Learning Probabilistic Circuits

from parameter to structure learning

antonio vergari (he/him)

@tetraduzione

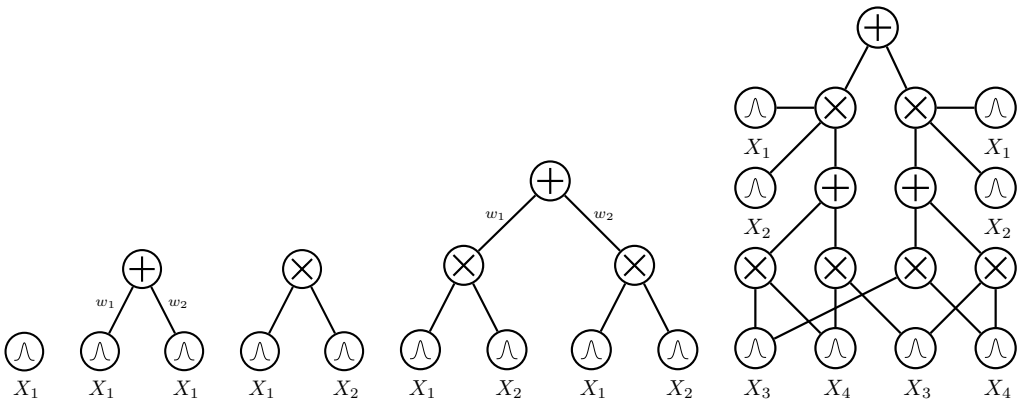
8th Mar 2024 - **Advanced Probabilistic Modeling** - *University of Trento*

in the previous episodes...

#1 circuits: representation & inference

Probabilistic Circuits (PCs)

A grammar for tractable computational graphs



Low-treewidh PGMs

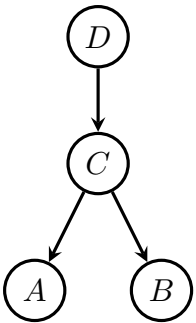
Tree, polytrees and Thin Junction trees can be turned into

- decomposable
- smooth
- deterministic

circuits

Therefore they support tractable

- EVI
- MAR/CON
- MAP



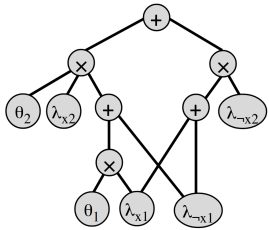
Arithmetic Circuits (ACs)

ACs *[Darwiche 2003]* are

- decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP



⇒ *parameters are attached to the leaves*

⇒ *...but can be moved to the sum node edges [Rooshenas and Lowd 2014]*

Lowd and Rooshenas, "Learning Markov Networks With Arithmetic Circuits", , 2013

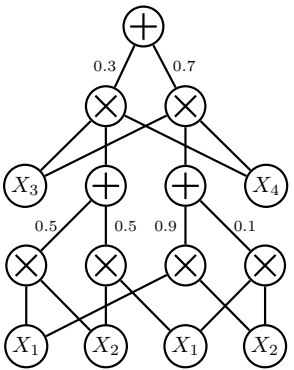
Sum-Product Networks (SPNs)

SPNs *[Poon and Domingos 2011]* are

- decomposable
- smooth
- ~~deterministic~~

They support tractable

- EVI
- MAR/CON
- ~~MAP~~



$\Rightarrow$ *deterministic SPNs are also called selective [Peharz, Gens, and Domingos 2014]*

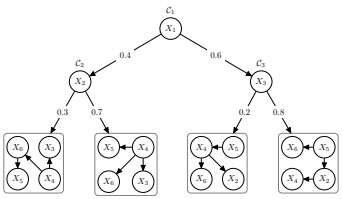
Cutset Networks (CNets)

CNets *[Rahman, Kothalkar, and Gogate 2014]* are

- decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP



Rahman, Kothalkar, and Gogate, "Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees", , 2014

Di Mauro, Vergari, and Esposito, "Learning Accurate Cutset Networks by Exploiting Decomposability", , 2015

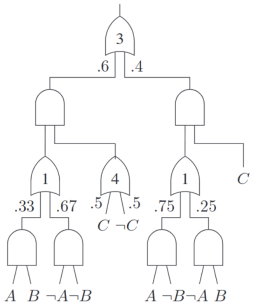
Probabilistic Sentential Decision Diagrams

PSDDs *[Kisa et al. 2014a]* are

- structured decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP
- Complex queries!



Kisa et al., "Probabilistic sentential decision diagrams", , 2014

Choi, Van den Broeck, and Darwiche, "Tractable learning for structured probability spaces: A case study in learning preference distributions", IJCAI, 2015

Shen, Choi, and Darwiche, "Conditional PSDDs: Modeling and learning with modular knowledge", , 2018

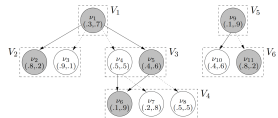
Probabilistic Decision Graphs

PDGs *[Jaeger 2004]* are

- structured decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP
- Complex queries!



Jaeger, "Probabilistic decision graphs—combining verification and AI techniques for probabilistic inference", International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 2004

jaeger2006learning, jaeger2006learning, jaeger2006learning, jaeger2006learning

AndOrGraphs

AndOrGarphs

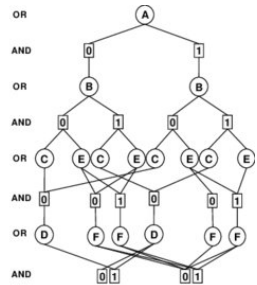
[Dechter and Mateescu 2007]

are

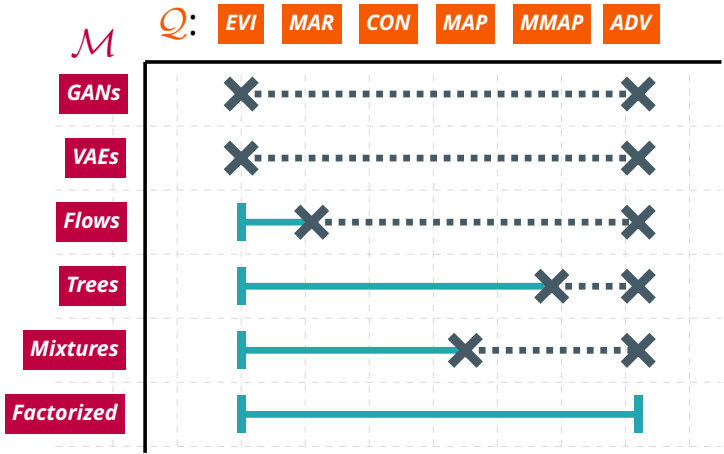
- structured
- decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP
- Complex queries!



Dechter and Mateescu, "AND/OR search spaces for graphical models", Artificial intelligence, 2007
Marinescu and Dechter, "Best-first AND/OR search for 0/1 integer programming", , 2007



tractable bands

Learning Probabilistic Circuits

Learning probabilistic circuits

Probabilistic circuits are (peculiar) neural networks… ***just backprop with SGD!***

Learning probabilistic circuits

Probabilistic circuits are (peculiar) neural networks… ***just backprop with SGD!***

…end of Learning section!

Learning probabilistic circuits

Probabilistic circuits are (peculiar) neural networks... ***just backprop with SGD!***

wait but...

SGD is slow to converge...can we do better?

How to learn normalized weights?

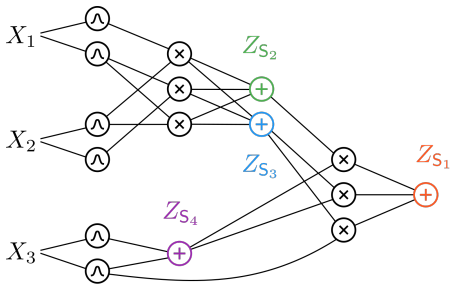
Can we exploit structural properties somehow?

