Probabilistic Circuits

Representation
Learning
Reasoning

Antonio Vergari
University of Edinburgh
avergari@exseed.ed.ac.uk

Robert Peharz
Graz University of Technology
robert.peharz@tugraz.at

YooJung Choi
Arizona State University
yj.choi@asu.edu

December 5th, 2022 - ***Tutorial @ NeurIPS 2022***

Why?

healthcare

loan grants

self-driving cars

ML models are everywhere...!

Why?

healthcare

loan grants

self-driving cars

neural networks are everywhere…!

Why?

struggle with uncertainty

be unfair

be not robust

but they can...

Why?

q_1 *"What is the probability of a treatment for a patient with **unavailable records**?"*

q_2 *"How **fair** is the prediction is a certain protected attribute changes?"*

q_3 *"Can we certify no **adversarial examples** exist?"*

how can we reason about their behavior?

Why

q_1 *"What is the probability of a treatment for a patient with **unavailable records**?"*

q_2 *"How **fair** is the prediction is a certain protected attribute changes?"*

q_3 *"Can we certify no **adversarial examples** exist?"*

reliably and efficiently?

Why?

"How can we design and learn deep learning models that can reliably reason?"

Why?

"How can we design and learn deep learning models that can reliably reason?"

expressive and ***flexible*** computational graphs

Why?

"How can we design and learn deep learning models that can reliably reason?"

seamlessly integrate ***probabilistic*** and ***logical*** inference

Why?

"How can we design and learn deep learning models that can reliably reason?"

exact and ***efficient*** inference

How?

structure!

structure!

structure!

How?

structure!

structure!

structure!

impose structure over ***computational graphs***

How?

structure!

structure!

structure!

exploit structure ***in the reasoning task***

How?

structure!

structure!

structure!

inject ***prior background knowledge***

Probabilistic Circuits

a grammar for structured tractable deep learning models

Probabilistic Circuits

a grammar for structured tractable deep learning models

Building Circuits

imposing structure and learning parameters from data and prior knowledge

Probabilistic Circuits

a grammar for structured tractable deep learning models

Building Circuits

imposing structure and learning parameters from data and prior knowledge

Advanced Reasoning

how do structure and reasoning interplay for real-world applications

Probabilistic Circuits

Reasoning about ML models


q_1 *"What is the probability of a treatment for a patient with **unavailable records**?"*

q_2 *"How **fair** is the prediction is a certain protected attribute changes?"*

q_3 *"Can we certify no **adversarial examples** exist?"*

Reasoning about ML models

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

...in the language of probabilities

Inspecting behaviors

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

it is crucial we compute them exactly and in polytime!

Inspecting behaviors


q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

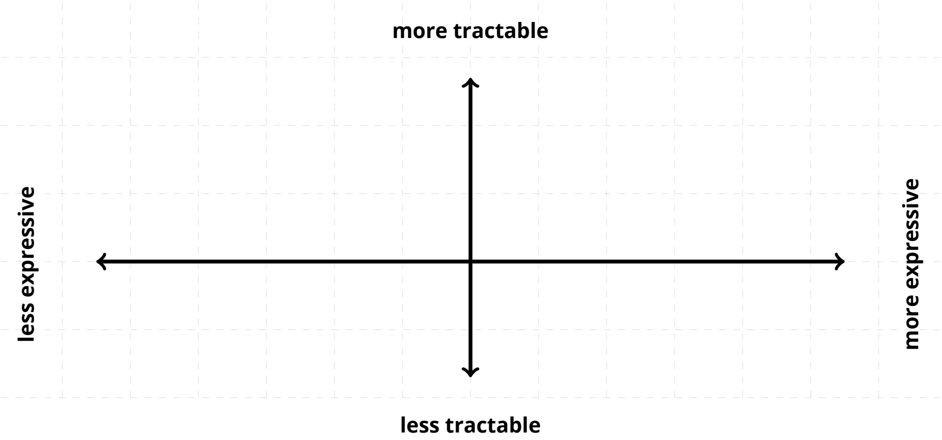
it is crucial we compute them tractably!

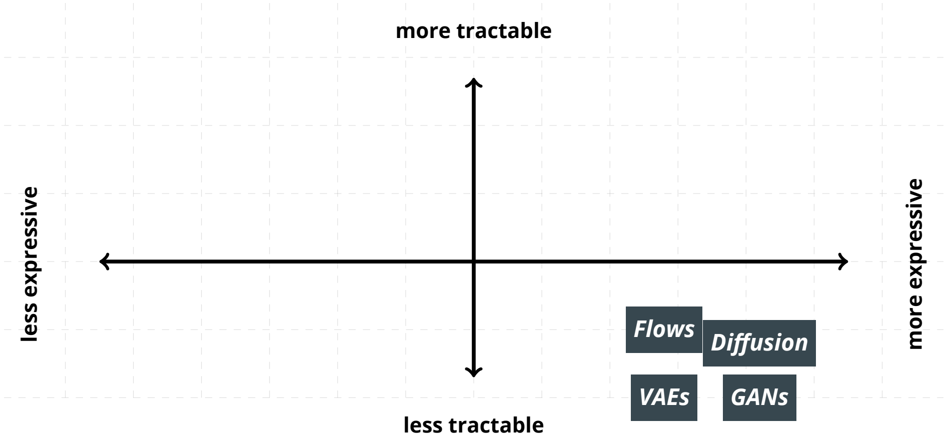
Goal

Given a reasoning task
can we design
a class of expressive models
that is tractable for it?

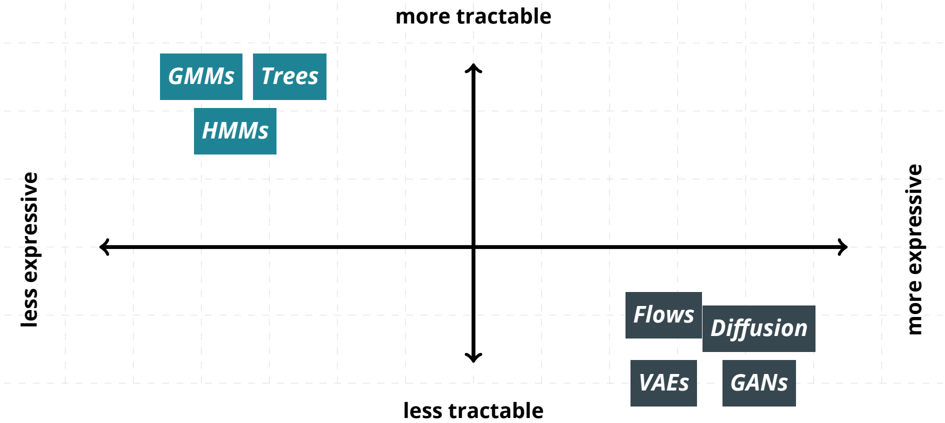
Goal

Given a reasoning task
can we design
a class of deep computational graphs
that is tractable for it?

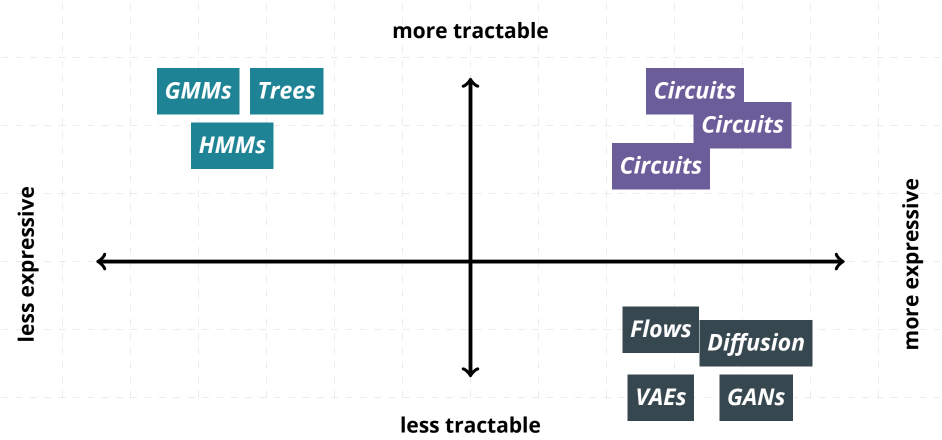

more tractable
less expressive
more expressive
less tractable



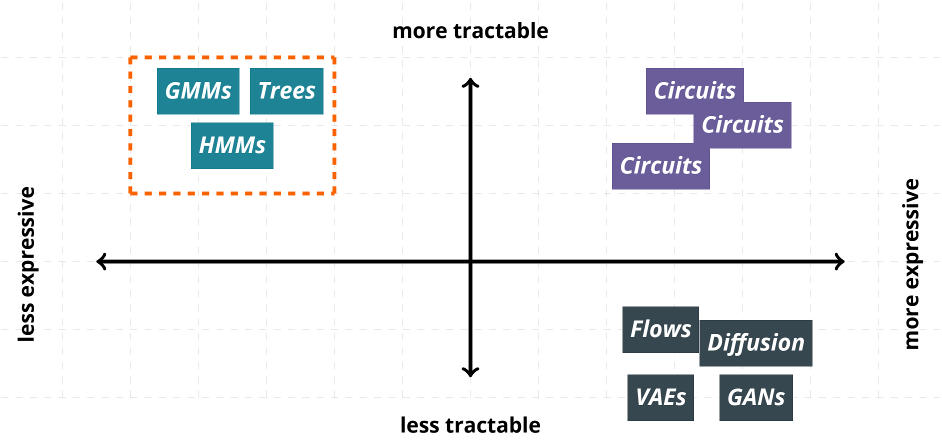
Expressive models are not much tractable...



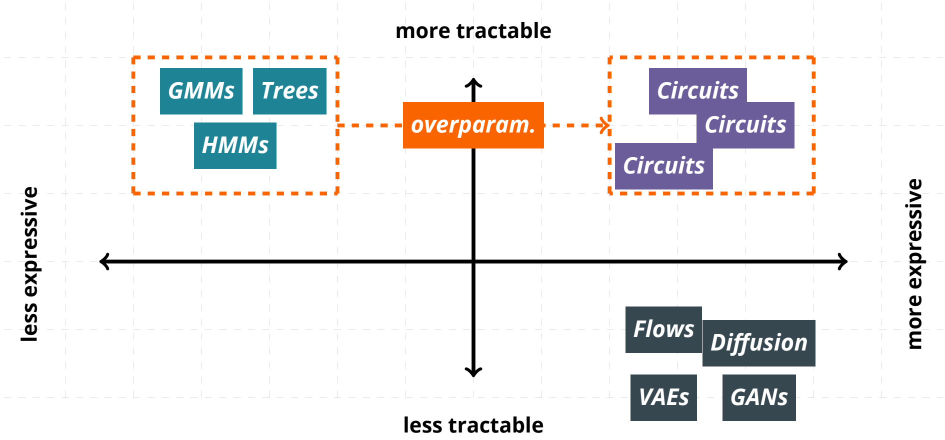
Tractable models are not that expressive...



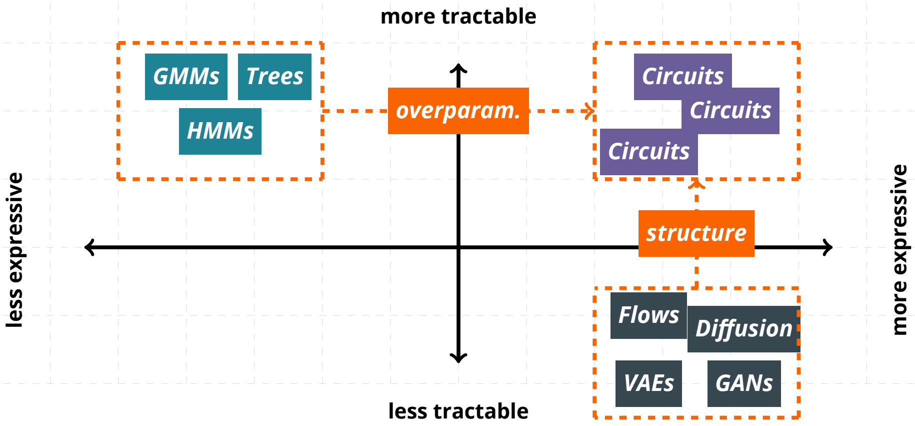
Circuits can be both expressive and tractable!



Start simple...



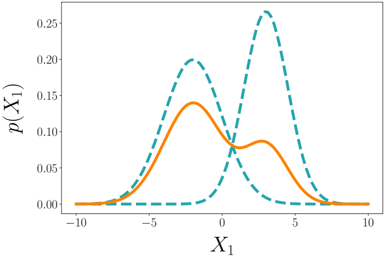
then make it more expressive!



impose structure!

