac{1}{S(\mathbf{x})} \frac{\partial S(\mathbf{x})}{\partial S_p(\mathbf{x})} S_c(\mathbf{x}) w_{pc}$
Hard Posterior ($p(H_p = c\|\mathbf{x})$)	$= \begin{cases} 1 \text{if} w_{pc} \in W_{\mathbf{x}} \\ 0 \text{otherwise} \end{cases}$

[4] Robert Peharz, Robert Gens, Franz Pernkopf, et al. [2016]. "On the Latent Variable Interpretation in Sum-Product Networks". In: *CoRR* abs/1601.06180. URL: `http://arxiv.org/abs/1601.06180`.

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

Bayesian Parameter Learning

Learning in a Bayesian setting is computing the posterior $p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^m)$ having a prior $p(\mathbf{w})$:

$$p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^{t+1}) \propto p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^{t})p(\mathbf{x}^{t+1}|\mathbf{w})$$

$p(\mathbf{w})$ modeled as a product of Dirichlet, $p(\mathbf{x}^{t+1}|\mathbf{w})$ is an exponential sum of monomials, $\rightarrow$ the posterior becomes a mixture of products of Dirichlets growing exponentially in the data and sum nodes!

Online Bayesian Moment Matching (OBMM): computing first two moments to approximate the intractable posterior, efficiently for tree SPNs [Rashwan et al. 2016]

Collapse Variational Inference (CVB-SPN) to optimize a logarithmic lower bound (better than ELBO) efficiently (linear in $|S|$) [Zhao, Adel, et al. 2016].

Parameter learning

	CVB-SPN[5]	OBMM[6]	SGD[7]	EM[8]	SEG[9]
NLTCS	-6.08	-6.07	-8.76	-6.31	-6.85
MSNBC	-6.29	-6.03	-6.81	-6.64	-6.74
KDDCup2k	-2.14	-2.14	-44.53	-2.20	-2.34
Plants	-12.86	-15.14	-21.50	-17.68	-33.47
Audio	-40.36	-40.70	-49.35	-42.55	-46.31
Jester	-54.26	-53.86	63.89	-54.26	-59.48
Netflix	-60.69	-57.99	64.27	-59.35	-64.48
Accidents	-29.55	-42.66	53.69	-43.54	-45.59
Retail	-10.91	-11.42	-97.11	-11.42	-14.94
Pumsb-star	-25.93	-45.27	-128.48	-46.54	-51.84
DNA	-86.73	-99.61	-100.70	-100.10	-105.25
Kosarek	-10.70	-11.22	34.64	-11.87	-17.71
MSWeb	-9.89	-11.33	-59.63	-11.36	-20.69
Book	-34.44	-35.55	-249.28	-36.13	-42.95
EachMovie	-52.63	-59.50	-227.05	-64.76	-84.82
WebKB	-161.46	-165.57	-338.01	-169.64	-179.34
Reuters-52	-85.45	-108.01	-407.96	-108.10	-108.42
20-Newsgrp	-155.61	-158.01	-312.12	-160.41	-167.89
BBC	-251.23	-275.43	-462.96	-274.82	-276.97
Ad	-19.00	-63.81	-638.43	-63.83	-64.11

[5] Zhao, Adel, et al. 2016.