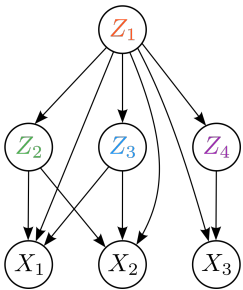
Learning probabilistic circuits

	Parameters	*Structure*
Generative	?	?
Discriminative	?	?

Maximum likelihood (frequentist)

PCs can be interpreted as ***hierarchical latent variable models***, where each sum node corresponds to a discrete latent variable *[Peharz et al. 2016].* This allows to perform **classical maximum-likelihood** estimation.





Closed-form maximum likelihood

When the circuit is **deterministic**, there is even an ***closed-form ML solution***, which is incredible fast:

```
julia> using ProbabilisticCircuits;
julia> data, structure = load(...);
julia> num_examples(data)
17412
julia> num_edges(structure)
270448
julia> @btime estimate_parameters(structure, data);
  63.585 ms (1182350 allocations: 65.97 MiB)
```

Custom SIMD and CUDA kernels to parallelize over layers and training examples.

https://github.com/Juice-jl/

Expectation-Maximization

When the PC is not deterministic, we can still apply ***expectation-maximization*** *[Peharz et al. 2016].* EM can piggy-back on autodiff:

```
train_x, valid_x, test_x = get_mnist_images([7])

graph = Graph.poon_domingos_structure(shape=(28,28), delta=[7])
args = EinsumNetwork.Args(num_var=train_x.shape[1], num_dims=1,
                          num_classes=1, num_sums=28,
                          num_input_distributions=28,
                          exponential_family=EinsumNetwork.BinomialArray,
                          exponential_family_args={'N':255},
                          online_em_frequency=1, online_em_stepsize=0.05)

PC = EinsumNetwork.EinsumNetwork(graph, args)
PC.initialize()
PC.to('cuda')
```

https://github.com/cambridge-mlg/EinsumNetworks

Expectation-Maximization

```
for epoch_count in range(10):
    train_ll, valid_ll, test_ll = compute_loglikelihood()
    start_t = time.time()

    for idx in get_batches(train_x, 100):
        outputs = PC.forward(train_x[idx, :])
        log_likelihood = EinsumNetwork.log_likelihoods(outputs).sum()
        log_likelihood.backward()
        PC.em_process_batch()

    print_performance(epoch_count, train_ll, valid_ll, test_ll, time.time() - start_t)
```

https://github.com/cambridge-mlg/EinsumNetworks

Expectation-Maximization

```
# train sample: 5175
# parameters: 1573486

[epoch 0]   train LL -140936.80   valid LL -140955.72   test LL -141033.80   ... elapsed time 3.621 sec
[epoch 1]   train LL  -15916.14   valid LL  -15693.25   test LL  -15976.43   ... elapsed time 3.438 sec
[epoch 2]   train LL  -10865.67   valid LL  -10616.72   test LL  -10943.56   ... elapsed time 3.436 sec
[epoch 3]   train LL  -10388.53   valid LL  -10158.84   test LL  -10475.49   ... elapsed time 3.473 sec
[epoch 4]   train LL  -10264.11   valid LL  -10041.66   test LL  -10352.59   ... elapsed time 3.497 sec
[epoch 5]   train LL  -10212.66   valid LL  -10001.09   test LL  -10319.35   ... elapsed time 3.584 sec
[epoch 6]   train LL  -10192.21   valid LL   -9965.98   test LL  -10314.84   ... elapsed time 3.508 sec
[epoch 7]   train LL  -10153.97   valid LL   -9920.09   test LL  -10261.41   ... elapsed time 3.446 sec
[epoch 8]   train LL  -10112.95   valid LL   -9882.48   test LL  -10236.34   ... elapsed time 3.579 sec
[epoch 9]   train LL  -10093.31   valid LL   -9862.15   test LL  -10200.94   ... elapsed time 3.483 sec
```

Peharz et al., "Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits", , 2020

Learning input distributions

As simple as tossing a coin

The simplest PC: a single input distribution p_{L} with parameters $\boldsymbol{\theta}$

$\Longrightarrow$ *maximum likelihood (ML) estimation over data* $\mathcal{D}$

Learning input distributions

As simple as tossing a coin

The simplest PC: a single input distribution p_{L} with parameters $\boldsymbol{\theta}$

$\Rightarrow$ *maximum likelihood (ML) estimation over data* $\mathcal{D}$

E.g. Bernoulli with parameter θ

$$\hat{\theta}_{\mathsf{ML}} = \frac{\sum_{x \in \mathcal{D}} \mathbb{1}[x = 1] + \alpha}{|\mathcal{D}| + 2\alpha} \quad \Rightarrow \text{ Laplace smoothing}$$

Learning probabilistic circuits

	Parameters	Structure
Generative	**deterministic** closed-form MLE *[Kisa et al. 2014b; Peharz, Gens, and Domingos 2014]* **non-deterministic** EM *[Poon and Domingos 2011; Peharz 2015; Zhao, Poupart, and Gordon 2016]* SGD *[Sharir et al. 2016; Peharz et al. 2019b]* Bayesian *[Jaini et al. 2016; Rashwan, Zhao, and Poupart 2016]* *[Zhao et al. 2016; Trapp et al. 2019; Vergari et al. 2019]*	?
Discriminative	?	?

Learning input distributions

General case: still simple

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are **exponential families** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

Learning input distributions

General case: still simple

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are **exponential families** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

Where:

- $A(\boldsymbol{\theta})$: log-normalizer
- $\boldsymbol{h}(\mathbf{x})$ base-measure
- $\boldsymbol{T}(\mathbf{x})$ sufficient statistics
- $\boldsymbol{\theta}$ natural parameters

Learning input distributions

General case: still simple

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are **exponential families** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

Where:

- $A(\boldsymbol{\theta})$: log-normalizer
- $\boldsymbol{h}(\mathbf{x})$ base-measure
- $\boldsymbol{T}(\mathbf{x})$ sufficient statistics
- $\boldsymbol{\theta}$ natural parameters
- or $\boldsymbol{\phi}$ expectation parameters — 1:1 mapping with $\boldsymbol{\theta} \Longrightarrow \boldsymbol{\theta} = \boldsymbol{\theta}(\boldsymbol{\phi})$

Learning input distributions

General case: still simple

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are **exponential families** of the form:

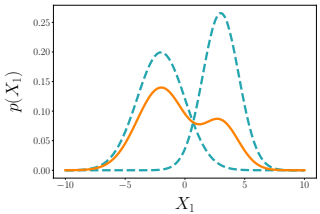
$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

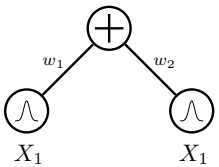
Maximum likelihood estimation is still "**counting**":

$$\hat{\boldsymbol{\phi}}_{\mathsf{ML}} = \mathbb{E}_{\mathcal{D}}[\boldsymbol{T}(\mathbf{x})] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}}\boldsymbol{T}(\mathbf{x})$$

$$\hat{\boldsymbol{\theta}}_{\mathsf{ML}} = \boldsymbol{\theta}(\hat{\boldsymbol{\phi}}_{\mathsf{ML}})$$

The simplest “real” PC: a sum node

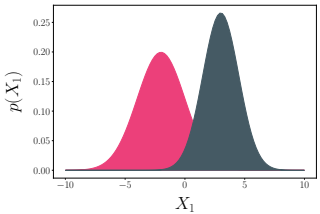


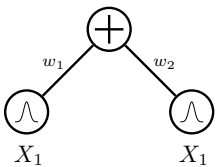


Recall that sum nodes represent **mixture models**:

$$p_{\mathsf{S}}(\mathbf{x}) = \sum_{k=1}^{K} w_k p_{\mathsf{L}_k}(\mathbf{x})$$

The simplest “real” PC: a sum node





Recall that sum nodes represent **latent variable models**:

$$p_{\mathsf{S}}(\mathbf{x}) = \sum_{k=1}^{K} p(Z = k) p(\mathbf{x} \mid Z = k)$$

Expectation-Maximization for mixtures

ML if Z was observed:

$$\hat{w}_k = \frac{\sum_{z \in \mathcal{D}} \mathbb{1}[z = k]}{|\mathcal{D}|} \qquad \hat{\boldsymbol{\phi}}_k = \frac{\sum_{\mathbf{x}, z \in \mathcal{D}} \mathbb{1}[z = k] T(\mathbf{x})}{\sum_{z \in \mathcal{D}} \mathbb{1}[z = k]}$$

Z is unobserved—but we have $p(Z = k \,|\, \mathbf{x}) \propto w_k \, \mathsf{L}_k(\mathbf{x})$.

$$w_k^{new} = \frac{\sum_{\mathbf{x} \in \mathcal{D}} p(Z = k \,|\, \mathbf{x})}{|\mathcal{D}|} \qquad \boldsymbol{\phi}_k^{new} = \frac{\sum_{\mathbf{x}, z \in \mathcal{D}} p(Z = k \,|\, \mathbf{x}) T(\mathbf{x})}{\sum_{z \in \mathcal{D}} p(Z = k \,|\, \mathbf{x})}$$

Expectation-Maximization for PCs

EM for mixtures well understood.