GMMs

as computational graphs

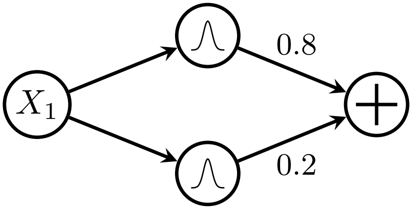


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

$\Longrightarrow$ *translating inference to data structures...*

GMMs

as computational graphs

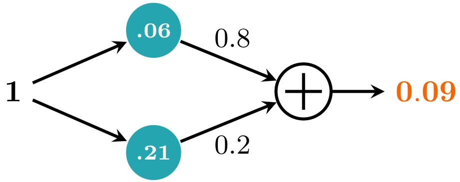


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

$\Rightarrow$ *...e.g., as a weighted sum unit over Gaussian input distributions*

GMMs

as computational graphs

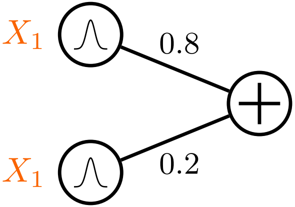


$$p(X=1) = 0.2 \cdot p_1(X_1=1) + 0.8 \cdot p_2(X_1=1)$$

$\Longrightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



A simplified notation:

$\Rightarrow$ ***scopes*** *attached to inputs*

$\Rightarrow$ *edge directions omitted*

GMMs

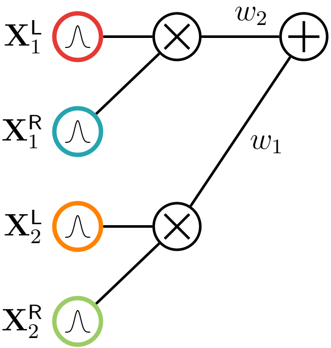
as computational graphs

$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}_1^{\mathsf{L}}) \cdot p_1(\mathbf{X}_1^{\mathsf{R}}) + \\ w_2 \cdot p_2(\mathbf{X}_2^{\mathsf{L}}) \cdot p_2(\mathbf{X}_2^{\mathsf{R}})$$

$\Longrightarrow$ *local factorizations...*

GMMs

as computational graphs

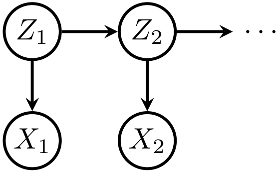


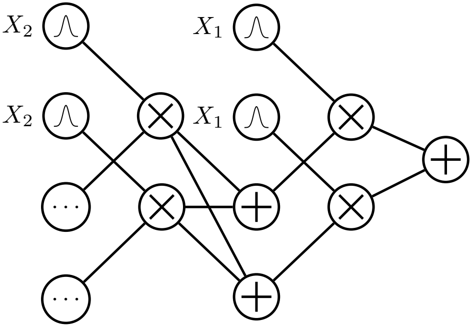
$$
\begin{aligned}
p(\mathbf{X}) = & w_1 \cdot p_1(\mathbf{X}_1^{\mathsf{L}}) \cdot p_1(\mathbf{X}_1^{\mathsf{R}}) + \\
& w_2 \cdot p_2(\mathbf{X}_2^{\mathsf{L}}) \cdot p_2(\mathbf{X}_2^{\mathsf{R}})
\end{aligned}
$$

$\Rightarrow$ *...are product units*

HMMs

as computational graphs





Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

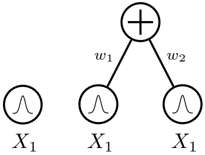
I. *A simple tractable function is a circuit*

Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

II. *A weighted combination of circuits is a circuit*



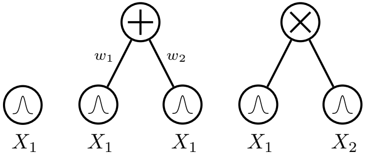
Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

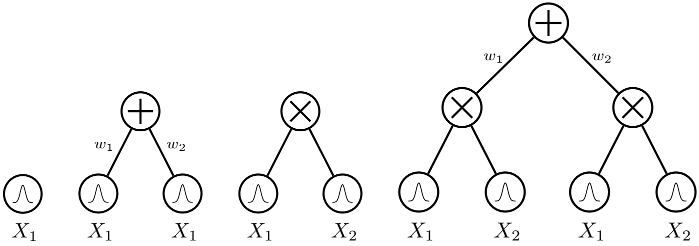
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



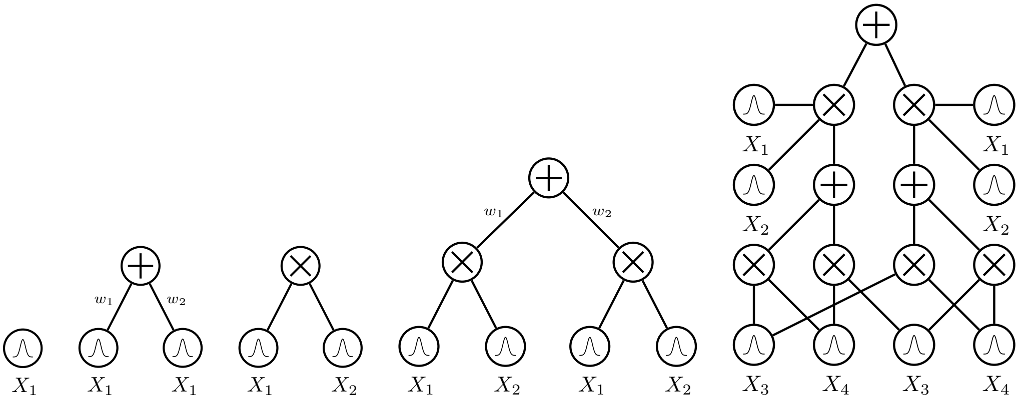
Probabilistic Circuits (PCs)

A grammar for tractable computational graphs



Probabilistic Circuits (PCs)

A grammar for tractable computational graphs



Building PCs in Python with SPFlow

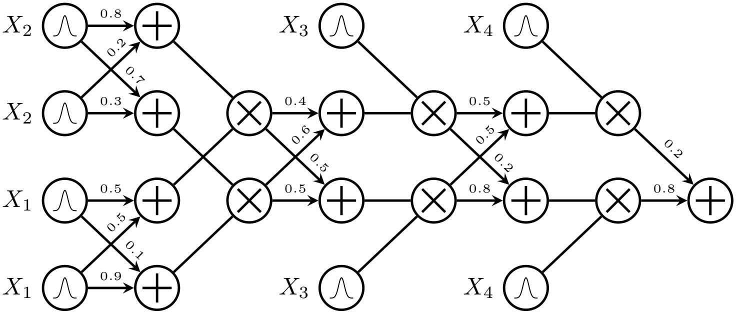
```
import spn.structure.leaves.parametric.Parametric as param
from param import Categorical, Gaussian

PC = 0.4 * (Categorical(p=[0.2, 0.8], scope=0) *
             (0.3 * (Gaussian(mean=1.0, stdev=1.0, scope=1) *
                     Categorical(p=[0.4, 0.6], scope=2))
            + 0.7 * (Gaussian(mean=-1.0, stdev=1.0, scope=1) *
                     Categorical(p=[0.6, 0.4], scope=2)))) \
    + 0.6 * (Categorical(p=[0.2, 0.8], scope=0) *
             Gaussian(mean=0.0, stdev=0.1, scope=1) *
             Categorical(p=[0.4, 0.6], scope=2))
```

Molina et al., "SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks", 2019

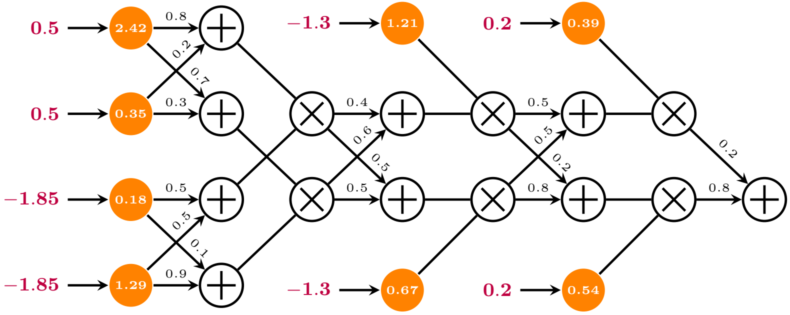
Probabilistic queries = feedforward evaluation

$$p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2)$$



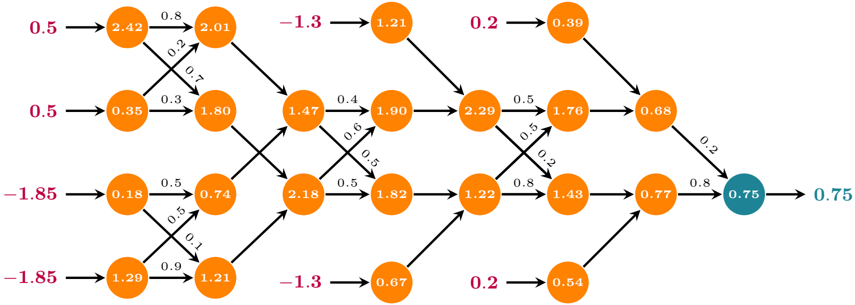
Probabilistic queries = feedforward evaluation

$$p(X_1 = \mathbf{-1.85}, X_2 = \mathbf{0.5}, X_3 = \mathbf{-1.3}, X_4 = \mathbf{0.2})$$

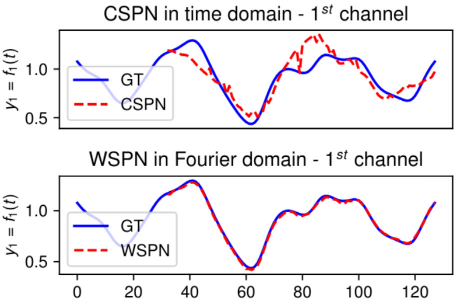


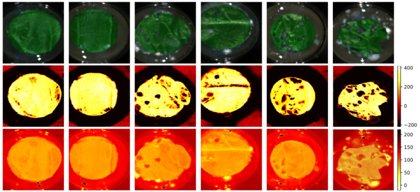
Probabilistic queries = feedforward evaluation

$$p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) = 0.75$$



Tractable likelihoods





modeling time, frequencies, latent spaces...

Yu et al., "Whittle Networks: A Deep Likelihood Model for Time Series", 2021

...why PCs?

1. A grammar for tractable models

One formalism to represent many models. *#GMMs #HMMs #Trees #XGBoost, ...*

...why PCs?

1. A grammar for tractable models

One formalism to represent many models. *#GMMs #HMMs #Trees #XGBoost, ...*

2. Expressiveness

Stacking millions latent variables. *#hierachical #mixtures #polynomials*

How expressive?

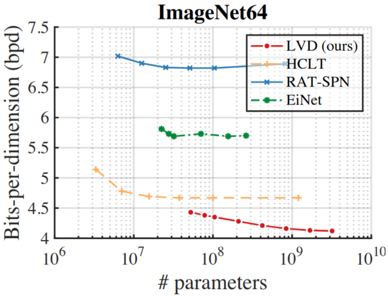
Dataset	Sparse PC (ours)	HCLT	RatSPN	IDF	BitSwap	BB-ANS	McBits
MNIST	**1.14**	1.20	1.67	1.90	1.27	1.39	1.98
EMNIST(MNIST)	**1.52**	1.77	2.56	2.07	1.88	2.04	2.19
EMNIST(Letters)	**1.58**	1.80	2.73	1.95	1.84	2.26	3.12
EMNIST(Balanced)	**1.60**	1.82	2.78	2.15	1.96	2.23	2.88
EMNIST(ByClass)	**1.54**	1.85	2.72	1.98	1.87	2.23	3.14
FashionMNIST	**3.27**	3.34	4.29	3.47	3.28	3.66	3.72

competitive with Flows and VAEs!

Dang et al., “Sparse Probabilistic Circuits via Pruning and Growing”, 2022

How scalable?

