Alternative Variable Splitting Methods to Learn Sum-Product Networks

University of Bari "Aldo Moro", Italy
Department of Computer Science
LACAM Machine Learning Lab

Nicola Di Mauro, Floriana Esposito, Fabrizio G. Ventola and Antonio Vergari *{firstname.lastname@uniba.it}*

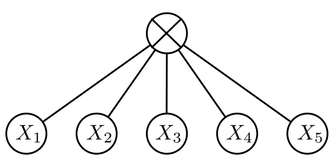
Sum-Product Networks

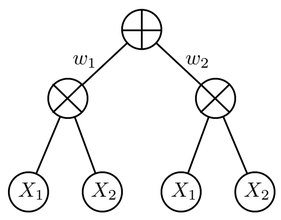
Probabilistic Graphical Models (PGMs) compactly represent a joint probability distribution $P(\mathbf{X})$. However, PGMs are not suitable as ***inference*** machines on many scenarios, since inference on PGMs is generally ***intractable***.

To ensure polynomial inference, tractable models trade off expressiveness. The main aim of ***Sum-Product Networks*** (**SPNs**) [5] is to keep both tractability and a good level of expressiveness.

SPNs are DAGs *compiling* $P(\mathbf{X})$ into a ***deep*** architecture of **sum** and **product** nodes with univariate distributions on $X_1, \ldots, X_n$ as leaves. The parameters of the network are the weights w_{ij} associated to sum nodes children edges.

Product nodes define factorizations over independent components, while sum nodes represent mixtures.
Products over nodes with different scopes (*decomposability*) and sums over nodes with same scopes (*completeness*) guarantee the exact and efficient computation of several inference queries (*validity*).





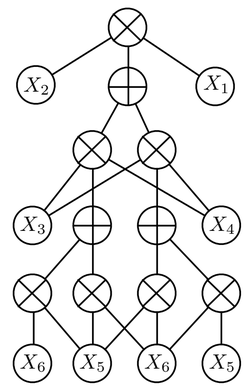
Bottom-up evaluation of the network:

$$S_{X_i}(x_j) = P(X_i = x_j)$$

$$S_{\bigoplus}(\mathbf{x}) = \sum_{i \in ch(\bigoplus)} w_i S_i(\mathbf{x}) \quad S_{\bigotimes}(\mathbf{x}) = \prod_{i \in ch(\bigotimes)} S_i(\mathbf{x})$$

Inferences linear in the ***size of the network*** (*# edges*):

- $Z = S(*)$
- $P(\mathbf{e}) = S(\mathbf{e})/S(*)$
- $P(\mathbf{q}|\mathbf{e}) = \frac{P(\mathbf{q},\mathbf{e})}{P(\mathbf{e})} = \frac{S(\mathbf{q},\mathbf{e})}{S(\mathbf{e})}$
- $MPE(\mathbf{q},\mathbf{e}) = \max_{\mathbf{q}} P(\mathbf{q},\mathbf{e}) = M(\mathbf{e})$ (approximate, by turning S into a Max-Product Network M)

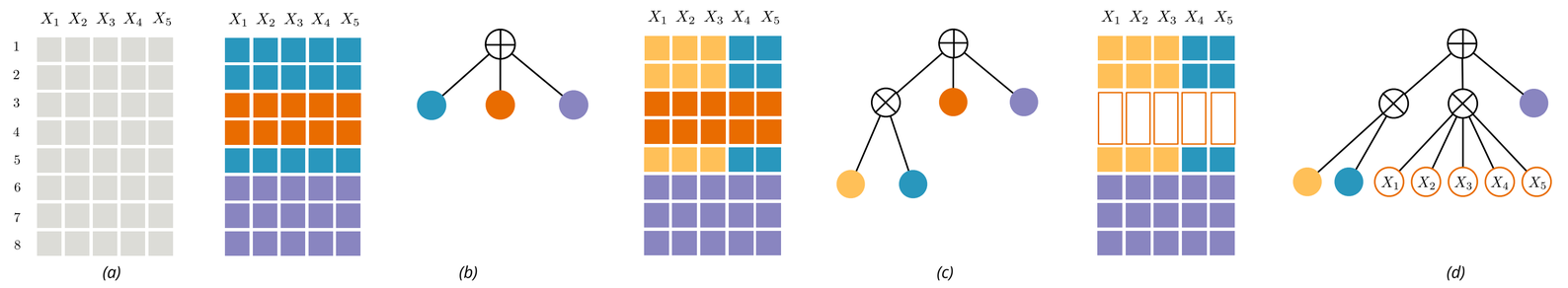


SPNs achieved impressive performances in several application domains such as NLP, robot control, speech recognition and computer vision. For instance, SPNs outperformed SOTA models and deep architectures on image completion task [5] (in red in the figure on the right).

We focus on ***learning the structure*** of SPNs from data as it enables the aforementioned achievements by freeing the user from crafting a network by hand.