Mixtures are PCs with 1 sum node.

The general case, PCs with many sum nodes, is similar ...

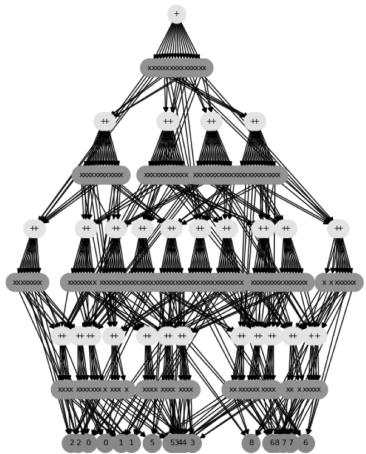
Expectation-Maximization for PCs

EM for mixtures well understood.

Mixtures are PCs with 1 sum node.

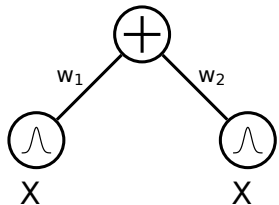
The general case, PCs with many sum nodes, is similar ...

...but a bit more complicated.



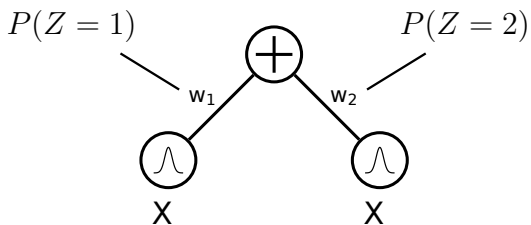
Expectation-Maximization for PCs

[Peharz et al. 2016]



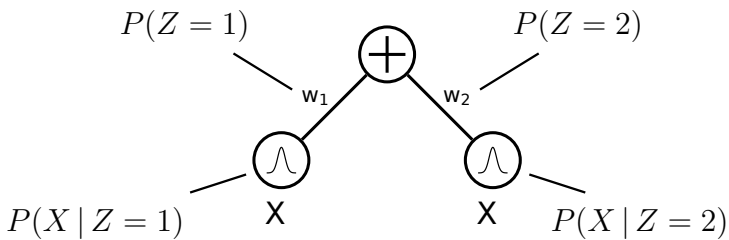
Expectation-Maximization for PCs

[Peharz et al. 2016]



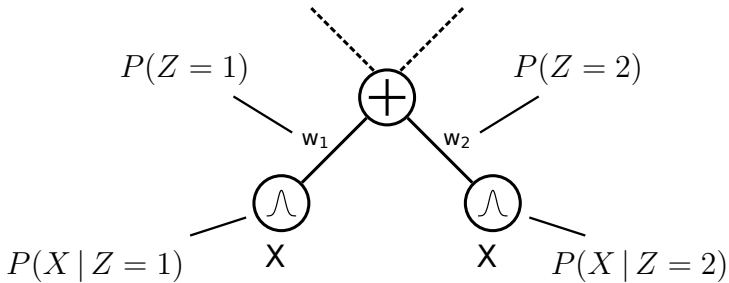
Expectation-Maximization for PCs

[Peharz et al. 2016]



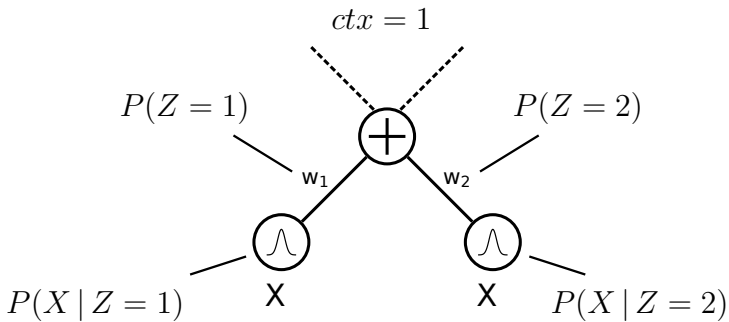
Expectation-Maximization for PCs

[Peharz et al. 2016]



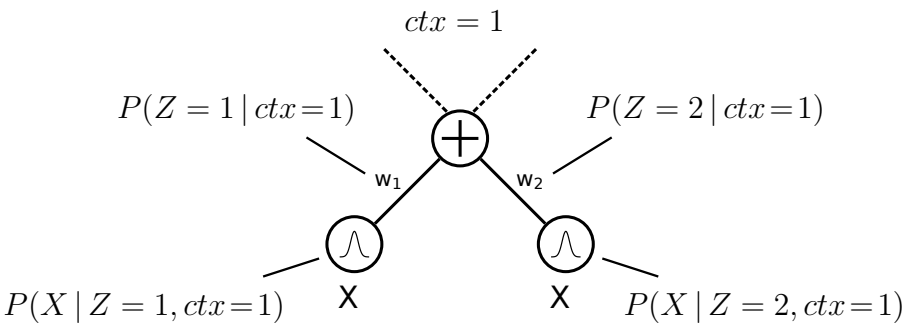
Expectation-Maximization for PCs

[Peharz et al. 2016]



Expectation-Maximization for PCs

[Peharz et al. 2016]



Expectation-Maximization

Tractable MAR (smooth, decomposable)

$$w_{i,j}^{new} \leftarrow \frac{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}]}{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1 \mid \mathbf{x}; \mathbf{w}^{old}]}$$

Darwiche, "A Differential Approach to Inference in Bayesian Networks", J.ACM, 2003
Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016

Expectation-Maximization

Tractable MAR (smooth, decomposable)

$$w_{i,j}^{new} \leftarrow \frac{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}]}{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1 \mid \mathbf{x}; \mathbf{w}^{old}]}$$

We get **all** the required statistics with a single backprop pass:

$$p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}] = \frac{1}{p(\mathbf{x})} \frac{\partial p(\mathbf{x})}{\partial \mathsf{S}_i(\mathbf{x})} \mathsf{N}_j(\mathbf{x}) w_{i,j}^{old}$$

Darwiche, "A Differential Approach to Inference in Bayesian Networks", J.ACM, 2003
Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016

Bayesian parameter learning

Formulate a prior $p(\mathbf{w}, \boldsymbol{\theta})$ over sum-weights and parameters of input units. Then perform posterior inference:

$$p(\mathbf{w}, \boldsymbol{\theta}|\mathcal{D}) \propto p(\mathbf{w}, \boldsymbol{\theta})\, p(\mathcal{D}|\mathbf{w}, \boldsymbol{\theta})$$

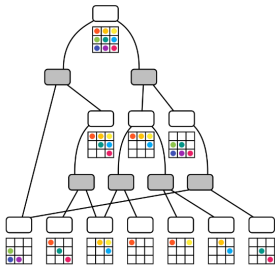
Moment matching (oBMM) *[Jaini et al. 2016; Rashwan, Zhao, and Poupart 2016]*