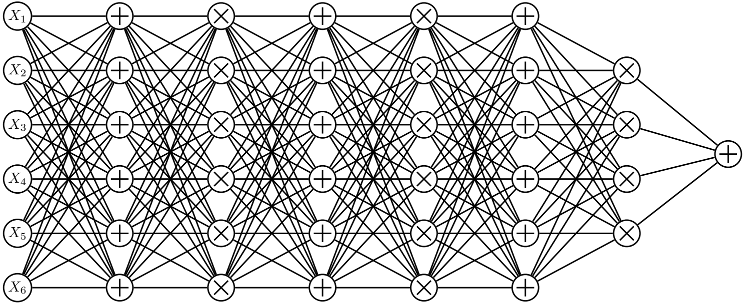
Dataset	TPMs				DGMs		
	LVD (ours)	HCLT	EiNet	RAT-SPN	Glow	RealNVP	BIVA
ImageNet32	**4.39**±0.01	4.82	5.63	6.90	4.09	4.28	3.96
ImageNet64	**4.12**±0.00	4.67	5.69	6.82	3.81	3.98	-
CIFAR	**4.38**±0.02	4.61	5.81	6.95	3.35	3.49	3.08



up to billions of parameters

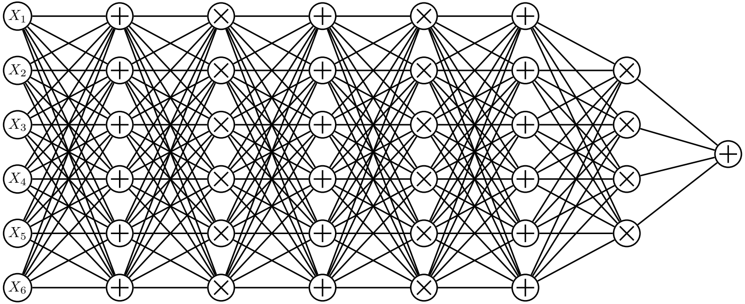
Liu et al., "Scaling Up Probabilistic Circuits by Latent Variable Distillation", 2022

Just sum, products and distributions?



just arbitrarily compose them like a neural network!

Just sum, products and distributions?



~~just arbitrarily compose them like a neural network!~~

$\Rightarrow$ *structural properties needed for tractability*

...why PCs?

1. A grammar for tractable models

One formalism to represent many models. *#GMMs #HMMs #Trees #XGBoost, ...*

2. Increase expressiveness

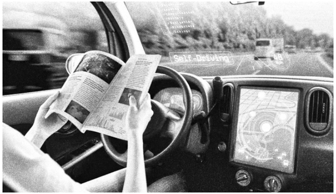
Stacking millions latent variables. *#hierachical #mixtures #polynomials*

3. Tractability == Structural Properties !!!

Exact computations for certain reasoning tasks are certified by verifying certain structural properties. *#marginals #expectations #MAP, ...*

Which structural properties

for complex reasoning



???

Structural properties

smoothness

decomposability

compatibility

determinism

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

Structural properties

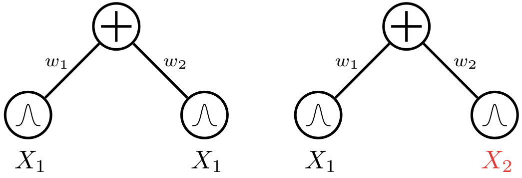
smoothness

decomposability

compatibility

determinism

the inputs of sum units are defined over the same variables



smooth* circuit** ***non-smooth circuit

__Vergari__ et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021

Structural properties

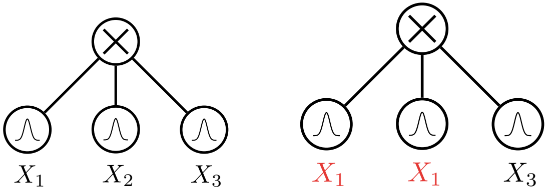
smoothness

decomposability

compatibility

determinism

the inputs of prod units are defined over disjoint variable sets



decomposable circuit ***non-decomposable circuit***

__Vergari__ et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021

smooth + decomposable circuits = ...

allow for the ***tractable*** computation of ***arbitrary integrals***

$$p(\mathbf{y}) = \int_{\mathsf{val}(\mathbf{Z})} p(\mathbf{z}, \mathbf{y}) \, d\mathbf{Z}, \quad \forall \mathbf{Y} \subseteq \mathbf{X}, \quad \mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$$

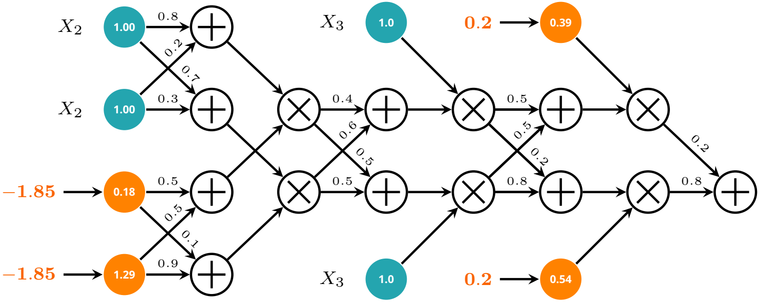
$\Longrightarrow$ *sufficient and necessary conditions for a **single feedforward evaluation***

$\Longrightarrow$ *can marginalize out any missing values*

Choi et al., "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling", 2021

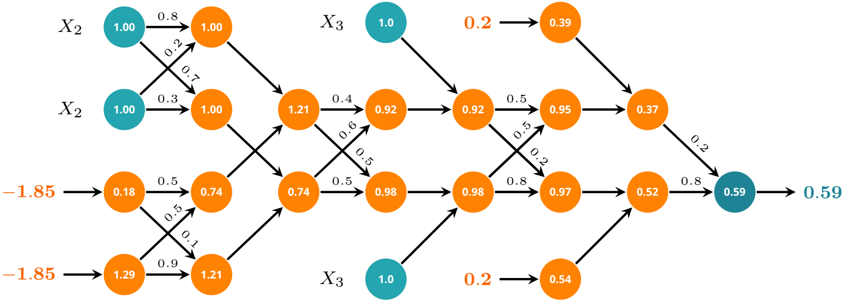
Probabilistic queries = feedforward evaluation

$$p(X_1 = -1.85, X_4 = 0.2)$$



Probabilistic queries = feedforward evaluation

$$p(X_1 = -1.85, X_4 = 0.2)$$



smooth + *decomposable* circuits = ...

Computing arbitrary integrations (or summations)

$\Longrightarrow$ *linear in circuit size!*

E.g., suppose we want to compute Z:

$$\int \boldsymbol{p}(\mathbf{x})d\mathbf{x}$$

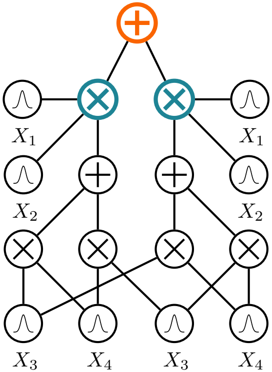
smooth + **decomposable** circuits = ...

If $\boldsymbol{p}(\mathbf{x}) = \sum_i w_i \boldsymbol{p}_i(\mathbf{x})$, (***smoothness***):

$$\int \boldsymbol{p}(\mathbf{x}) d\mathbf{x} = \int \sum_i w_i \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x} =$$

$$= \sum_i w_i \int \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x}$$

$\Longrightarrow$ *integrals are "pushed down" to inputs*



smooth + decomposable circuits = ...

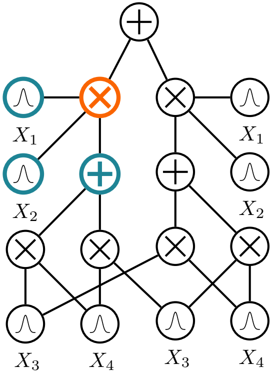
If $p(\mathbf{x},\mathbf{y},\mathbf{z}) = p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})$, (***decomposability***):

$$\int\int\int p(\mathbf{x},\mathbf{y},\mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int\int\int p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int p(\mathbf{x})d\mathbf{x}\int p(\mathbf{y})d\mathbf{y}\int p(\mathbf{z})d\mathbf{z}$$

$\Longrightarrow$ *integrals decompose into easier ones*



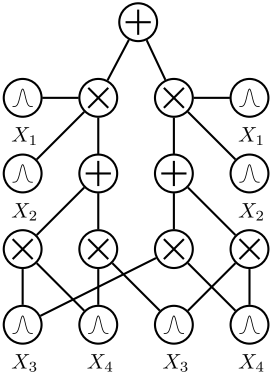
smooth + decomposable circuits = ...

Analogously, for arbitrary conditional queries:

$$p(\mathbf{q} \mid \mathbf{e}) = \frac{p(\mathbf{q}, \mathbf{e})}{p(\mathbf{e})}$$

1. evaluate $p(\mathbf{q}, \mathbf{e})$ $\Longrightarrow$ *one feedforward pass*
2. evaluate $p(\mathbf{e})$ $\Longrightarrow$ *another feedforward pass*

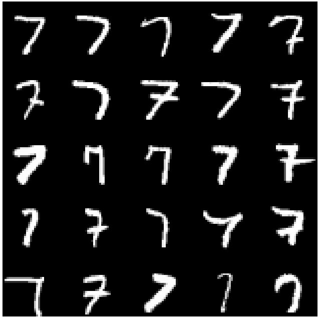
$\Longrightarrow$ *...still linear in circuit size!*



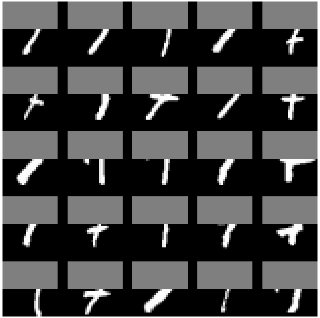
Tractable inference on PCs

Einsum networks

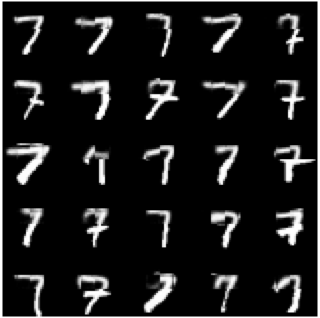
Original



Missing



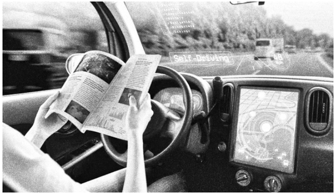
Conditional sample



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

Which structural properties

for complex reasoning



smooth + decomposable

Which structural properties

for complex reasoning

smooth + decomposable

??????

Adversarial smoothing

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

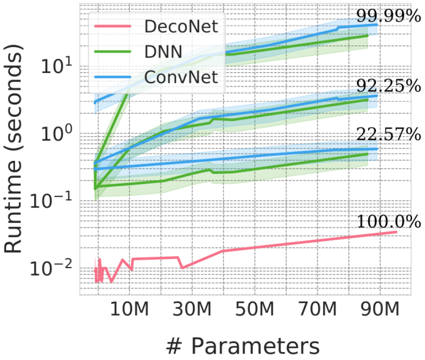
$$g_\sigma(\mathbf{x}) = \mathbb{E}_{\mathbf{e}\sim\mathcal{N}(0,\sigma\mathbf{I})}\left[f(\mathbf{x}+\mathbf{e})\right]$$

Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", 2021

Adversarial smoothing

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

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



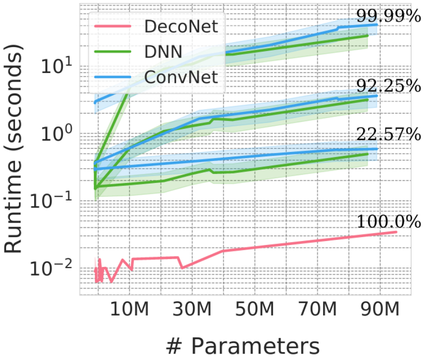
Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", 2021

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



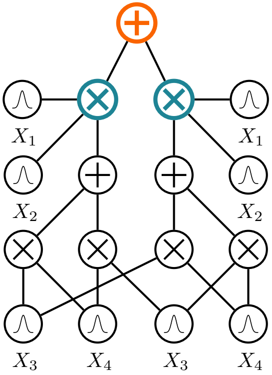
Subramani et al., "Exact and Efficient Adversarial Robustness with Decomposable Neural Networks", 2021

decomposable circuits = tractable adv smoothing

If $f(\mathbf{x}) = \sum_i w_i f_i(\mathbf{x})$:

$$\int \mathcal{N}(\mathbf{e}) f(\mathbf{x}+\mathbf{e}) d\mathbf{e} = \sum_i w_i \mathbb{E}_{\mathcal{N}(\mathbf{e})}[f_i(\mathbf{x}+\mathbf{e})]$$

$\Longrightarrow$ *expectations are "pushed down" to inputs*



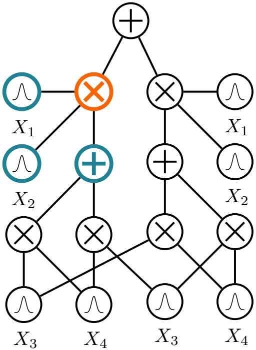
decomposable circuits = tractable *adv smoothing*