Parameter learning VS LearnSPN

	LearnSPN[10]	LearnSPN-b[11]	CVB-SPN[12]	OBMM[13]	SGD[14]	EM[15]	SEG[16]
NLTCS	-6.11	-6.05	-6.08	-6.07	-8.76	-6.31	-6.85
MSNBC	-6.11	-6.04	-6.29	-6.03	-6.81	-6.64	-6.74
KDDCup2k	-2.18	-2.14	-2.14	-2.14	-44.53	-2.20	-2.34
Plants	-12.98	-12.81	-12.86	-15.14	-21.50	-17.68	-33.47
Audio	-40.50	-40.57	-40.36	-40.70	-49.35	-42.55	-46.31
Jester	-53.48	-53.53	-54.26	-53.86	63.89	-54.26	-59.48
Netflix	-57.33	-57.73	-60.69	-57.99	64.27	-59.35	-64.48
Accidents	-30.04	-29.34	-29.55	-42.66	53.69	-43.54	-45.59
Retail	-11.04	-10.94	-10.91	-11.42	-97.11	-11.42	-14.94
Pumsb-star	-24.78	-23.31	-25.93	-45.27	-128.48	-46.54	-51.84
DNA	-82.52	-81.91	-86.73	-99.61	-100.70	-100.10	-105.25
Kosarek	-10.99	-10.72	-10.70	-11.22	34.64	-11.87	-17.71
MSWeb	-10.25	-9.83	-9.89	-11.33	-59.63	-11.36	-20.69
Book	-35.89	-34.30	-34.44	-35.55	-249.28	-36.13	-42.95
EachMovie	-52.49	-51.36	-52.63	-59.50	-227.05	-64.76	-84.82
WebKB	-158.20	-154.28	-161.46	-165.57	-338.01	-169.64	-179.34
Reuters-52	-85.07	-83.34	-85.45	-108.01	-407.96	-108.10	-108.42
20-Newsgrp	-155.93	-152.85	-155.61	-158.01	-312.12	-160.41	-167.89
BBC	-250.69	-247.30	-251.23	-275.43	-462.96	-274.82	-276.97
Ad	-19.73	-16.23	-19.00	-63.81	-638.43	-63.83	-64.11

[10] Gens and Domingos 2013.

[11] Antonio Vergari et al. 2015

Why learning parameters only

Even if simple, LearnSPN hardly scales on large datasets.

$\longrightarrow$ generate a random (but valid) structure, then optimize the weights

	LearnSPN	OBMM	ODMM	SGB	OEM	OEG
KOS	-444.55	**-422.19**	-437.30	-3492.9	-538.21	-657.13
NIPS	-	**-1691.87**	-1709.04	-7411.20	-1756.06	-3134.59
ENRON	-	**-518.842**	-522.45	-13961.40	-554.97	-14193.90
NyTIMES	-	-1503.65	-1559.39	-43153.60	**-1189.39**	-6318.71

$\longrightarrow$ distribute the computation of gradients and updates (over instances,...etc)

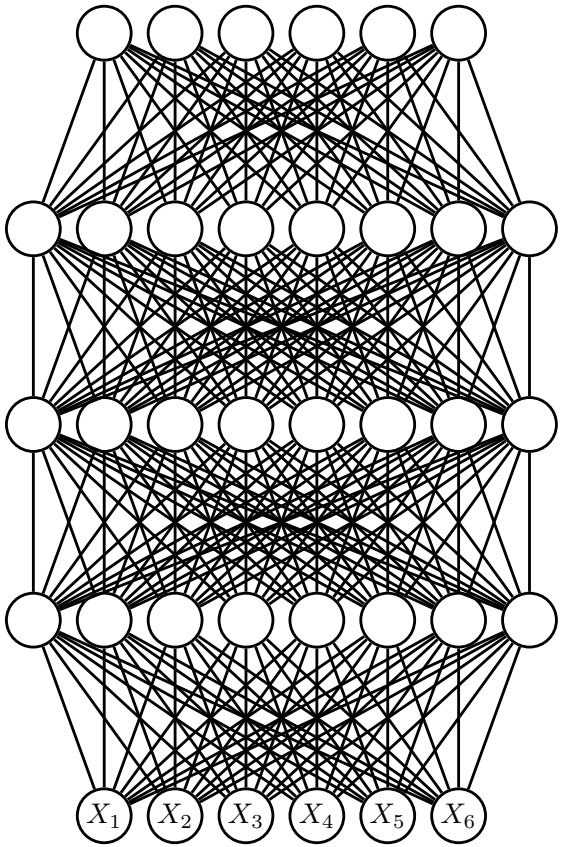
	LearnSPN	OBMM	ODMM	SGB	OEM	OEG
KOS	1439.11	89.40	**8.66**	162.98	59.49	155.34
NIPS	-	139.50	**9.43**	180.25	64.62	178.35
ENRON	-	2018.05	**580.63**	876.18	694.17	883.12
NyTIMES	-	12091.7	**1643.60**	5626.33	5540.40	6895.00

Abdullah Rashwan et al. [2016]. "Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1469–1477.