Structure learning and variable splitting

SPN structure learning is a constraint-based search. Main ideas: to discover hidden variables for sum nodes and independencies for product nodes by applying some form of clustering along matrix axis. ***LearnSPN*** *[1]* is the first principled *top-down greedy algorithm*.



LearnSPN builds a tree-like SPN by recursively splitting the data matrix: finding sets of independent variables (columns) by performing ***Greedy Variable Splitting*** (**GVS**) (Figure 1.c); splitting instances in $|C|$ groups by performing ***sample clustering***, e.g., by Expectation Maximization (Figure 1.b). Sum node weights are estimated as the cluster proportions. If there are less than μ instances, it puts a ***naive factorization*** over leaves (Figure 1.d). For each univariate distribution it gets its ***ML estimation*** smoothed by α.

Structure quality matters as much as likelihood: tractability is guaranteed if the network size is polynomial in # variables, and the ***depth of the network*** (*# layers*) determines expressive efficiency [4].

When GVS tries to split variables into independent groups, it builds a ***dependency graph*** by exploiting a ***G-test*** [7] with statistical significance threshold ρ:

$$\text{GTest}(\mathcal{D}, X_i, X_j) = 2 \sum_{x_i \sim X_i} \sum_{x_j \sim X_j} c(x_i, x_j) \cdot \log \frac{c(x_i,x_j) \cdot |\mathcal{D}|}{c(x_i)c(x_j)}$$

GVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)

1: **Input:** set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; ρ: independence threshold
2: **Output:** a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$,
3: $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}$, $X_0 \leftarrow$ drawVariableAtRandom($\boldsymbol{X}$)
4: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{X_0\}$, $\boldsymbol{P} \leftarrow \{X_0\}$, $\boldsymbol{X}_{d_1} \leftarrow \{X_0\}$, $\boldsymbol{R} \leftarrow \emptyset$
5: **repeat**
6: $X_p \leftarrow$ pop($\boldsymbol{P}$) ▷ $\boldsymbol{P} \leftarrow \boldsymbol{P} \setminus \{X_p\}$
7: **for each** $X_k \in \mathbf{X}$ **do**
8: **if** not GTest($\mathcal{D}, X_p, X_k$) $< \rho$ **then**
9: $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{X_k\}$, $\boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup \{X_k\}$, $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{X_k\}$
10: **for each** $X_j \in \boldsymbol{R}$ **do**
11: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{X_j\}$
12: **until** $|\boldsymbol{P}| > 0 \wedge |\boldsymbol{X}| > 0$
13: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2} \setminus \boldsymbol{X}_{d_1}\}$

Issue: Variable splitting step via GVS scales ***quadratically*** on # variables: a bottleneck for structure learning!
Idea: We conceive alternative faster approximate variable splitting methods with ***sub-quadratic*** complexity.
One method employs ***random subspaces and stochastic decisions*** when splitting, while the other is based on the concept of ***entropy*** from information theory [3] i.e. by grouping variables with *low entropy* (independent from the rest).

Stochastic variable splitting method

Random Greedy Variable Splitting (**RGVS**) finds two sets of independent variables, by first applying GVS on a randomly selected subset of the variables. Then, it stochastically agglomerates the remaining ones.

RGVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)

1: **Input:** set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; ρ: a statistical independence threshold
2: **Output:** a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$
3: $\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$, $n \leftarrow |\boldsymbol{X}|$, $k \leftarrow max(\lfloor \sqrt{n} \rfloor, 2)$
4: **if** $k = n$ **then**
5: **return** GVS($\mathcal{D}, \boldsymbol{X}, \rho$)
6: $\boldsymbol{R} \leftarrow$ randomSubspace($\boldsymbol{X}, k$)
7: $\boldsymbol{S} \leftarrow \boldsymbol{X} \setminus \boldsymbol{R}$
8: $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}, \boldsymbol{R}, \rho$)
9: **if** $\boldsymbol{X}_{d_2} = \emptyset$ **then**
10: **return** $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \emptyset\}$
11: $r \leftarrow$ Bernoulli(0.5)
12: **if** $r = 0$ **then**
13: **return** $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \boldsymbol{X}_{d_2}\}$
14: **else**
15: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2} \cup \boldsymbol{S}\}$

We randomly select only $\sqrt{n}$ variables ($n = |\boldsymbol{X}|$), hence RGVS only takes **linear time** w.r.t. # of variables!

Entropy-based variable splitting method

Entropy Based Variable Splitting (**EBVS**) first performs a ML estimation of the entropy of each variable (with smoothing parameter α). Then, it groups those with entropies lower than a threshold η, hence achieving ***linear complexity***!