If $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = f(\mathbf{x}) f(\mathbf{y}) f(\mathbf{z})$, (***decomposability***):

$$\int \mathcal{N}(\mathbf{e_x})\mathcal{N}(\mathbf{e_y})\mathcal{N}(\mathbf{e_y}) f(\mathbf{x} + \mathbf{e_x}, \mathbf{y} + \mathbf{e_y}, \mathbf{z} + \mathbf{e_z}) d\mathbf{e_x} d\mathbf{e_y} d\mathbf{e_z}$$

$$\mathbb{E}_{\mathbf{e_x}}[f(\mathbf{x} + \mathbf{e_x})] \cdot \mathbb{E}_{\mathbf{e_y}}[f(\mathbf{y} + \mathbf{e_y})] \cdot \mathbb{E}_{\mathbf{e_z}}[f(\mathbf{z} + \mathbf{e_z})]$$

$\Longrightarrow$ *expectations decompose into easier ones*



Which structural properties

for complex reasoning

smooth + decomposable

decomposable

Which structural properties

for complex reasoning

smooth + decomposable

???????

decomposable

General expectations

Integrals involving two or more functions:

$$\int \boldsymbol{p}(\mathbf{x})\boldsymbol{f}(\mathbf{x})d\,\mathbf{X}$$

General expectations

Integrals involving two or more functions:

$$\int \boldsymbol{p}(\mathbf{x})\boldsymbol{f}(\mathbf{x})d\,\mathbf{X}$$

represent both $\boldsymbol{p}$ and $\boldsymbol{f}$ as circuits...but with which structural properties? E.g.,

General expectations

Integrals involving two or more functions:

$$\int p(\mathbf{x}) f(\mathbf{x}) d\,\mathbf{X}$$

represent both p and f as circuits...but with which structural properties? E.g.,

$$\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)}\left[f_0(\mathbf{x}_c)\right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)}\left[f_1(\mathbf{x}_c)\right]$$

Structural properties

smoothness

decomposability

compatibility

determinism

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

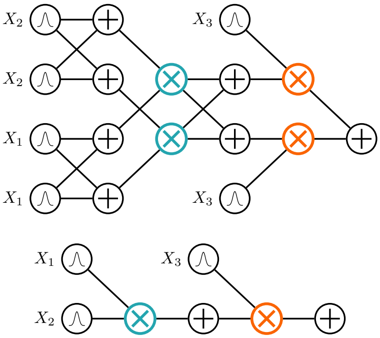
Structural properties

smoothness

decomposability

compatibility

determinism



compatible circuits

__Vergari__ et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021

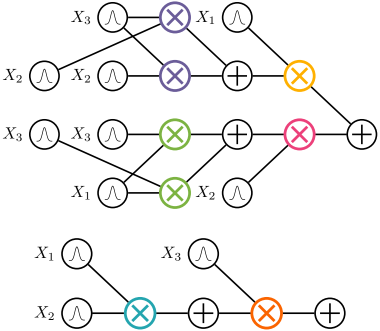
Structural properties

smoothness

decomposability

compatibility

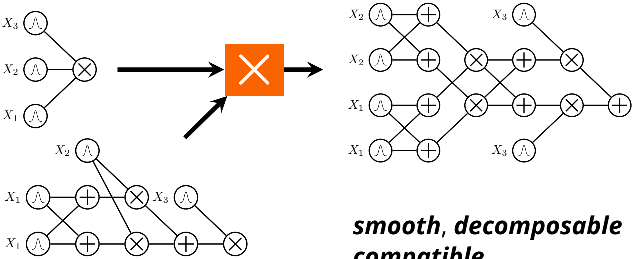
determinism



non-compatible circuits

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

Tractable products



smooth**, **decomposable
compatible

exactly compute $\int \boldsymbol{p}(\mathrm{x})\boldsymbol{f}(\mathrm{x})dX$ **in time** $O(|\boldsymbol{p}||\boldsymbol{f}|)$

__Vergari__ et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021

Which structural properties

for complex reasoning

smooth + decomposable

smooth + compatible

decomposable

Structural properties

smoothness

decomposability

compatibility

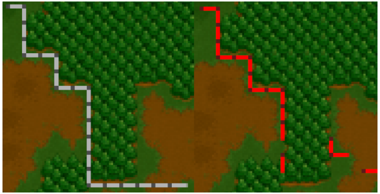
determinism

stay tuned!

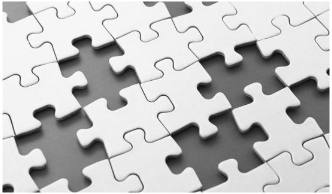
Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

Which structural properties

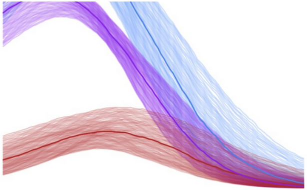
for complex reasoning



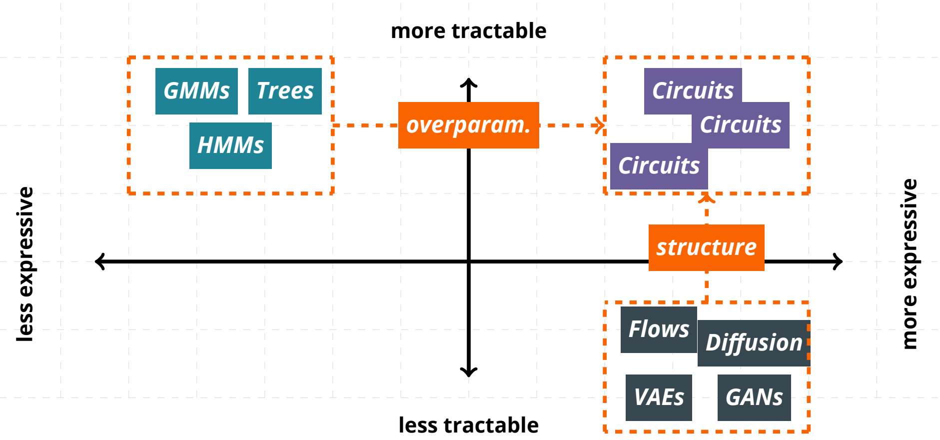
reason with constraints



expected predictions



computing uncertainties



structure + expressiveness

Building Probabilistic Circuits

Information

Prior Knowledge
domain assumptions
constraints
other models

Data
experimental data
samples
measurements

compilation

learning

Circuits

Structure

Parameters

decomposability
smoothness
determinism
compatibility

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

generative
discriminative
Bayesian
credal

Origins: Compilation

Knowledge compilation

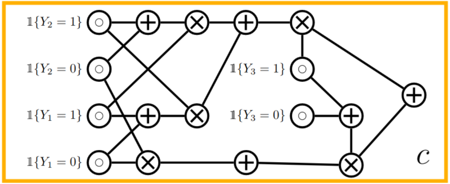
Tractable Boolean circuits *(Darwiche et al. 2002)*

- Compile logic prior knowledge into a propositional formula
- Natural representation: **deep logic circuits (negational normal form, NNF)**
- Equipped with structural properties such as **decomposability**, **smoothness**, **determinism**, etc., corresponding to various tractable inference routines (**SAT**, **model counting**, **entailment**, **equivalence**, ...)

Semantic Probabilistic Layers

(Ahmed et al. 2022a)

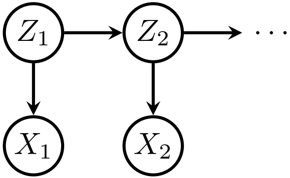
```
All cats are animals
All dogs are animals
...
```

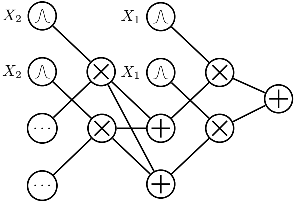


Compiling probabilistic graphical models

Arithmetic circuits (Darwiche 2002, 2003, 2009)

- Compile a given Bayesian network into an **arithmetic circuit**—syntactically equivalent to smooth, decomposable and deterministic PCs
- Either via logic encoding of Bayesian network + knowledge compilation
- Or record "execution trace" (sum and product operations) of traditional inference algorithms (junction tree, variable elimination)





Compilation

Selected references

Logic circuits, interplay between structural properties and tractable reasoning

(Darwiche et al. 2002)

Converting probabilistic graphical models via knowledge compilation

(Darwiche 2002)

Logic circuit compilers

(Darwiche 2004; Muise et al. 2012; Bova et al. 2015; Lagniez et al. 2017; Oztok et al. 2018)

Neuro-symbolic models using logic circuits

(Ahmed et al. 2022a,b)

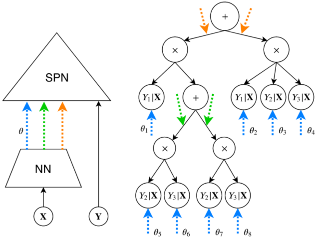
Parameter Learning

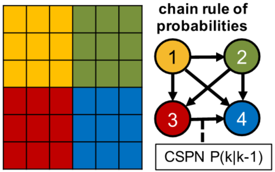
Gradient descent (of course)

- PCs are computational graphs
- Hence we can just learn them as any other neural net using SGD
- Use re-parameterization if parameters should satisfy constraints:
  - soft-max for sum-weights (non-negative, sum-to-one)
  - soft-plus for variances
  - low-rank plus diagonal for covariance matrices
- Allows for conditional distributions

Conditional PCs

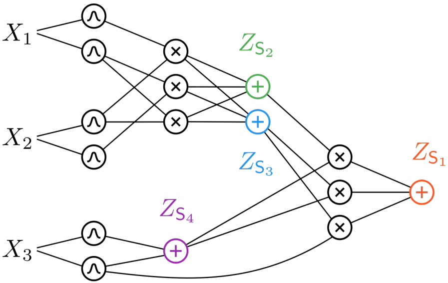
(Shao et al. 2019)

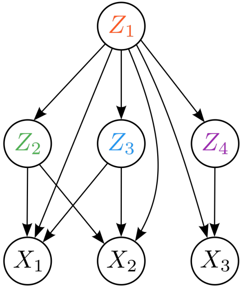




Maximum likelihood (frequentist)