Representation Learning

Extracting Embeddings

From deep neural networks



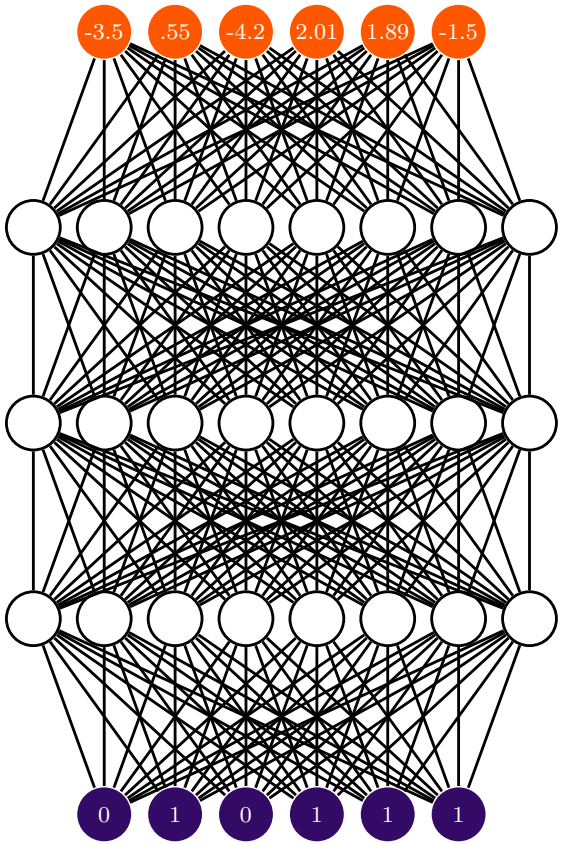
Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

Yoshua Bengio et al. [2012]. "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives". In: *CoRR* abs/1206.5538.

Extracting Embeddings

From deep neural networks



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating the network and collecting the last layer(s) activations

$$\mathbf{e}^i = \langle -3.5, .55, -4.2, 2.01, 1.89, -1.5 \rangle$$

Yoshua Bengio et al. [2012]. "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives". In: *CoRR* abs/1206.5538.

Extracting Embeddings

Exploiting SPNs as feature extractors

Given an SPN S, a filtering criterion f, generate a dense vector for each sample $\mathbf{x}^i$

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

Issues with SPNs as NNs:

- layer-wise extraction may be arbitrary
- power law distribution of nodes by scopes
- scope lengths as proxy for feature abstraction levels (see filter visualizations)

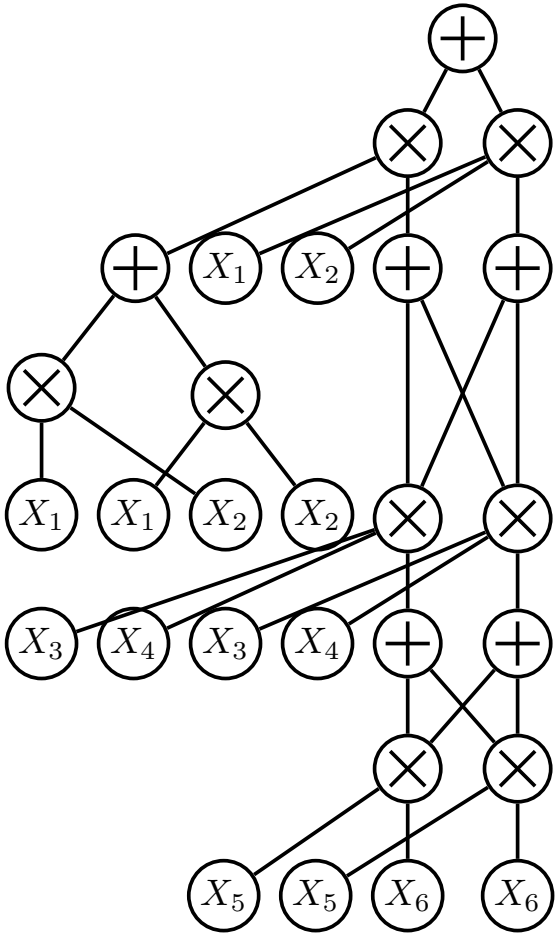
Which filtering criterion to employ?

Which interpretation for the extracted features?