EBVS($\mathcal{D}$, $\boldsymbol{X}$, η, α)

1: **Input:** set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; η: a threshold for entropies, α: Laplacian smoothing parameter
2: **Output:** a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$
3: $\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$
4: **for each** $X_i \in \mathbf{X}$ **do**
5: **if** computeEntropy($\mathcal{D}, X_i, \alpha$) $< \eta$ **then**
6: $\boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup \{X_i\}$
7: **else**
8: $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}_{d_2} \cup \{X_i\}$
9: **if** $|\boldsymbol{X}_{d_1}| = 0$ **then**
10: **return** $\{\boldsymbol{X}_{d_2}, \boldsymbol{X}_{d_1}\}$
11: **else**
12: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

Since entropy tends to decrease as the # instances in a data slice gets smaller, we devised **EBVS-AE**: a variant of EBVS that ***automatically adapts the threshold*** η w.r.t. the # instances in the data slice $\mathcal{D}$.

Experiments

We empirically evaluated the proposed approximate variants—EBVS, EBVS-AE, RGVS—against GVS on ***9 standard benchmark*** datasets for probabilistic model structure learning [2]. We employed LearnSPN-b [6], a variant of LearnSPN that limits the row cluster number to two, slowing down the greedy learning process and building deeper networks.
We compared the learned SPNs under their achieved accuracies, learning times and their structure quality. We measured:

- ***average log-likelihood*** on predicting test instances
- ***learning time*** in seconds
- network ***size*** (# edges)
- network ***depth*** (# alternated type layers)
- network ***weights*** (# network parameters)

Model selection is performed via a *grid search* taking the best validation models. For GVS and RGVS we used the same parameter space. For EBVS and EBVS-AE we included hyperparameter η instead of ρ:

- $\rho \in \{5, 10, 15, 20\}$
- $\eta \in \{0.05, 0.1, 0.3, 0.5\}$
- $\mu \in \{10, 50, 100, 500\}$
- $\alpha \in \{0.1, 0.2, 1.0, 2.0\}$

	learning time				log-likelihood			
	EBVS	EBVS-AE	RGVS	GVS	EBVS	EBVS-AE	RGVS	GVS
NLTCS	98	31	**4**	8	**-6.051**	**-6.046**	-6.329	**-6.040**
Plants	255	179	**24**	70	-12.890	**-12.853**	-16.633	**-12.880**
Audio	130	108	**25**	59	-40.763	**-40.632**	-41.983	**-40.697**
Jester	73	67	**16**	41	-53.897	**-53.528**	-54.881	-53.919
Netflix	131	123	**48**	89	-58.234	**-58.021**	-59.752	-58.4360
Accidents	623	96	**18**	34	-35.336	-35.635	-39.370	**-29.002**
Retail	105	97	**11**	22	-11.245	-11.198	-11.290	**-10.969**
Pumsb-star	256	245	**18**	34	-29.235	-29.485	-41.969	**-23.282**
DNA	17	18	**3**	9	-97.876	-97.764	-99.123	**-81.931**

	# edges				# layers				# params			
	EBVS	EBVS-AE	RGVS	GVS	EBVS	EBVS-AE	RGVS	GVS	EBVS	EBVS-AE	RGVS	GVS
NLTCS	8185	3969	**316**	1129	23	9	9	17	1190	331	58	271
Plants	42318	67821	**2770**	15129	27	17	15	27	2304	1863	415	2635
Audio	36499	43243	**2581**	17811	21	11	11	25	550	477	308	2736
Jester	26263	27609	**1862**	12460	17	5	13	25	308	276	270	2071
Netflix	42033	44512	**5097**	30417	21	11	15	31	503	465	737	4351
Accidents	98218	33377	**1529**	11861	33	15	15	27	4315	436	240	2656
Retail	10096	6973	**368**	1010	41	37	7	19	446	258	31	175
Pumsb-star	32776	137092	**1478**	12821	29	19	13	27	1941	1885	216	2679
DNA	8694	8694	**475**	3384	7	7	9	13	52	52	38	938

From our results it turns out that there is ***no free lunch***:

- EBVS-AE scores comparable or better accuracy than GVS (bold values are statistically better than all the others according to a Wilcoxon signed rank test with p-value of 0.05) but it favors bigger networks. All in all, even if it moves faster in the search space, it requires more time than GVS to build a whole network
- EBVS-AE performs better—log-likelihood-wise—than EBVS and should be preferred over it if one looks for more accurate models
- RGVS is much faster than other methods during learning and it is able to build very compact networks thus shortening inference times, however, its accuracy dramatically degrades on some datasets

References

[1] Robert Gens and Pedro Domingos. "Learning the Structure of Sum-Product Networks". In: *ICML*. 2013, pp. 873–880.

[2] Jan Van Haaren and Jesse Davis. "Markov Network Structure Learning: A Randomized Feature Generation Approach". In: *AAAI*. AAAI Press, 2012.

[3] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press, 2002. ISBN: 0521642981.

[4] James Martens and Venkatesh Medabalimi. "On the Expressive Efficiency of Sum Product Networks". In: *CoRR* abs/1411.7717 (2014).

[5] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: a New Deep Architecture". In: *UAI 2011* (2011).

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

[7] B. Woolf. "The log likelihood ratio test". In: *Annals of Human Genetics 21* (1957).

https://github.com/fabriziov/alt-vs-spyn