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 autodfiff:

```
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

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 et al. 2016)*
- Collapsed variational inference algorithm *(Zhao et al. 2016)*
- Gibbs sampling *(Trapp et al. 2019; Vergari et al. 2019)*

Structure Learning

Region graphs

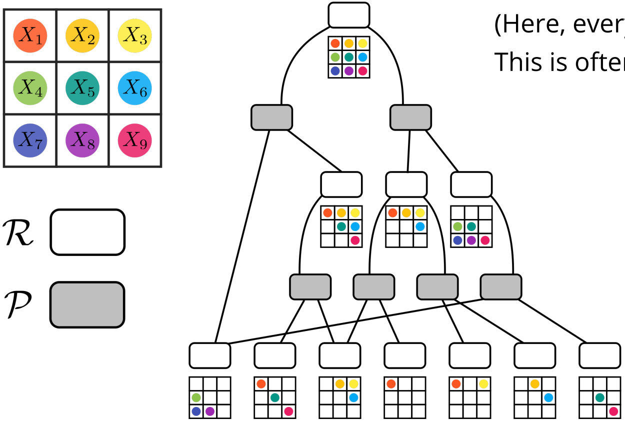
Laying out the PC structure on a high level

- Region graphs (RGs) describe decompositional structure
- RGs are bipartite, directed graphs containing regions ($\mathcal{R}$) and partitions ($\mathcal{P}$)
- Input and output nodes of the RG are regions
- Regions have a scope (receptive field), denoted as $sc(\mathcal{R}) \subseteq \mathbf{X}$
- For every partition $\mathcal{P}$ it holds that

$$sc(\mathcal{R}_{out}) = \bigcup_{\mathcal{R}_{in} \in inputs(\mathcal{P})} sc(\mathcal{R}_{in})$$

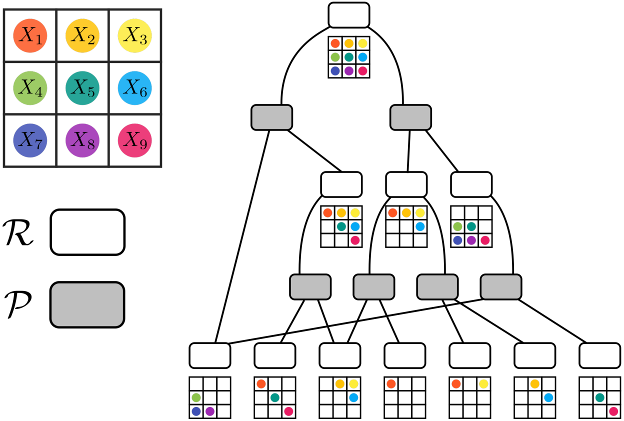
$$sc(\mathcal{R}') \cap sc(\mathcal{R}'') = \emptyset, \qquad \mathcal{R}' \neq \mathcal{R}'' \in inputs(\mathcal{P})$$

Example region graph



(Here, every partition has 2 input regions. This is often assumed, but not necessary.)

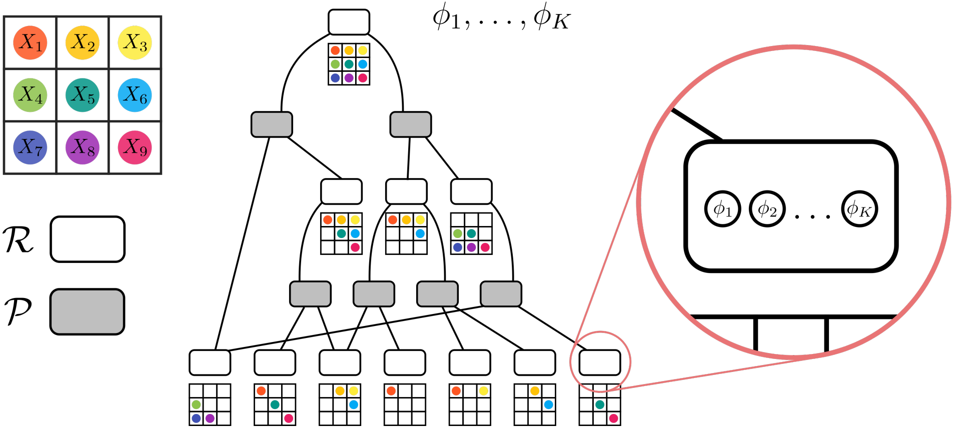
From region graphs to PCs



From region graphs to PCs

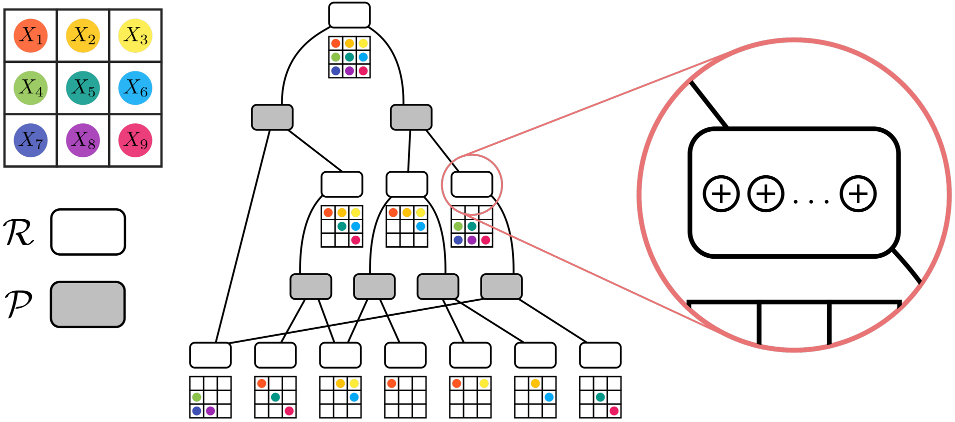
Equip each input region with non-linear units

$\phi_1, \ldots, \phi_K$

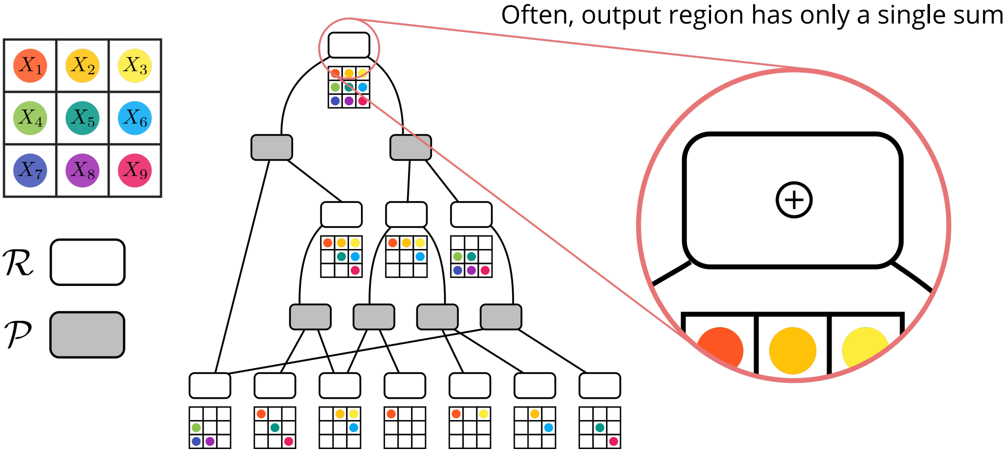


From region graphs to PCs

Equip each internal region with sum nodes

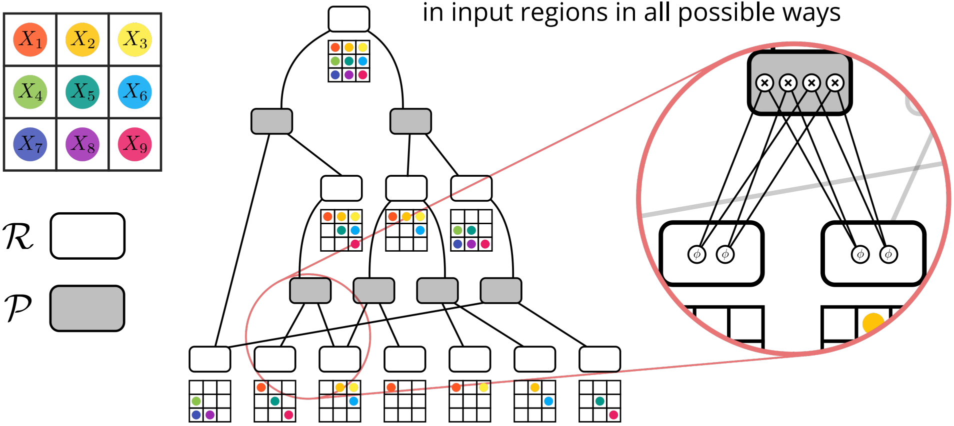


From region graphs to PCs



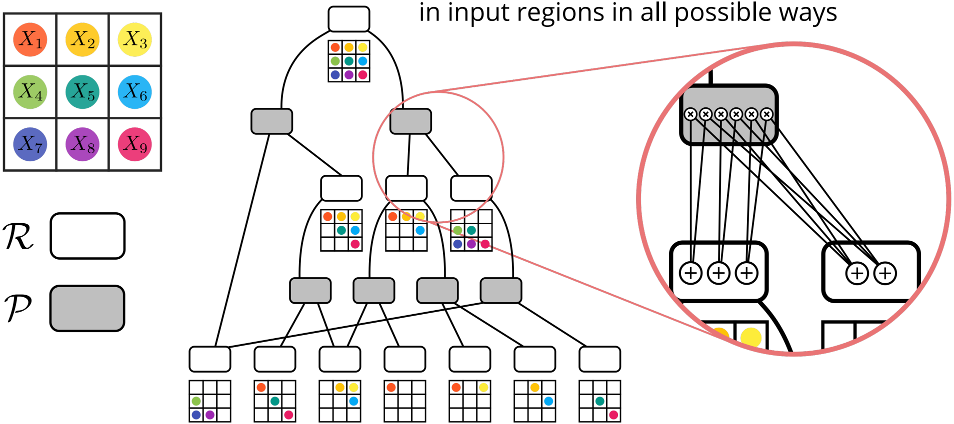
From region graphs to PCs

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



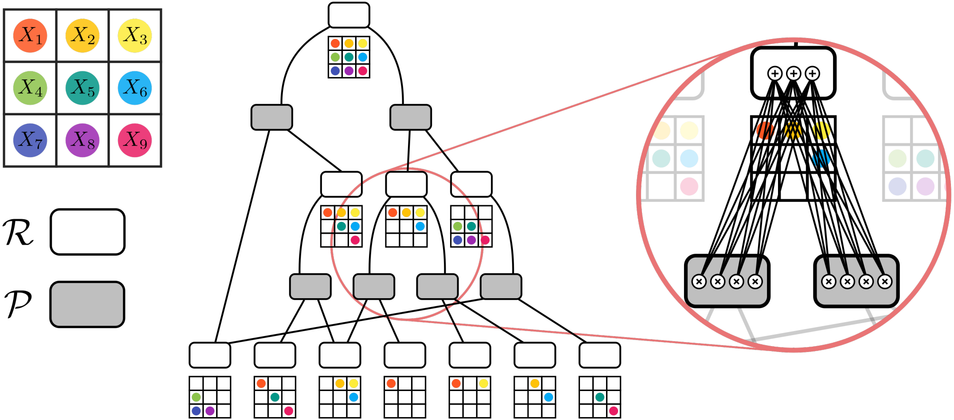
From region graphs to PCs

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



From region graphs to PCs

Connect products to sum units above



From region graphs to PCs

- Equip each input region (leaf) $\mathcal{R}$ with K units $\phi_1, \ldots, \phi_K$, which are non-linear functions over $sc(\mathcal{R})$. Usually, $\phi_1, \ldots, \phi_K$ are probability densities. K can be different for each input region.
- Equip each other region with K sum units. K can be different for each internal region. Often, for the root region $K = 1$, when PC is used as density estimator.
- Equip each partition $\mathcal{P}$ with as many products as there are combinations of units in the input regions to $\mathcal{P}$, selecting one unit from each region. Formally, if $\mathcal{P}$ has input regions $\mathcal{R}_1, \mathcal{R}_2 \ldots, \mathcal{R}_I$, insert one product $\prod_{i=1}^{I} u_i$ for each $(u_1, u_2, \ldots, u_I) \in \mathcal{R}_1 \times \mathcal{R}_2 \times \cdots \times \mathcal{R}_I$.
- Connect each $\prod_{i=1}^{I} u_i$ in $\mathcal{P}$ to all sum units in the output regions of $\mathcal{P}$.

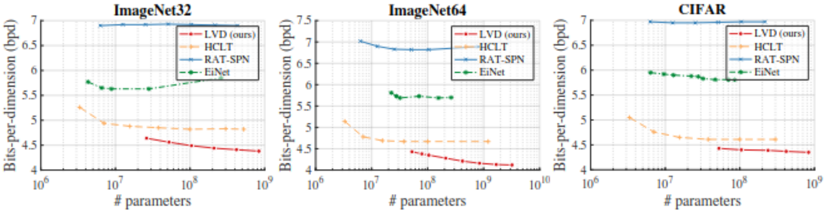
From region graphs to PCs

- Resulting PC has alternating sum and product units (not a strong constraint)
- We can easily scale the PC (overparameterize, increase expressivity) by equipping regions with more units
- RGs can be seen as a vectorized version of PCs – each region and partition can be seen as as a module
- Resulting PC will be smooth and decomposable, i.e., we can integrate, marginalize, and take conditionals
- After the PC has been constructed, we might discard the RG

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 et al., “Scaling Up Probabilistic Circuits by Latent Variable Distillation”, 2022

How to construct and learn RGs?

Random regions graphs

The "no-learning" option

(Peharz et al. 2019)

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

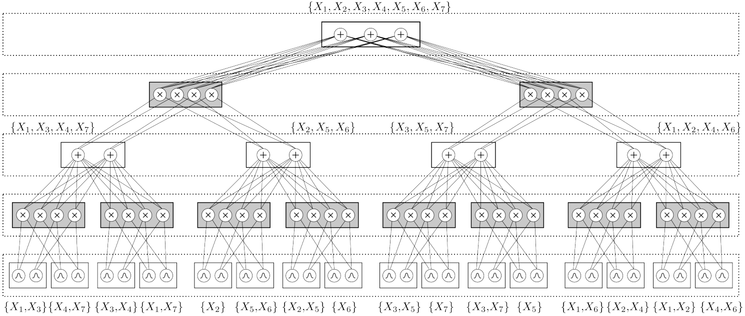
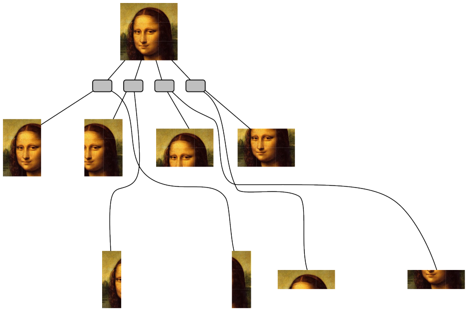


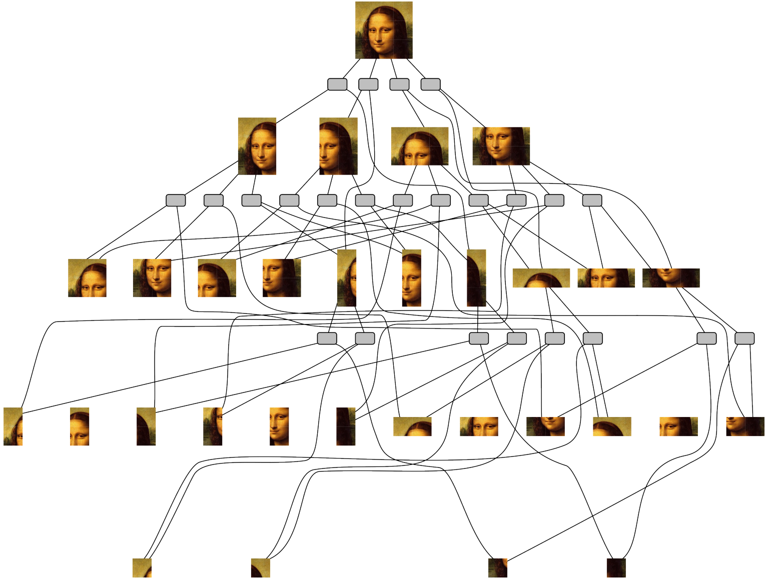
Image-tailored circuit structure

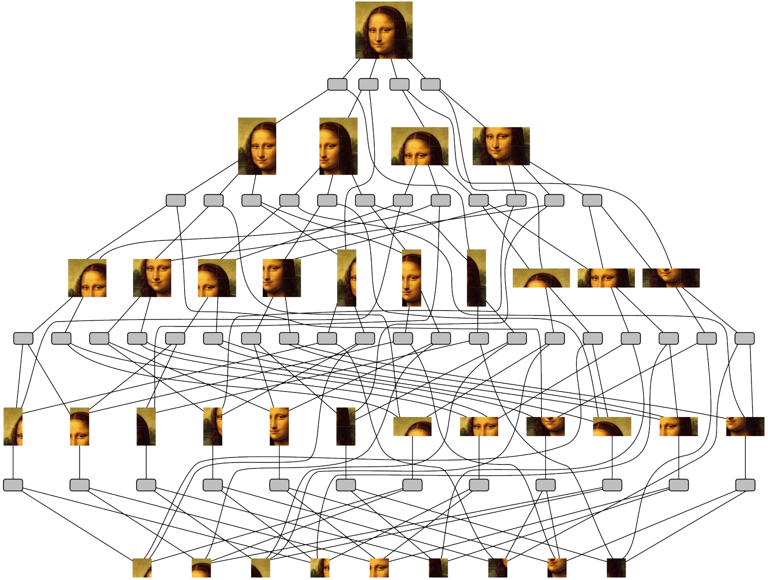
"Recursive image slicing" *(Poon et al. 2011)*

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





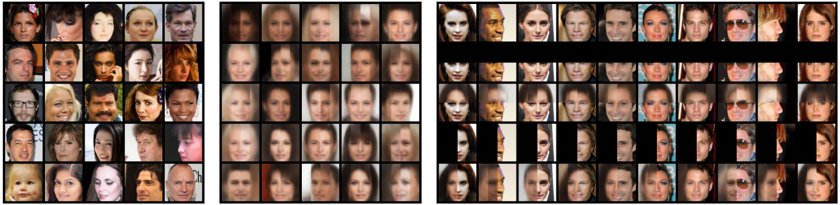


Generative modeling

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

- *Salman et al. 2019* showed that **smoothing** a classifier f with noise delivers strong guarantees on the non-existence of adversarial examples
- Specifically, if the output of f is bounded, then the smoothed classifier

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

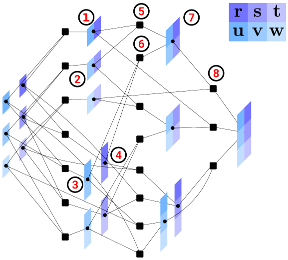
  will be Lipschitz, guaranteeing the non-existence of adversarial examples within a certain ℓ_2-ball around $\boldsymbol{x}$ (depending on the class margin and σ)
- How to compute $\bar{f}(\mathbf{x})$ for neural networks? **Monte Carlo** seems to only solution, which is never exact and requires **many evaluations for a <u>single</u> test sample**

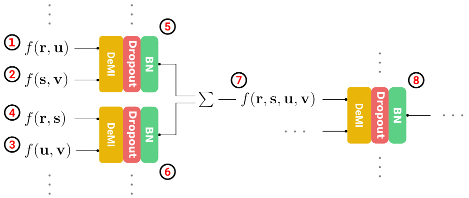
Adversarial smoothing

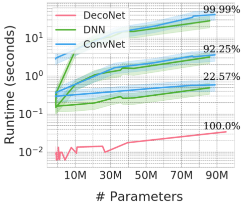
Exact smoothing with DecoNets

(Subramani et al. 2021)

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







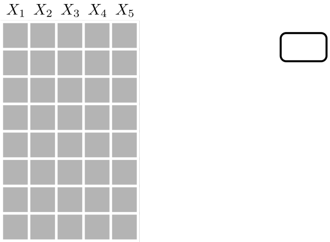
MNIST (300K)			FMNIST (1M)		
DNN	ConvNet	DecoNet	DNN	ConvNet	DecoNet
98.7	99.3	99.1	88.6	90.3	90.1

Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Expand regions with **clustering**

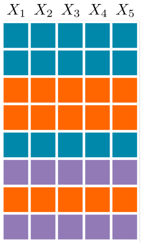


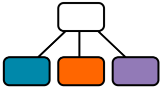
Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Number of clusters = number of partitions



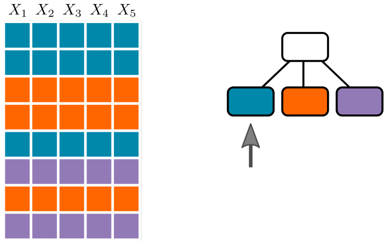


Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

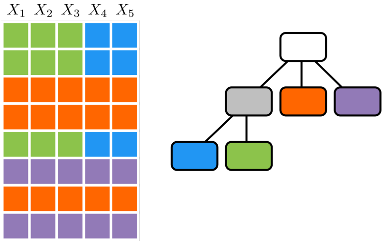
Try to find independent groups of variables
(e.g. independence tests)



Data-driven structure learning

“Recursive data slicing” *(Gens et al. 2013)*

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

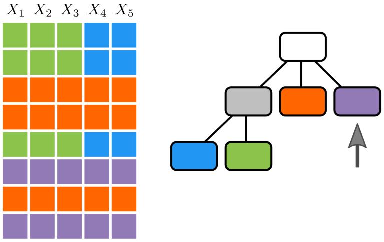


Data-driven structure learning

“Recursive data slicing”

(Gens et al. 2013)