A. Vergari et al. [2016]. "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*.

Extracting embeddings

Inner node activations

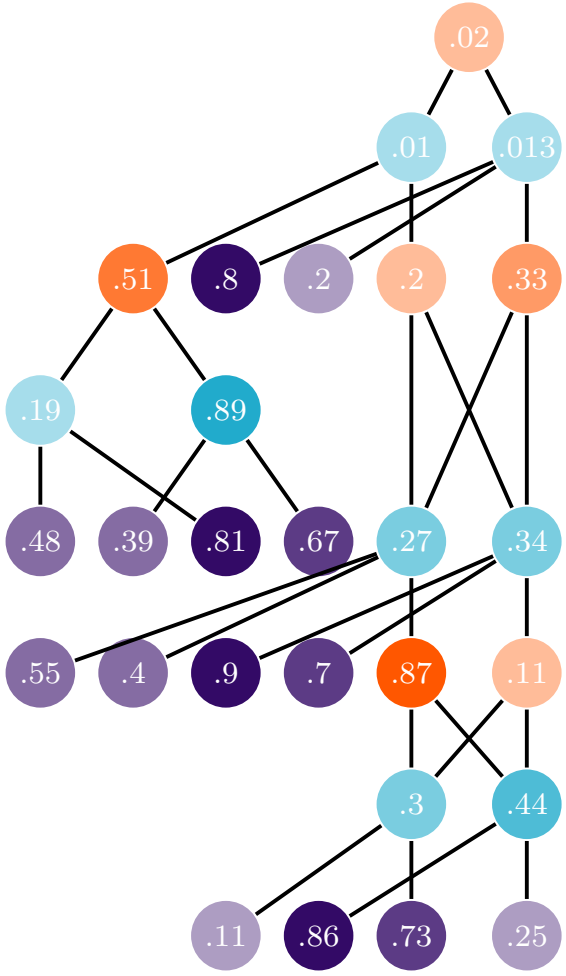


Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

Extracting embeddings

Inner node activations



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

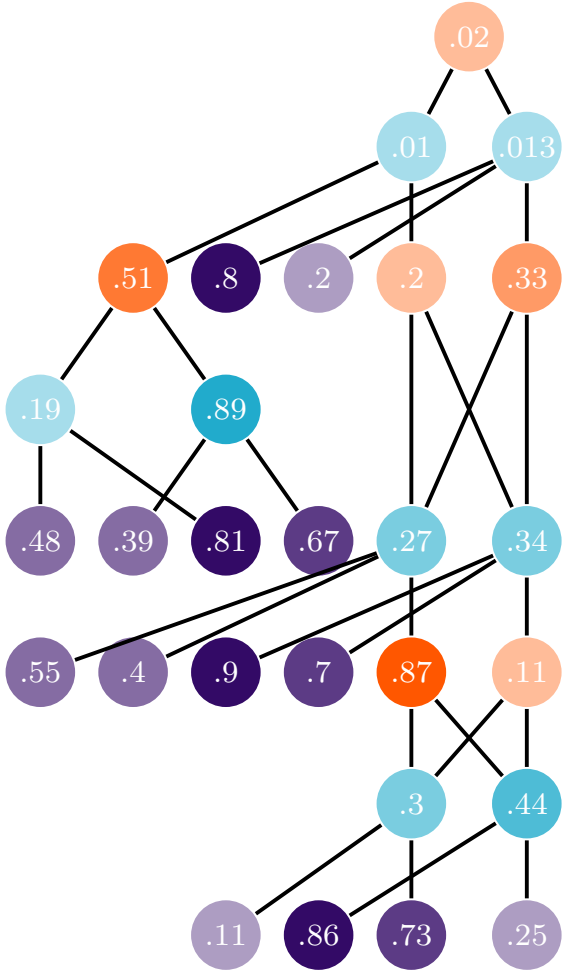
$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating $S(\mathbf{x}^i)$ and collecting inner node (sum , product but not for leaf nodes) activations

$$\mathbf{e}^i = \langle .55, .66, .55, .66, .55, .66, .55, .66, .55, .66, .55, .66, .55, .66 \rangle$$

Extracting embeddings

Filtering by type



Build embeddings $\mathbf{e}^i_{\text{sum}}, \mathbf{e}^i_{\text{prod}}$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