Collapsed variational inference algorithm *[Zhao et al. 2016]*

Gibbs sampling *[Trapp et al. 2019; Vergari et al. 2019]*

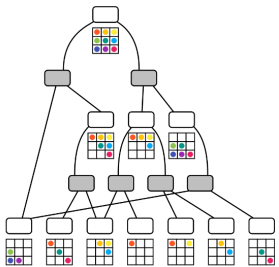
Structure Learning

Learning recipe

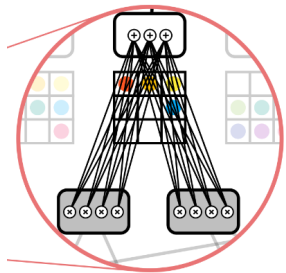


1) Build a ***region graph***

Learning recipe

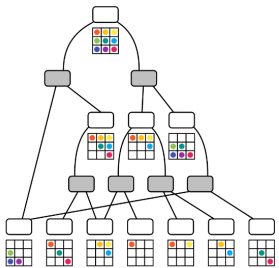


1) Build a ***region graph***

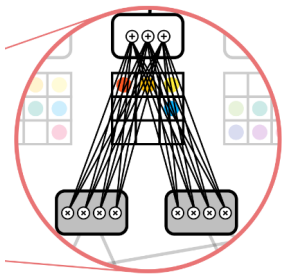


2) Overparameterize

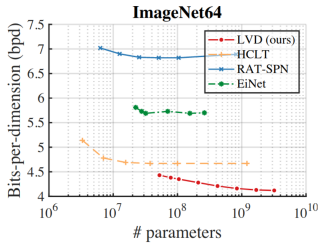
Learning recipe



1) Build a ***region graph***

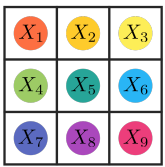


2) Overparameterize



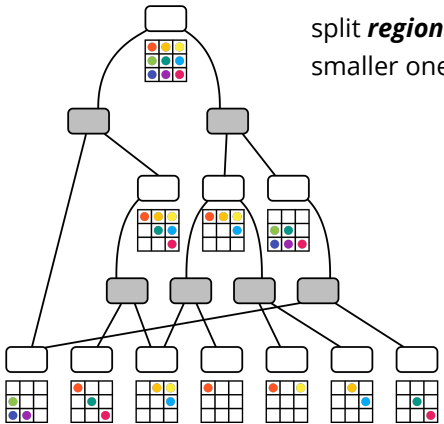
3) Learn parameters

Region graphs

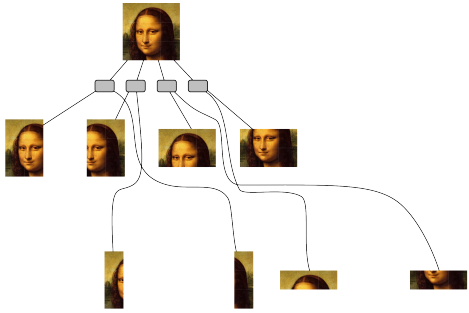


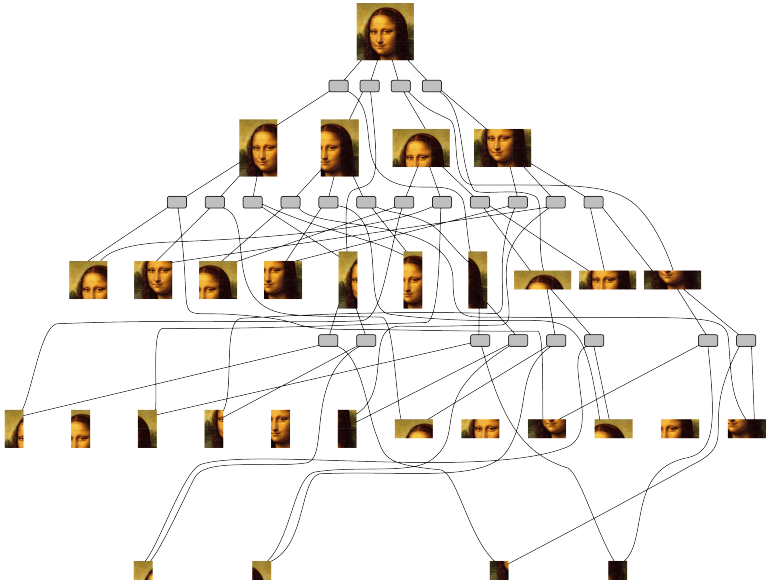
$\mathcal{R}$

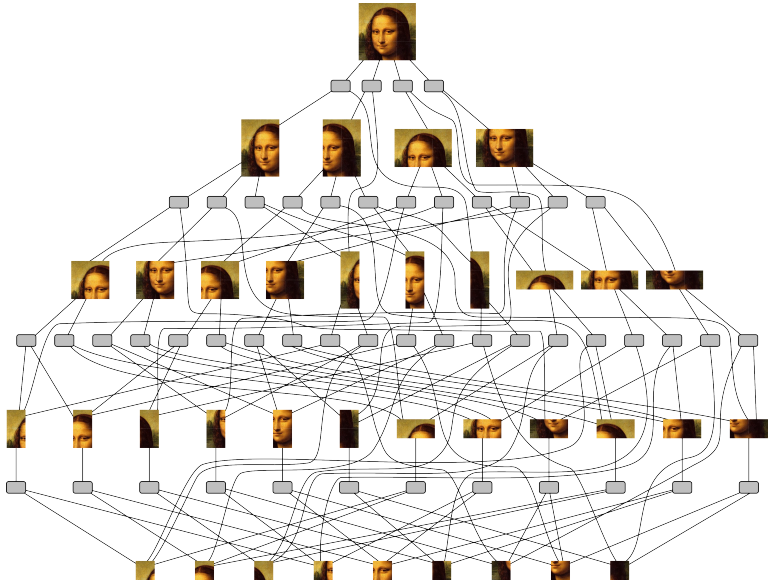
$\mathcal{P}$



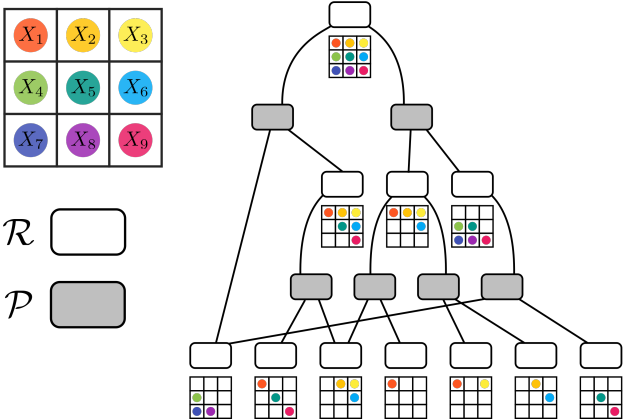
split ***regions*** (sets of variables $\mathcal{R}$) into smaller ones via ***partitions*** ($\mathcal{P}$)







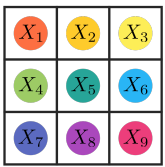
From region graphs to PCs



From region graphs to PCs

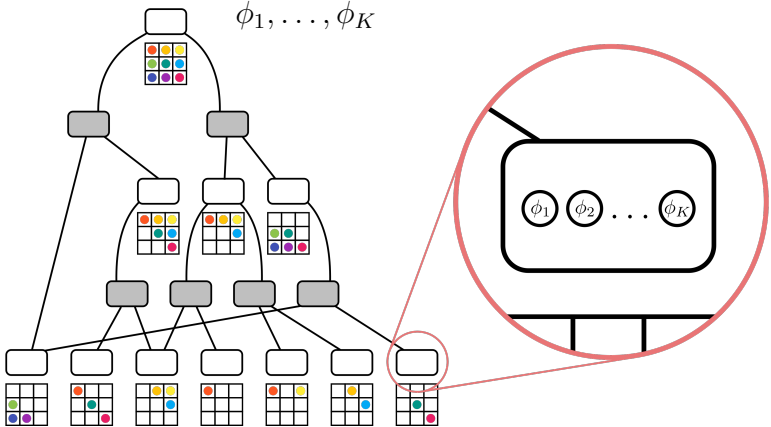
Equip each input region with non-linear units