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

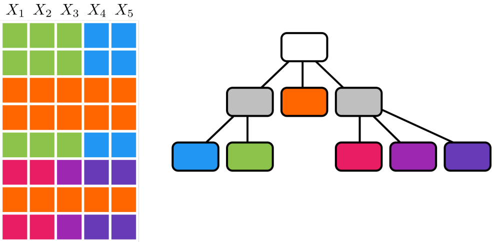


Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

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

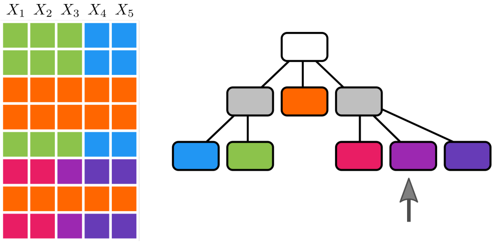


Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Single variable

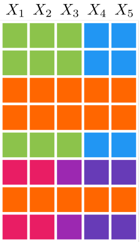


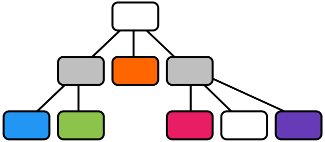
Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Single variable $\longrightarrow$ ***input region***



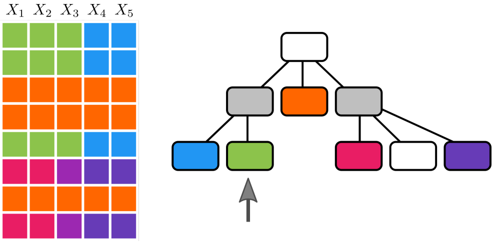


Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Expand regions with **clustering**



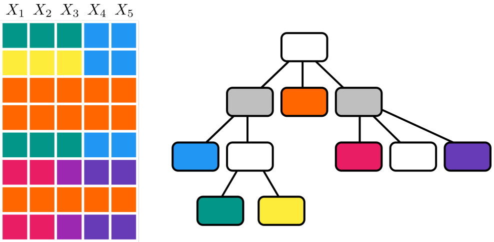
Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Number of clusters = number of partitions

And so on...

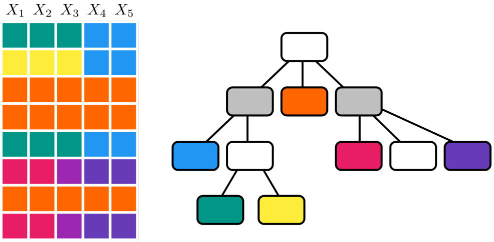


Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

- Stopping conditions: minimal number of features, samples, depth, ...
- Clustering ratios also deliver (initial) parameters
- **Smooth & Decomposable Circuits**
- **Tractable integration**



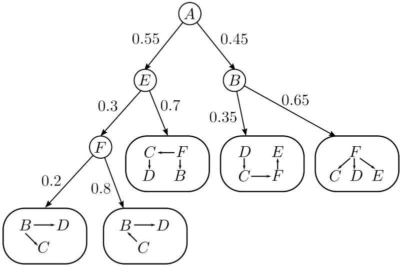
LearnSPN

Selected references

- **ID-SPN** *(Rooshenas et al. 2014)*
- **LearnSPN-b/T/B** *(Vergari et al. 2015)*
- For **heterogeneous data** *(Molina et al. 2018)*
- Using **k-means** *(Butz et al. 2018)* or **SVD** splits *(Adel et al. 2015)*
- Learning **DAGs** *(Dennis et al. 2015; Jaini et al. 2018)*
- **Approximating** independence tests *(Di Mauro et al. 2018)*

Cutset networks

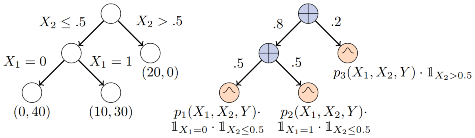
Besides clustering, **decision tree learning** can be used as PC learner. **Cutset networks,** decision trees over simple probabilistic models (Chow-Liu trees) *(Rahman et al. 2014):*

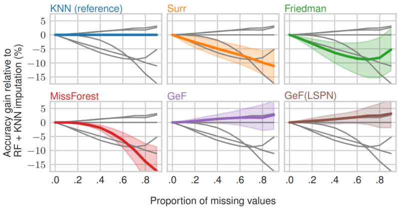


Cutset networks can easily be converted into **smooth, decomposable and deterministic PCs**.

Decision trees as PCs

Also vanilla decision tree learners can be used to learn PCs, by augmenting the leaves with distributions over inputs *(Correia et al. 2020).* Allows to treat **missing features** and **outlier detection**.





Information

Prior Knowledge
domain assumptions
constraints
other models

Data
experimental data
samples
measurements

compilation

learning

Circuits

Structure

Parameters

decomposability
smoothness
determinism
compatibility

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

generative
discriminative
Bayesian
credal

Advanced Reasoning with Probabilistic Circuits

Reasoning about ML models

q_1 *"What is the probability of a treatment for a patient with **unavailable records**?"*

$$\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$$

q_2 *"How **fair** is the prediction is a certain protected attribute changes?"*

$$\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$$

q_3 *"Can we certify no **adversarial examples** exist?"*

$$\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$$

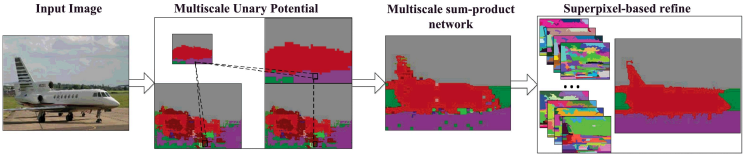
Maximum-a-posteriori (MAP) Inference

aka Most probable explanation (MPE)

E.g., multi-label classification: what are the most likely labels $\mathbf{y}$ for an input $\mathbf{x}$?

$$\operatorname*{argmax}_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$$

E.g., image segmentation: what is the most likely latent space for the given pixels?

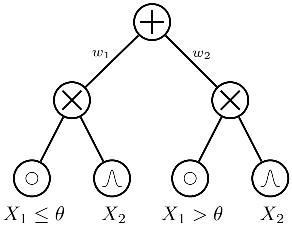


Yuan et al., "Modeling spatial layout for scene image understanding via a novel multiscale sum-product network", 2016
Friesen and Domingos, "Submodular Sum-product Networks for Scene Understanding", 2016

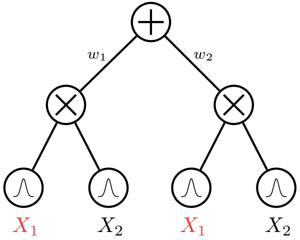
Determinism

aka support-decomposability

A sum unit is deterministic if its inputs have disjoint supports



deterministic circuit



non-deterministic circuit

Darwiche and Marquis, "A knowledge compilation map", 2002

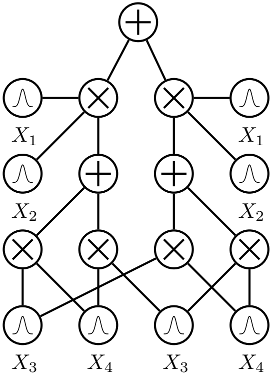
Determinism + decomposability = tractable MAP

Computing maximization with arbitrary evidence $\mathbf{e}$

$\Rightarrow$ *linear in circuit size!*

E.g., suppose we want to compute:

$$\max_{\mathbf{q}} p(\mathbf{q} \mid \mathbf{e})$$

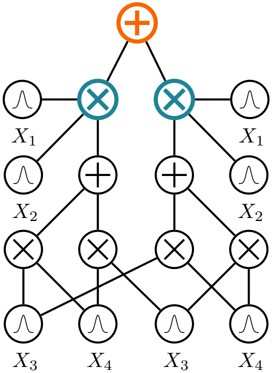


Determinism + **decomposability** = **tractable MAP**

If $p(\mathbf{q}, \mathbf{e}) = \sum_i w_i p_i(\mathbf{q}, \mathbf{e}) = \max_i w_i p_i(\mathbf{q}, \mathbf{e})$,
(***deterministic*** sum unit):

$$\begin{aligned}
\max_{\mathbf{q}} p(\mathbf{q}, \mathbf{e}) &= \max_{\mathbf{q}} \sum_i w_i p_i(\mathbf{q}, \mathbf{e}) \\
&= \max_{\mathbf{q}} \max_i w_i p_i(\mathbf{q}, \mathbf{e}) \\
&= \max_i \max_{\mathbf{q}} w_i p_i(\mathbf{q}, \mathbf{e})
\end{aligned}$$

$\Longrightarrow$ *one non-zero term, thus sum is max*

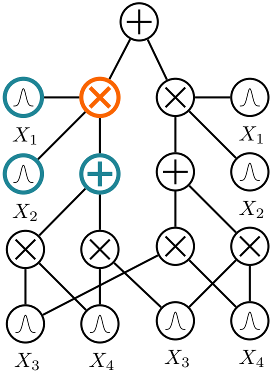


Determinism + decomposability = tractable MAP

If $p(\mathbf{q}, \mathbf{e}) = p(\mathbf{q_x}, \mathbf{e_x}, \mathbf{q_y}, \mathbf{e_y}) = p(\mathbf{q_x}, \mathbf{e_x})p(\mathbf{q_y}, \mathbf{e_y})$
(***decomposable*** product unit):

$$\max_{\mathbf{q}} p(\mathbf{q} \mid \mathbf{e}) = \max_{\mathbf{q}} p(\mathbf{q}, \mathbf{e})$$
$$= \max_{\mathbf{q_x}, \mathbf{q_y}} p(\mathbf{q_x}, \mathbf{e_x}, \mathbf{q_y}, \mathbf{e_y})$$
$$= \max_{\mathbf{q_x}} p(\mathbf{q_x}, \mathbf{e_x}) \cdot \max_{\mathbf{q_y}} p(\mathbf{q_y}, \mathbf{e_y})$$

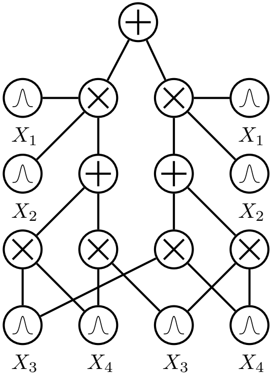
$\Longrightarrow$ *solving optimization independently*



Determinism + decomposability = tractable MAP

E.g., for $\text{argmax}_{x_1,x_3}\, p(x_1, x_3 \mid x_2, x_4)$:

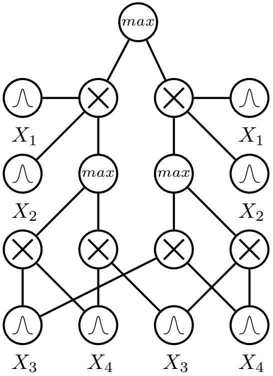
1. turn sum into max units and input distributions into max distributions
2. feedforward evaluation for $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
3. retrieve max activations in backward pass
4. compute **MAP states** for X_1 and X_3 at input units



Determinism + **decomposability** = **tractable MAP**