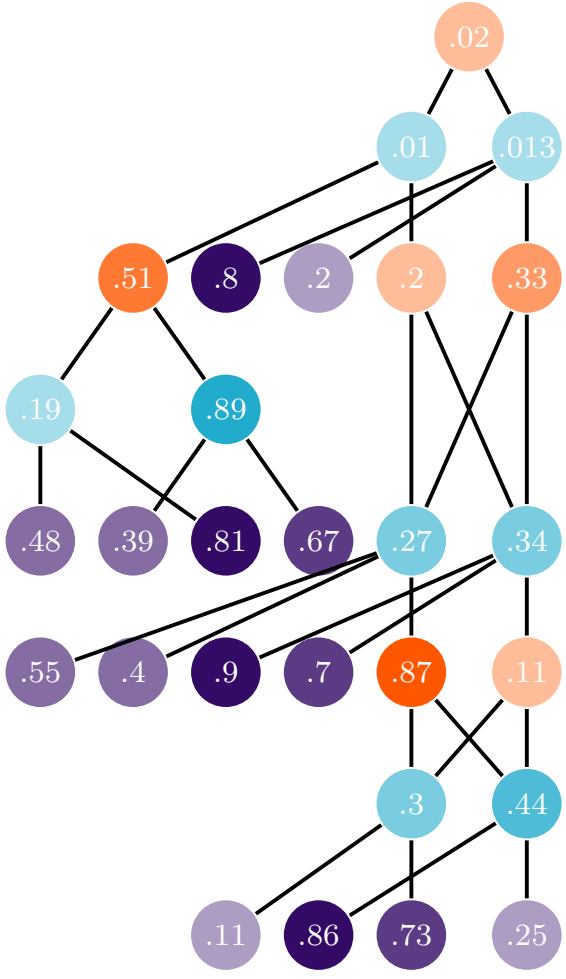
by evaluating $S(\mathbf{x}^i)$ and collecting inner node activations filtered by node type

$$\mathbf{e}^i_{\text{sum}} = \langle .55, .66, .55, .66, .55, .66 \rangle$$

$$\mathbf{e}^i_{\text{prod}} = \langle .55, .66, .55, .66, .55, .66, .55, .66 \rangle$$

Extracting embeddings

Filtering by scope length



Build embeddings $\mathbf{e}^i_{|\mathsf{sc}(n)|=k} \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating $S(\mathbf{x}^i)$ and collecting inner node activations filtered by scope length

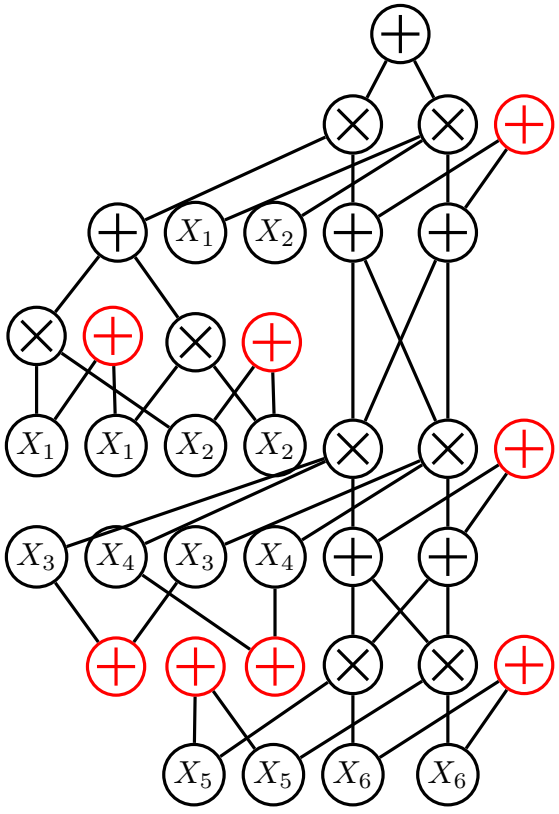
$$\mathbf{e}^i_{|\mathsf{sc}(n)|=2} = \langle .55, .66, .55, .66, .55, .66, .66 \rangle$$