$\phi_1, \ldots, \phi_K$



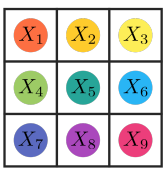
$\mathcal{R}$

$\mathcal{P}$



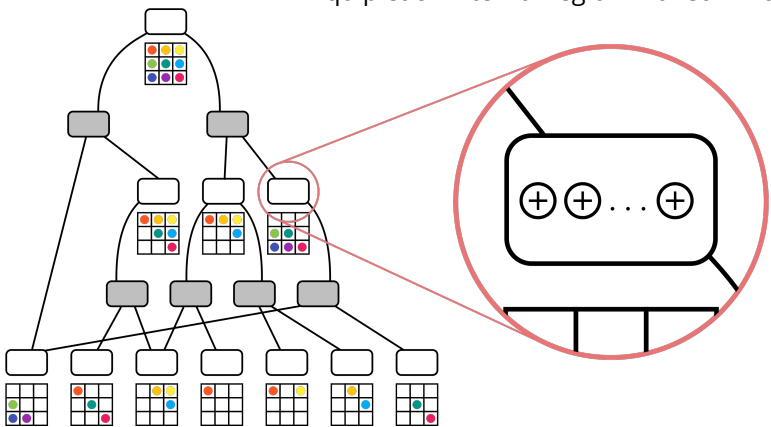
From region graphs to PCs

Equip each internal region with sum nodes

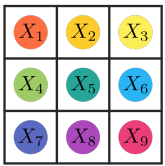


$\mathcal{R}$

$\mathcal{P}$

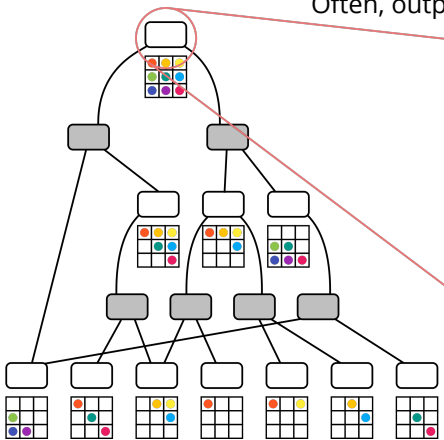


From region graphs to PCs

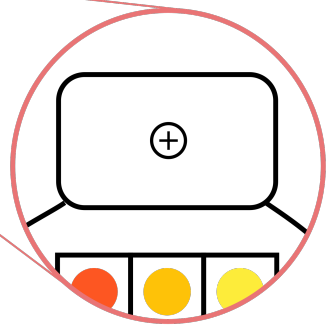


$\mathcal{R}$

$\mathcal{P}$

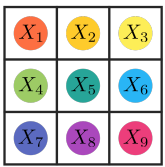


Often, output region has only a single sum



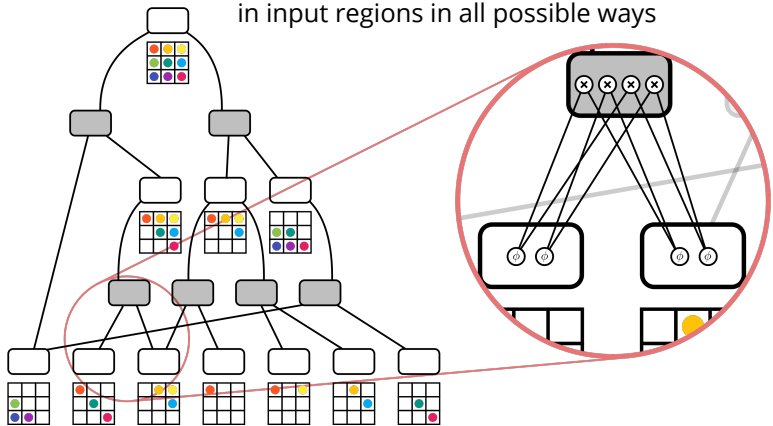
From region graphs to PCs

Equip partitions with products, combining units in input regions in all possible ways



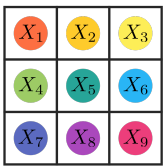
$\mathcal{R}$

$\mathcal{P}$



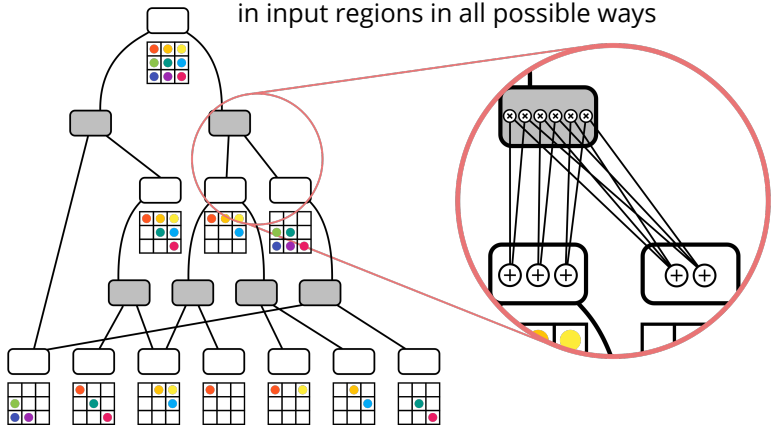
From region graphs to PCs

Equip partitions with products, combining units in input regions in all possible ways

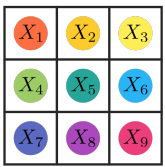


$\mathcal{R}$

$\mathcal{P}$



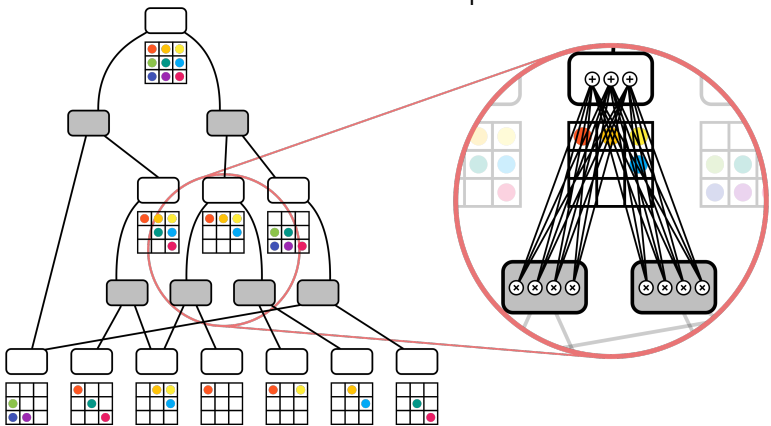
From region graphs to PCs



$\mathcal{R}$

$\mathcal{P}$

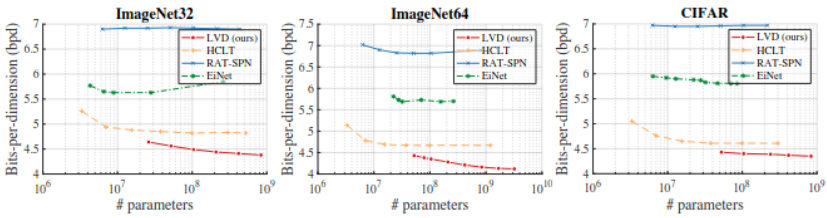
Connect products to sum units above



Scaling up image models

Latent Variable Distillation

Dataset	TPMs				DGMs		
	LVD (ours)	HCLT	EiNet	RAT-SPN	Glow	RealNVP	BIVA
ImageNet32	**4.38**	4.82	5.63	6.90	4.09	4.28	3.96
ImageNet64	**4.12**	4.67	5.69	6.82	3.81	3.98	-
CIFAR	**4.37**	4.61	5.81	6.95	3.35	3.49	3.08



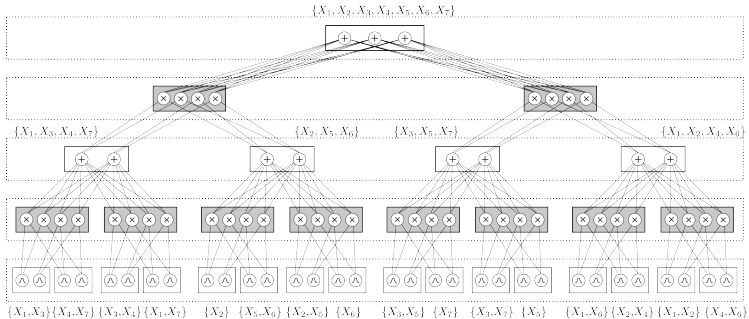
Liu, Zhang, and Broeck, "Scaling Up Probabilistic Circuits by Latent Variable Distillation", arXiv preprint, 2022

How to construct and learn RGs?