E.g., for $\text{argmax}_{x_1,x_3}\, p(x_1, x_3 \mid x_2, x_4)$:

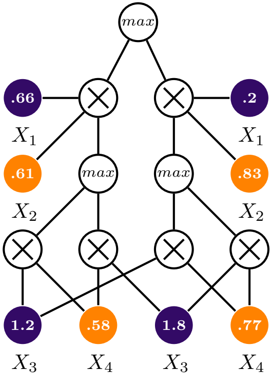
1. turn sum into max units and input distributions into max distributions
2. feedforward evaluation for $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
3. retrieve max activations in backward pass
4. compute **MAP states** for X_1 and X_3 at input units



Determinism + **decomposability** = **tractable MAP**

E.g., for $\operatorname{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

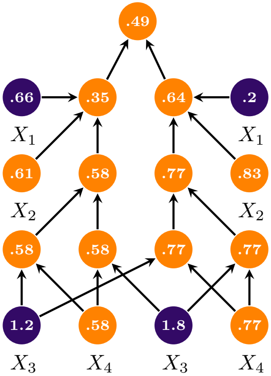
1. turn sum into max units and input distributions into max distributions
2. feedforward evaluation for $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
3. retrieve max activations in backward pass
4. compute **MAP states** for X_1 and X_3 at input units



Determinism + **decomposability** = **tractable MAP**

E.g., for $\operatorname{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

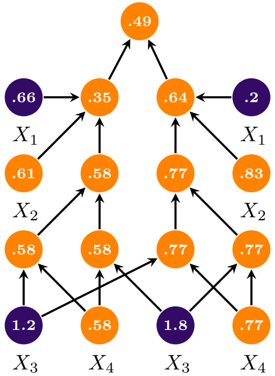
1. turn sum into max units and input distributions into max distributions
2. feedforward evaluation for $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
3. retrieve max activations in backward pass
4. compute **MAP states** for X_1 and X_3 at input units



Determinism + decomposability = tractable MAP

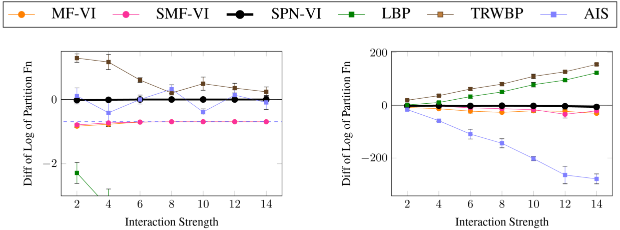
E.g., for $\text{argmax}_{x_1,x_3}\, p(x_1, x_3 \mid x_2, x_4)$:

1. turn sum into max units and input distributions into max distributions
2. feedforward evaluation for $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
3. retrieve max activations in backward pass
4. compute **MAP states** for X_1 and X_3 at input units



Example: Tractable ELBO

Using deterministic and decomposable PCs as expressive variational family $\mathcal{Q}$ for discrete polynomial log-densities, i.e. $\operatorname{argmax}_{q \in \mathcal{Q}} \mathbb{E}_{\mathbf{x} \sim q} [\log w(\mathbf{x})] + \mathbb{H}(q)$



Closed-form computation for the entropy $\mathbb{H}$ *(Liang et al. 2017)*

Shih and Ermon, "Probabilistic Circuits for Variational Inference in Discrete Graphical Models", 2020

Goal

Given a class of queries can we systematically find a class of probabilistic circuits that is tractable for it?

A language for queries

Integral expressions that can be formed by composing these operators

$+$, $\times$, **pow**, **log**, **exp** and $/$

$\Longrightarrow$ *many divergences and information-theoretic queries*

A language for queries

Integral expressions that can be formed by composing these operators

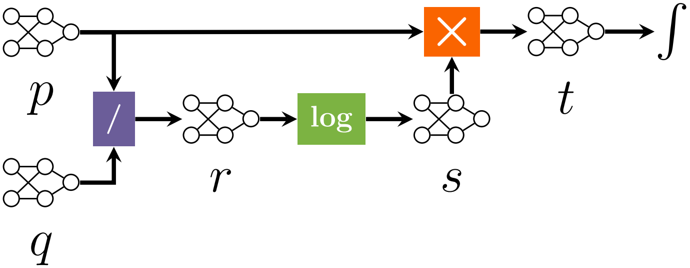
$+$, $\times$, pow, log, exp and $/$

$\Longrightarrow$ *many divergences and information-theoretic queries*

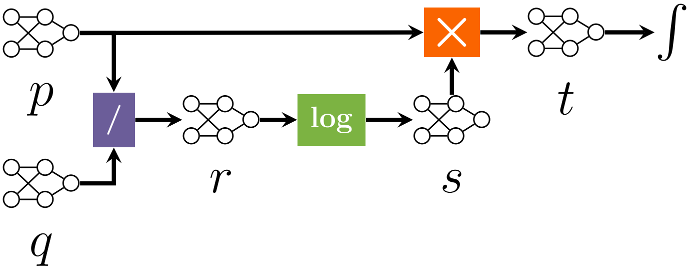
Represented as ***higher-order computational graphs***—pipelines—operating over circuits!

$\Longrightarrow$ *re-using intermediate transformations across queries*

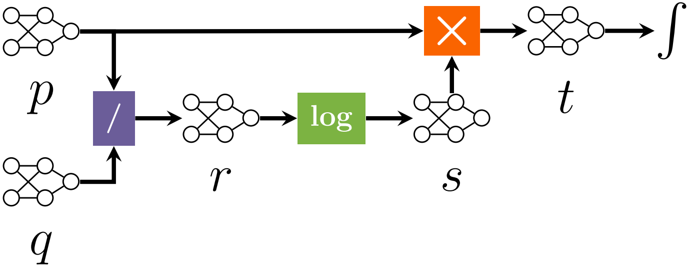
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log\left(p(\mathbf{x})/q(\mathbf{x})\right)\ d\mathbf{X}$$



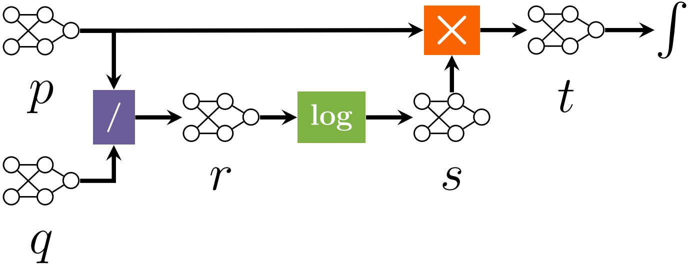
$$\mathbb{KLD}(p \,||\, q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \log \left(p(\mathbf{x}) \,\boxed{/}\, q(\mathbf{x}) \right) \, d\mathbf{X}$$



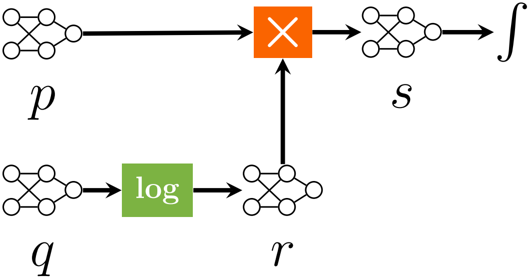
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times \boxed{\mathbf{log}} \, (p(\mathbf{x})/q(\mathbf{x})) \; d\mathbf{X}$$



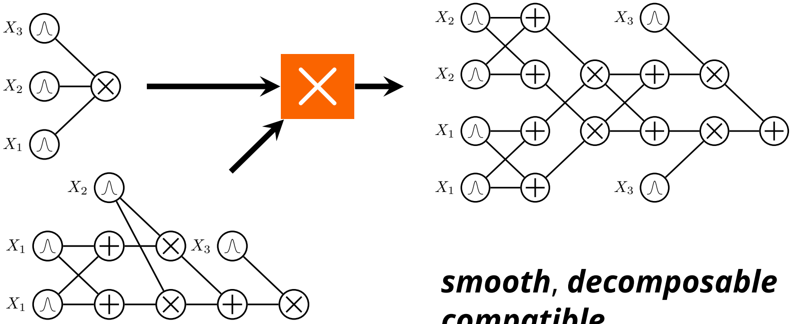
$$\mathbb{KLD}(p \mid\mid q) = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \boxed{\times} \log\left(p(\mathbf{x})/q(\mathbf{x})\right) \, d\mathbf{X}$$



$$\mathbb{XENT}(p \,||\, q) = \int p(\mathbf{x}) \times \log q(\mathbf{x}) \, d\mathbf{X}$$

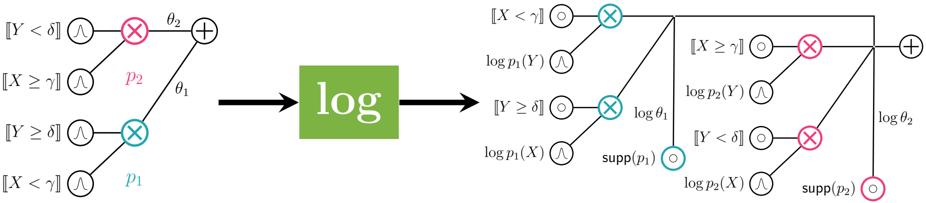


Tractable operators



smooth, decomposable compatible

Tractable operators

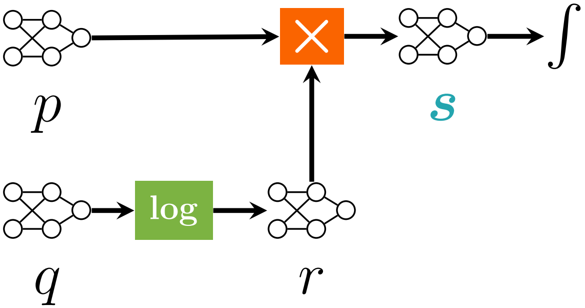


smooth, decomposable deterministic

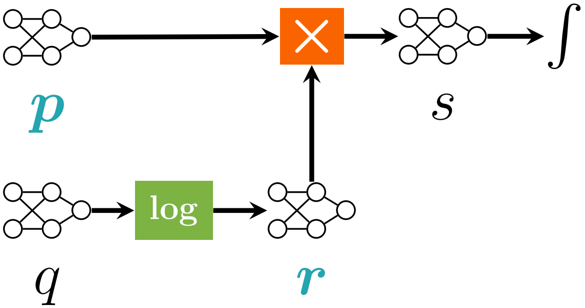
smooth, decomposable

		× $p \times q$	+ $p + q$	pow$_{\mathbb{N}}$ p^n	pow$_{\mathbb{R}}$ p^α	/ p/q	log $\log p$	exp $\exp p$
smoothness	SMO	✗	✓	✓	✓	✓	✓	✓
decomposability	DEC	✗	✗	✗	✓	✗	✗	✗
determinism	DET	✗	✗	✗	✓	✓	✓	✗
compatibility	CMP	✗	✓	✓	✗	✓	✗	✗

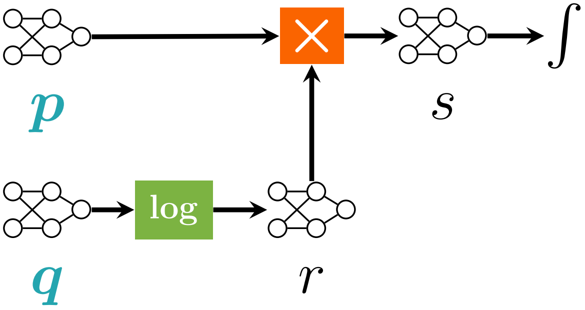
*Building an atlas of composable **tractable atomic operations***



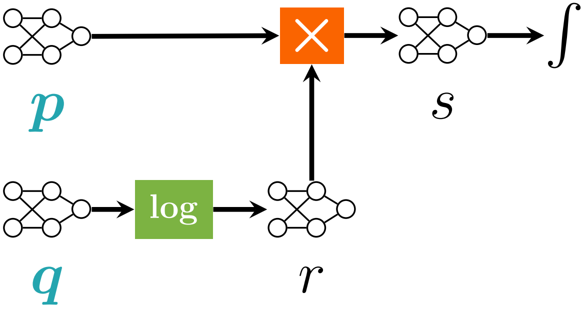
To perform tractable integration we need s to be ***smooth and decomposable...***



hence we need $\boldsymbol{p}$ and $\boldsymbol{r}$ to be smooth, decomposable and ***compatible***...



therefore $\boldsymbol{q}$ must be smooth, decomposable and ***deterministic***...



we can compute $\mathbb{XENT}$ tractably if $\boldsymbol{p}$ and $\boldsymbol{q}$ are smooth, decomposable, compatible and $\boldsymbol{q}$ is deterministic...

	Query	Tract. Conditions	Hardness
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}_+$	Sm, Dec, Det	#P-hard w/o Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x})) d\mathbf{X}$	Cmp, Det	#P-hard w/o Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp

***compositionally** derive the tractability of many more queries*

***Vergari** et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

	Query	Tract. Conditions	Hardness
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det	#P-hard w/o Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x},\boldsymbol{y}) \log(p(\boldsymbol{x},\boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x}))d\mathbf{X}$	Cmp, Det	#P-hard w/o Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x})q(\boldsymbol{x})d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x})d\mathbf{X} \int q^2(\boldsymbol{x})d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp

*and **prove hardness** when some input properties are not satisfied*

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries", 2021*

Composable tractable sub-routines