$$\mathbf{e}^i_{|\mathsf{sc}(n)|=4} = \langle .55, .66, .55, .66 \rangle$$

$$\mathbf{e}^i_{|\mathsf{sc}(n)|=6} = \langle .55, .66, .55 \rangle$$

Extracting embeddings

Aggregating by scope



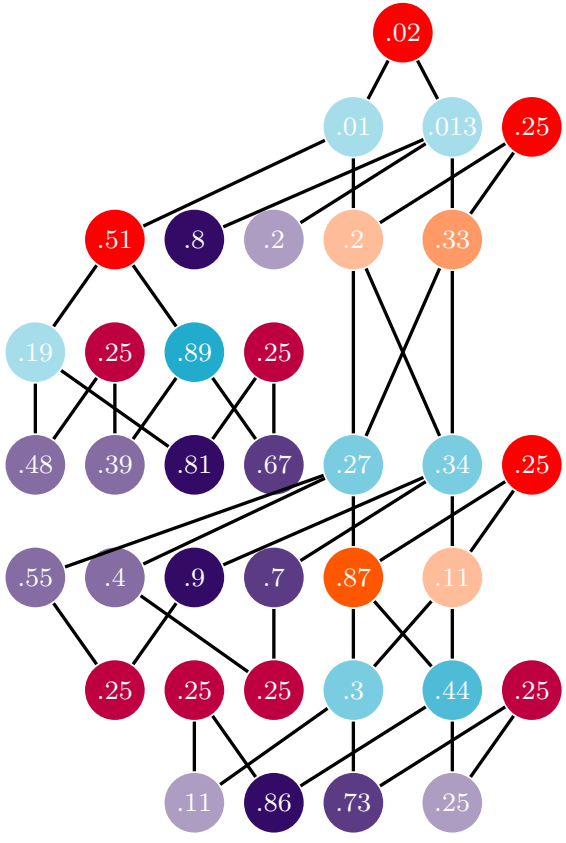
Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by adding fictitious sum nodes over unique scopes

Extracting embeddings

Aggregating by scope



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by adding fictitious sum nodes over unique scopes, then evaluating $S(\mathbf{x}^i)$ and collecting they activations from (inner nodes, and even leaves

$$\mathbf{e}^i_{\text{w/o}-\text{leaves}} = \langle .55, .66, .55, .66 \rangle$$

$$\mathbf{e}^i_{\text{w}-\text{leaves}} = \langle .55, .66, .55, .66, .55, .66, .55, .66 \rangle$$

Extracting embeddings

Random marginal queries

Supervised classification

Experimental settings to evaluate embeddings

Extract embeddings unsupervisedly on $\mathbf{X}$, then train a logistic regressor on them to predict Y.

Five image datasets: REC, CON, OCR, CAL, BMN.

Grid search with LearnSPN-b for three models with different capacities: SPN-I, SPN-II and SPN-III for $m \in \{500, 100, 50\}$.

Compare them against RBM models: RBM-5h, RBM-1k and RBM-5k with 500, 1000 and 5000 hidden units.

Compare them against other tractable PGMs: mixtures of 3, 15, 30 Chow-Liu trees.

Scope distribution

Embeddings (I)

Table : Test set accuracy scores for the embeddings extracted with the best SPN, RBM models and with the baseline LR model on all datasets. Bold values denote significantly better scores than all the others for a dataset.

	LR	SPN-I	SPN-II	SPN-III	RBM-5h	RBM-1k	RBM-5k
REC	69.28	77.31	**97.77**	97.66	94.22	96.10	96.36
CON	53.48	67.48	78.31	**84.69**	67.55	75.37	79.15
OCR	75.58	82.60	**89.95**	**89.94**	86.07	87.96	88.76
CAL	62.67	59.17	65.19	66.62	67.36	**68.88**	67.71
BMN	90.62	95.15	**97.66**	97.59	96.09	96.80	97.47

Embeddings (II)

Table : Test set accuracy scores for the embeddings extracted with SPN models and filtered by node type. Results for SPN-III embeddings filtered by Small , Medium and Large scope lengths are reported in columns 8-10. Bold values denote significantly better scores than all the others. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