Random regions graphs

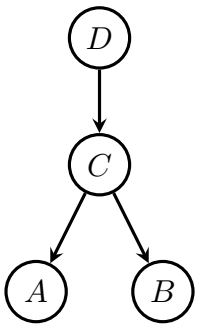
The "no-learning" option *[Peharz et al. 2019a]*

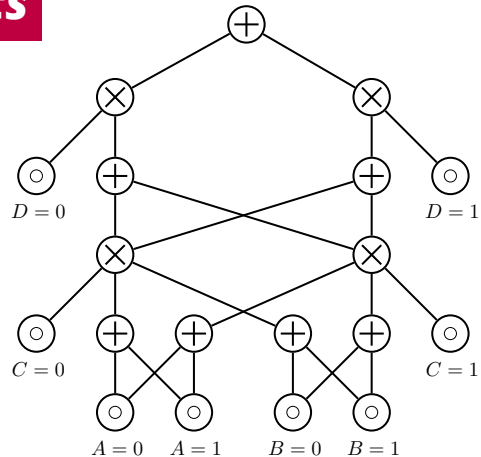
Generating a random region graph, by recursively splitting $\mathbf{X}$ into two random parts:



from PGMs to circuits

via compilation

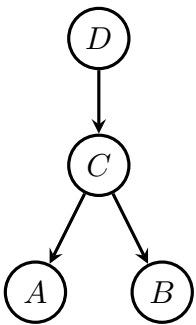




from PGMs to circuits

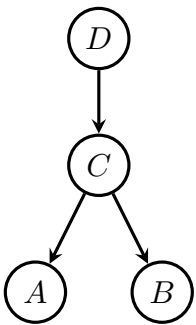
via compilation

Bottom-up ***compilation***: starting from leaves...



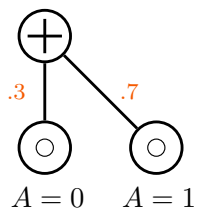
from PGMs to circuits

via compilation



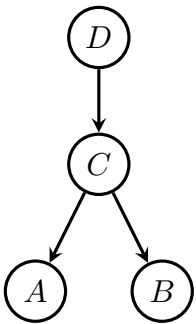
...compile a leaf CPT

$p(A|C=0)$

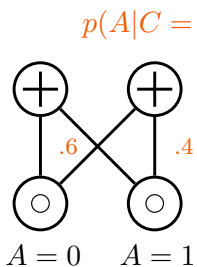


from PGMs to circuits

via compilation

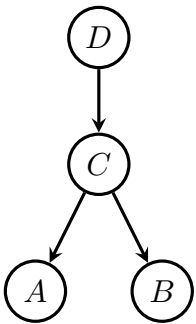


...compile a leaf CPT

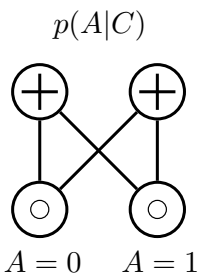


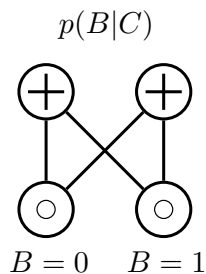
from PGMs to circuits

via compilation



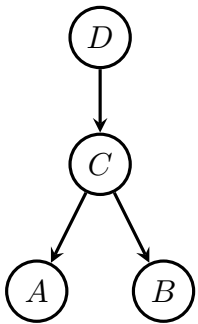
...compile a leaf CPT...for all leaves...



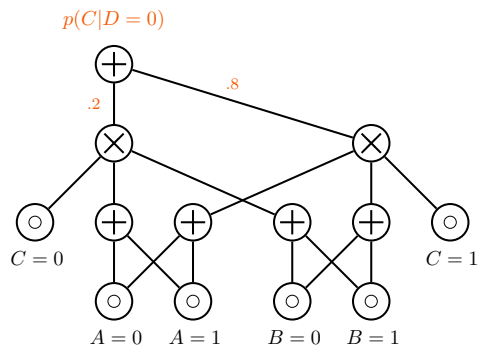


from PGMs to circuits

via compilation

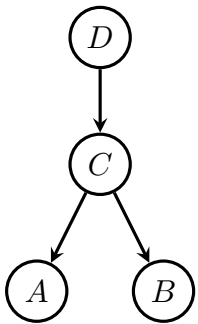


...and recurse over parents...

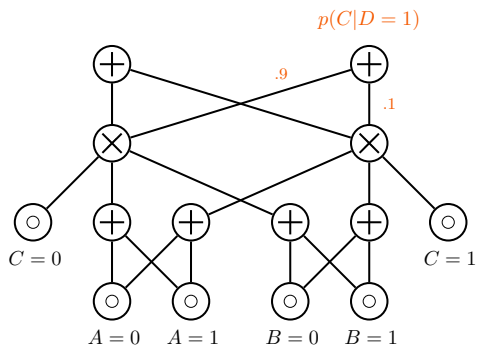


from PGMs to circuits

via compilation

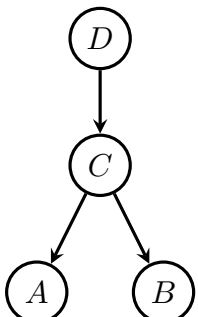


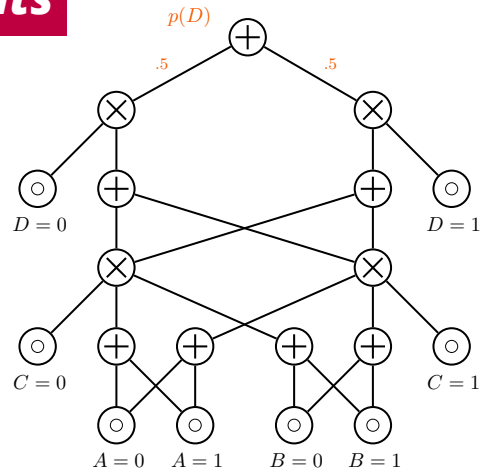
...while reusing previously compiled nodes!...



from PGMs to circuits

via compilation





Compilation

Selected references

Logic circuits, interplay between structural properties and tractable reasoning

[Darwiche and Marquis 2002]

Converting probabilistic graphical models via knowledge compilation

*[**darwiche2002logical**]*

Logic circuit compilers

*[**darwiche2004new**; **lagniez2017improved**; **muise2012d**; **bova2015compiling**; **oztok2018exhaustive**]*