	SPN-I		SPN-II		SPN-III		SPN-III		
	sum	prod	sum	prod	sum	prod	S	M	L
REC	72.46	62.25	**98.03**$^{\blacktriangle}$	97.06$^{\blacktriangle}$	**98.00**$^{\blacktriangle}$	97.04$^{\blacktriangle}$	88.73	**98.45**$^{\blacktriangle}$	93.91
CON	62.36	64.03	77.13$^{\blacktriangle}$	76.07$^{\blacktriangle}$	**83.59**$^{\blacktriangle}$	82.06$^{\blacktriangle}$	70.51$^{\triangledown}$	77.18	**83.32**$^{\blacktriangle}$
OCR	74.19	81.58	89.73$^{\blacktriangle}$	88.78$^{\blacktriangle}$	**90.02**$^{\blacktriangle}$	89.32	87.22$^{\triangledown}$	**89.29**$^{\blacktriangle}$	88.19$^{\blacktriangle}$
CAL	38.19	56.95	62.64	64.80	**66.58**$^{\triangledown}$	66.40$^{\triangledown}$	63.37$^{\triangledown}$	**66.23**$^{\triangledown}$	66.10
BMN	93.50	94.75	97.67	96.90$^{\triangledown}$	**97.80**	97.20$^{\triangledown}$	96.02$^{\triangledown}$	**97.42**$^{\triangledown}$	97.38

Embeddings (III)

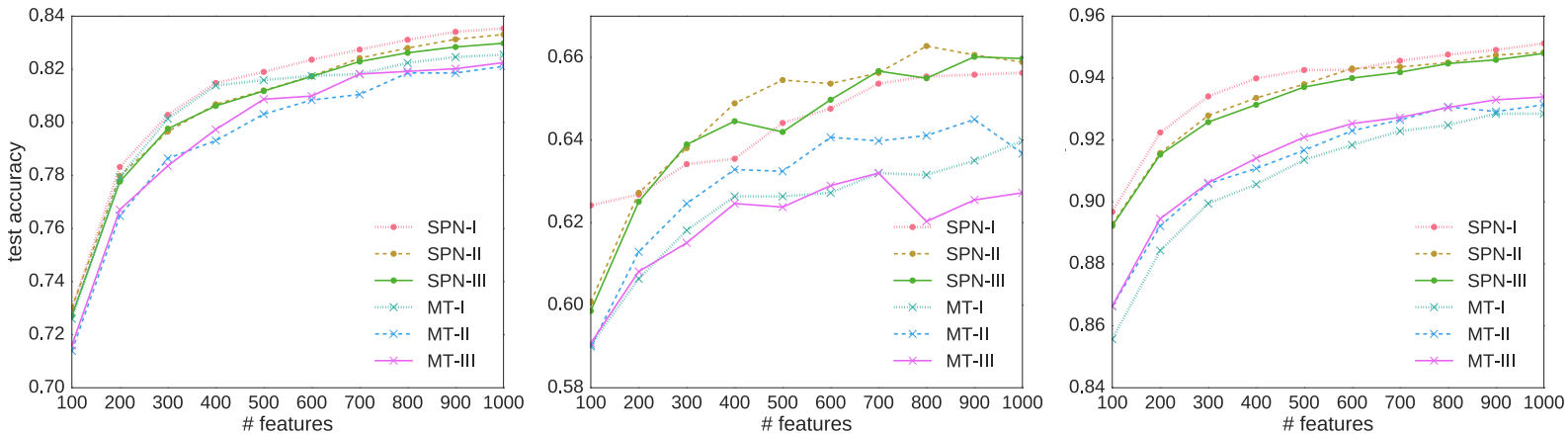
Table : Test set accuracy scores for the embeddings extracted with SPN models by aggregating node outputs with the same scope. Results for when leaves are not counted in the aggregarion (no-leaves columns) are reported alongside the case in which they are considered (leaves columns). Bold values denote significantly better scores than all the others for each dataset. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

	SPN-I		SPN-II		SPN-III	
	no-leaves	leaves	no-leaves	leaves	no-leaves	leaves
REC	72.47	$75.92^{\triangledown}$	$\mathbf{97.94}^{\blacktriangle}$	$\mathbf{97.99}^{\blacktriangle}$	$\mathbf{97.94}^{\blacktriangle}$	$\mathbf{98.02}^{\blacktriangle}$
CON	62.35	$66.49^{\triangledown}$	$77.21^{\blacktriangle}$	78.05	$\mathbf{83.52}^{\blacktriangle}$	$\mathbf{83.84}^{\blacktriangle}$
OCR	74.32	81.85	$89.71^{\blacktriangle}$	$89.68^{\blacktriangle}$	$\mathbf{89.90}^{\blacktriangle}$	$\mathbf{89.91}^{\blacktriangle}$
CAL	38.10	$63.19^{\triangledown}$	62.59	$62.76^{\triangledown}$	$\mathbf{66.49}^{\triangledown}$	$\mathbf{66.58}^{\triangledown}$
BMN	93.51	$94.83^{\triangledown}$	$97.64^{\blacktriangle}$	$97.62^{\blacktriangle}$	**97.80**	**97.80**

Random Marginal Queries

Generate embeddings by asking several random queries to a black box density estimator.

Eg. marginals: $e_j^i = P_\theta(\mathbf{Q}_j = \mathbf{x}^i_{\mathbf{Q}_j})$, according to estimator θ where $\mathbf{Q}_j \subseteq \mathbf{X}, j = \ldots, k$.



A. Vergari et al. [2016]. "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*.

Encoding/Decoding Embeddings

MPN as autoencoders[17].

[17] Vergari et al. Encoding and Decoding Representations with Sum-Product Networks, 2016, to appear

Applications

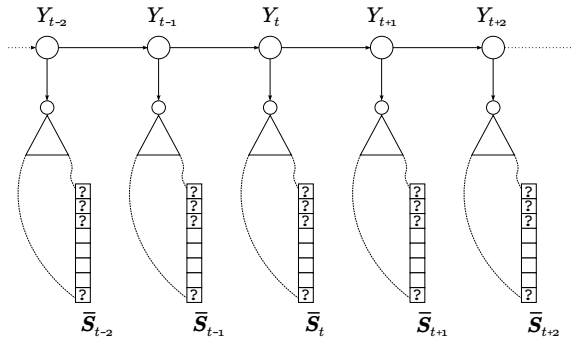
Applications I: computer vision

Applications II: language modeling

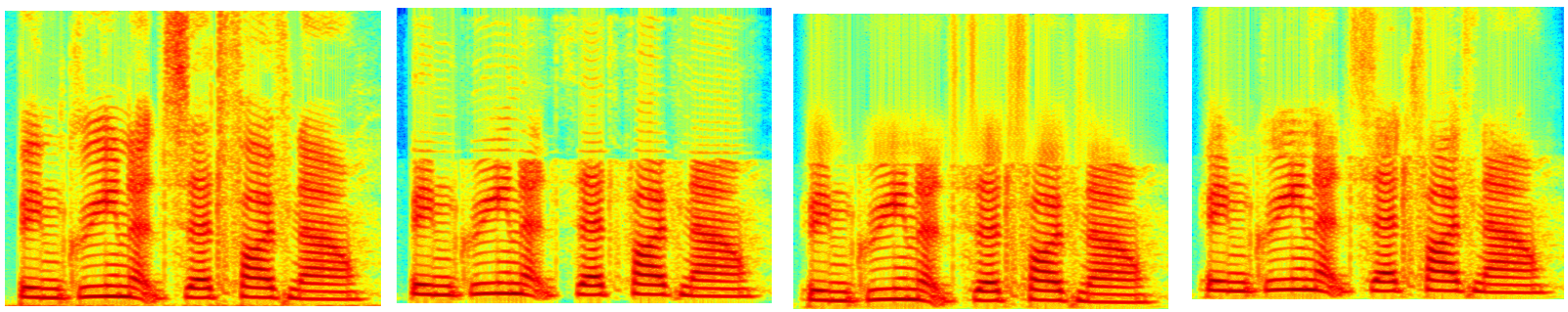
Applications III: activity recognition

Applications IV: speech

SPNs to model the joint pdf of observed RVs in HMMs (HMM-SPNs).



State-of-the-art high frequency reconstruction (MPE inference)



(a) Original full bandwidth (b) Reconstruction HMM-LP (c) Reconstruction HMM-GMM (d) Reconstruction HMM-SPN

Robert Peharz, Georg Kapeller, et al. [2014]. "Modeling speech with sum-product networks: Application to bandwidth extension". In: *ICASSP2014*.

Trends & What to do next

Scalable structure learning Continuous RVs structure learning
End-to-end training with hybrid NN architectures

References

awesome-spn

A curated and structured list of resources about SPNs[18].
`https://github.com/arranger1044/awesome-spn`

[18] Inspired by the SPN page http://spn.cs.washington.edu/ at the Washington University