Neuro-symbolic models using logic circuits

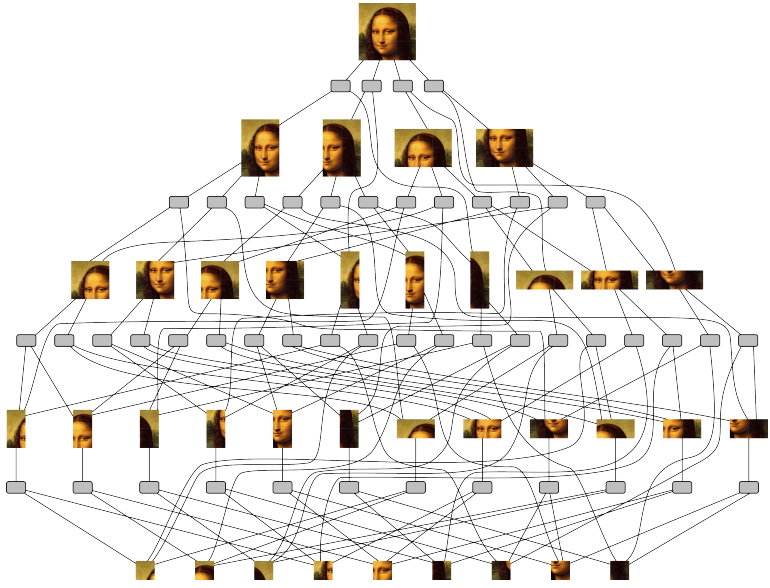
*[**Ahmed2022**; Ahmed et al. 2022]*

Image-tailored circuit structure

"Recursive image slicing" *[Poon and Domingos 2011]*

Images yield a natural region graph by using *axis-aligned splits*:

- Start with the full image (=output region)
- Define partitions by applying *horizontal* and *vertical* splits
- Recurse on the newly generated sub-images (internal regions)
- Structure somewhat reminiscent to convolutions
- Generates RGs which are "true DAGs," i.e. regions get re-used



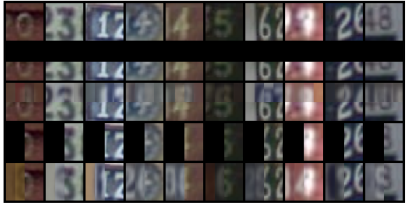
Generative modeling

Inpainting

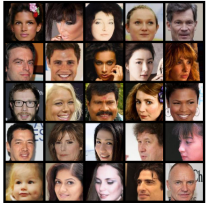
[Peharz et al. 2020]

(a) Real SVHN images.

(b) EiNet SVHN samples.

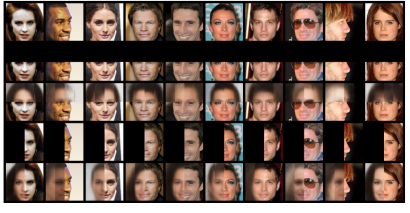


(c) Real images (top), covered images, and EiNet reconstructions



(d) Real CelebA samples.

(e) EiNet CelebA samples.



(f) Real images (top), covered images, and EiNet reconstructions

Adversarial smoothing

Certify robustness for inputs $\mathbf{x}$ by ***smoothing*** it by computing

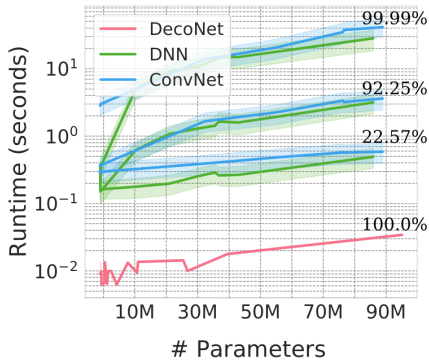
$$g_\sigma(\mathbf{x}) = \mathbb{E}_{\mathbf{e}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f(\mathbf{x}+\mathbf{e})\right]$$

Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", Uncertainty in Deep Learning Workshop ICML, 2021

Adversarial smoothing

Certify robustness for inputs $\mathbf{x}$ by ***smoothing*** it by computing

$$g_\sigma(\mathbf{x}) = \mathbb{E}_{\mathbf{e}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f(\mathbf{x}+\mathbf{e})\right]$$



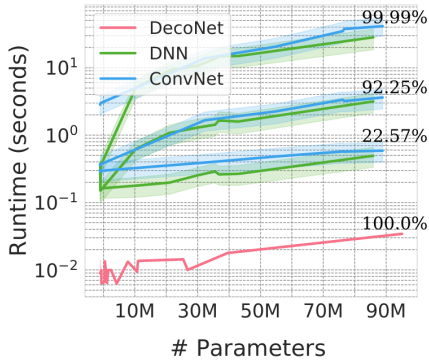
Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", Uncertainty in Deep Learning Workshop ICML, 2021

Adversarial smoothing

Certify robustness for inputs $\mathbf{x}$ by ***smoothing*** it by computing

$$g_\sigma(\mathbf{x}) = \mathbb{E}_{\mathbf{e}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f(\mathbf{x}+\mathbf{e})\right]$$

in a single feed-forward evaluation, if we ***impose some structure*** over a computational graph

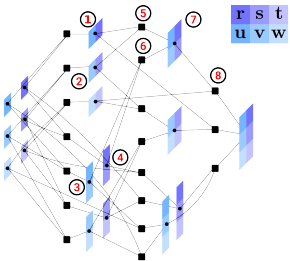


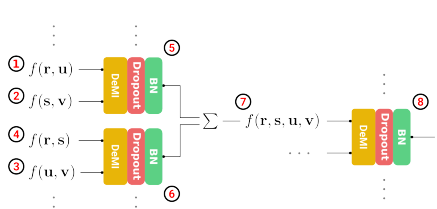
Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", Uncertainty in Deep Learning Workshop ICML, 2021

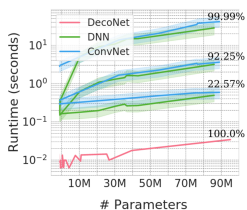
Adversarial smoothing

Exact smoothing with DecoNets *[Subramani et al. 2021]*

Using image circuits with shallow neural networks as inputs (“DecoNets”) delivers **competitive image classifiers** which allow **exact probabilistic smoothing**.





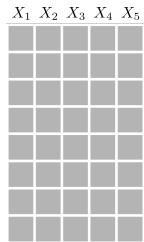


MNIST (300K)			FMNIST (1M)		
DNN	ConvNet	DecoNet	DNN	ConvNet	DecoNet
98.7	99.3	99.1	88.6	90.3	90.1

Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

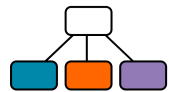
Expand regions with **clustering**



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

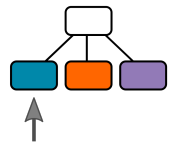
Number of clusters = number of partitions



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

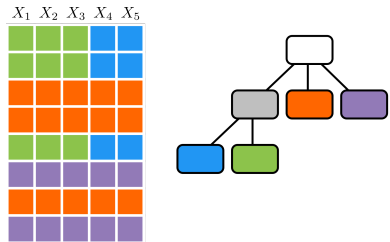
Try to find independent groups of variables
(e.g. independence tests)



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

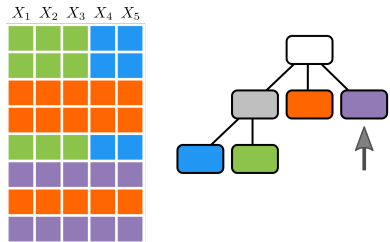
Success $\longrightarrow$ ***partition*** into new regions



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

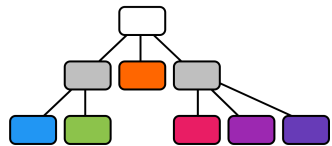
Try to find independent groups of variables
(e.g. independence tests)



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

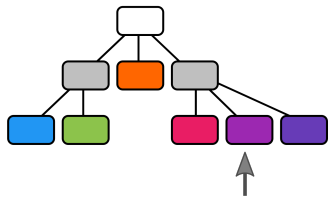
Success $\longrightarrow$ ***partition*** into new regions



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

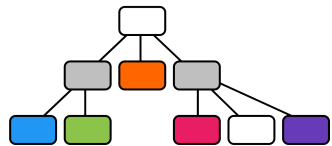
Single variable



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

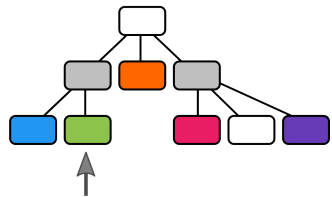
Single variable $\rightarrow$ ***input region***



Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

Expand regions with **clustering**

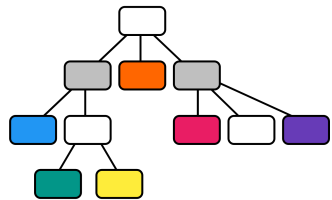


Data-driven structure learning

"Recursive data slicing" *[Gens and Domingos 2013]*

Number of clusters = number of partitions

And so on…



Data-driven structure learning

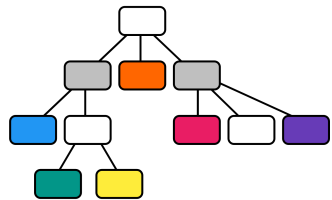
"Recursive data slicing" *[Gens and Domingos 2013]*

Stopping conditions: minimal number of features, samples, depth, …

Clustering ratios also deliver (initial) parameters

Smooth & Decomposable Circuits

Tractable integration



LearnSPN

Selected references

ID-SPN *[Rooshenas and Lowd 2014]*

LearnSPN-b/T/B *[Vergari, Di Mauro, and Esposito 2015]*

For **heterogeneous data** *[Molina et al. 2018]*

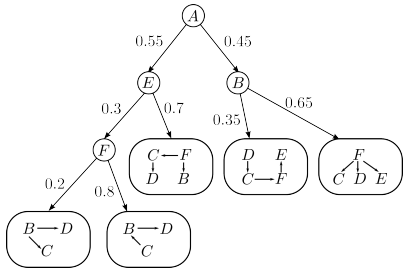
Using **k-means** *[Butz et al. 2018]* or **SVD** splits *[Adel, Balduzzi, and Ghodsi 2015]*

Learning **DAGs** *[Dennis and Ventura 2015; Jaini, Ghose, and Poupart 2018]*

Approximating independence tests *[Di Mauro et al. 2018]*

Cutset networks

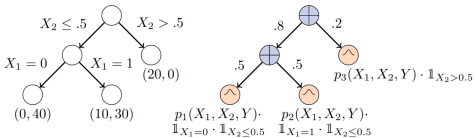
Besides clustering, **decision tree learning** can be used as PC learner. **Cutset networks,** decision trees over simple probabilistic models (Chow-Liu trees) *[Rahman, Kothalkar, and Gogate 2014]:*

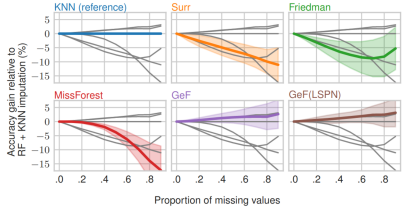


Cutset networks can easily be converted into **smooth, decomposable and deterministic PCs**.

Decision trees as PCs

Also vanilla decision tree learners can be used to learn PCs, by augmenting the leaves with distributions over inputs *[Correia, Peharz, and Campos 2020].* Allows to treat **missing features** and **outlier detection**.





Learning probabilistic circuits

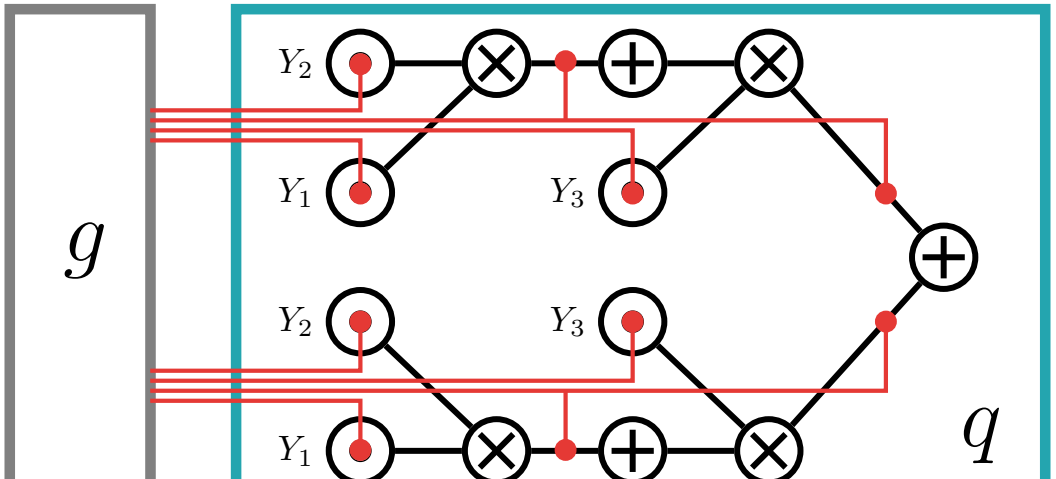
	Parameters	Structure
Generative	**deterministic** closed-form MLE *[Kisa et al. 2014b; Peharz, Gens, and Domingos 2014]* **non-deterministic** EM *[Poon and Domingos 2011; Peharz 2015; Zhao, Poupart, and Gordon 2016]* SGD *[Sharir et al. 2016; Peharz et al. 2019b]* Bayesian *[Jaini et al. 2016; Rashwan, Zhao, and Poupart 2016]* *[Zhao et al. 2016; Trapp et al. 2019; Vergari et al. 2019]*	**greedy** top-down *[Gens and Domingos 2013; Rahman, Ko* *[Vergari, Di Mauro, and Esposito 2015]* bottom-up **hill climbing** *[Lowd and Domingos 2008; Lowd an* *[**olascoaga2019towards**; Dennis and Ventura 2* *[**dangstrudel**]* **random** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di*
Discriminative	**?**	**?**

Learning probabilistic circuits

	Parameters	Structure
Generative	**deterministic** closed-form MLE *[Kisa et al. 2014b; Peharz, Gens, and Domingos 2014]* **non-deterministic** EM *[Poon and Domingos 2011; Peharz 2015] [Zhao, Poupart, and Gordon 2016]* SGD *[Sharir et al. 2016; Peharz et al. 2019b]* Bayesian *[Jaini et al. 2016; Rashwan, Zhao, and Poupart 2016] [Zhao et al. 2016; Trapp et al. 2019; Vergari et al. 2019]*	**greedy** top-down *[Gens and Domingos 2013; Rahman, Kothalkar,* *[Vergari, Di Mauro, and Esposito 2015]* bottom-up *[Peharz,* **hill climbing** *[Lowd and Domingos 2008; Lowd and Rooshe* *[**olascoaga2019towards**; Dennis and Ventura 2015; Lian* *[**dangstrudel**]* **random** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di Mauro e*
Discriminative	**deterministic** convex-opt MLE *[Liang and Broeck 2019]* **non-deterministic** EM *[Rashwan, Poupart, and Zhitang 2018]* SGD *[Gens and Domingos 2012; Sharir et al. 2016] [Peharz et al. 2019b]*	**greedy** top-down *[Shao et al. 2020]* **hill climbing** *[Rooshenas and Lowd 2016; Liang and Broec*

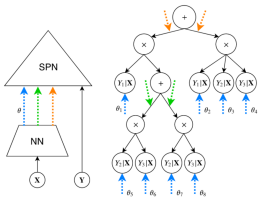
Conditional PCs

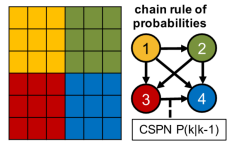
[Shao et al. 2020]

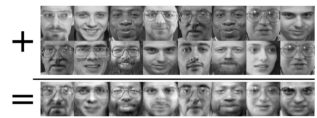


Conditional PCs

[Shao et al. 2020]







Information

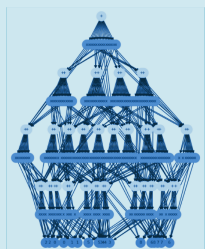
Prior Knowledge
domain assumptions
constraints
other models

Data
experimental data
samples
measurements

compilation **learning**

Circuits

Structure



Parameters

$\boldsymbol{\theta}$, $\boldsymbol{w}$

decomposability
smoothness
determinism
compatibility

generative
discriminative
Bayesian
credal