```
function kld(p, q)
    r = quotient(p, q)
    s = log(r)
    t = product(p, s)
    return integrate(t)
end
```

```
function xent(p, q)
    r = log(q)
    s = product(p, r)
    return -integrate(s)
end
```

```
function ent(p)
    q = log(p)
    r = product(p, q)
    return -integrate(s)
end
```

```
function alphadiv(p, q, alpha=1.5)
    r = real_pow(p, alpha)
    s = real_pow(q, 1.0-alpha)
    t = product(r, s)
    return log(integrate(t)) / (1.0-alpha)
end
```

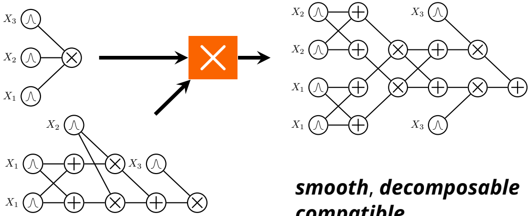
Efficient inference algorithms ***in a couple lines of Julia code!*** [1]

[1] https://github.com/UCLA-StarAI/circuit-ops-atlas

Next up...

1. Learning and reasoning with symbolic constraints
2. Expected predictions: handling missing values, fairness

$\Rightarrow$ *using tractable products*



smooth**, **decomposable
compatible

Symbolic constraints

"How can neural nets reason and learn with symbolic constraints reliably and efficiently?"

When?

Ground Truth

e.g. predict shortest path in a map

When?

Ground Truth

given $\mathbf{x}$ *// e.g. a tile map*

structured output prediction (SOP) tasks

Vlastelica et al., "Differentiation of blackbox combinatorial solvers", 2020

When?

Ground Truth

given $\mathbf{x}$ *// e.g. a tile map*

find $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})$ *// e.g. a configurations of edges in a grid*

structured output prediction (SOP) tasks

Vlastelica et al., "Differentiation of blackbox combinatorial solvers", 2020

When?

Ground Truth

given $\mathbf{x}$ *// e.g. a tile map*

find $\mathbf{y}^* = \mathrm{argmax}_{\mathbf{y}}\, p_\theta(\mathbf{y} \mid \mathbf{x})$ *// e.g. a configurations of edges in a grid*

s.t. $\mathbf{y} \models \mathsf{K}$ *// e.g., that form a valid path*

structured output prediction (SOP) tasks

Vlastelica et al., "Differentiation of blackbox combinatorial solvers", 2020

When?

Ground Truth

given $\mathbf{x}$ *// e.g. a tile map*

find $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})$ *// e.g. a configurations of edges in a grid*

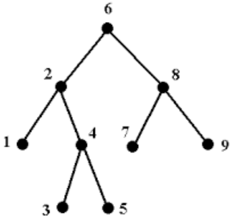
s.t. $\mathbf{y} \models \mathsf{K}$ *// e.g., that form a valid path*

// for a 12×12 grid, 2^{144} states but only 10^{10} valid ones!

structured output prediction (SOP) tasks

Vlastelica et al., "Differentiation of blackbox combinatorial solvers", 2020

When?



given $\mathbf{x}$ *// e.g. a feature map*

find $\mathbf{y}^* = \mathrm{argmax}_{\mathbf{y}}\, p_\theta(\mathbf{y} \mid \mathbf{x})$ *// e.g. labels of classes*

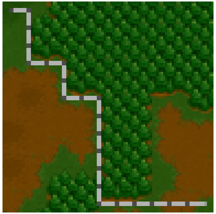
s.t. $\mathbf{y} \models \mathsf{K}$ *// e.g., constraints over superclasses*

$$\mathsf{K} : (Y_{\mathsf{cat}} \implies Y_{\mathsf{animal}}) \wedge (Y_{\mathsf{dog}} \implies Y_{\mathsf{animal}})$$

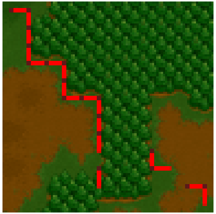
hierarchical multi-label classification

Giunchiglia and Lukasiewicz, "Coherent hierarchical multi-label classification networks", 2020

When?



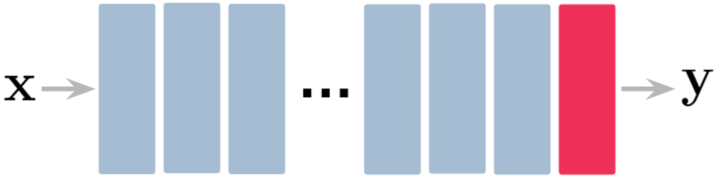
Ground Truth



ResNet-18

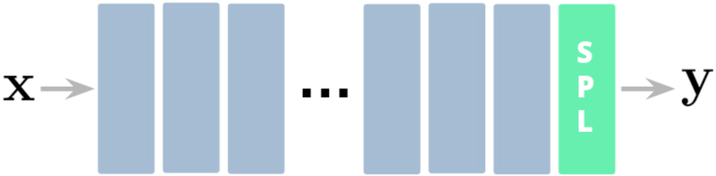
neural nets struggle to satisfy validity constraints!

How?



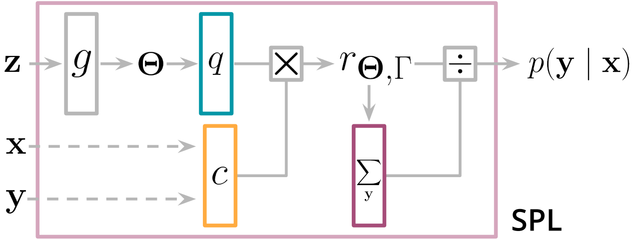
take an unreliable neural network architecture...

How?



......and replace the last layer with a semantic probabilistic layer

SPL

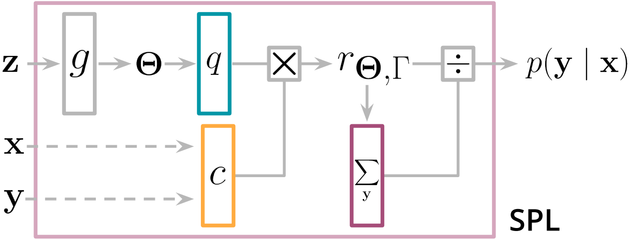


$q_{\boldsymbol{\Theta}}(\mathbf{y} \mid g(\mathbf{z}))$ is an expressive distribution over labels

$c_{\mathsf{K}}(\mathbf{x}, \mathbf{y})$ encodes the constraint $\mathbb{1}\{\mathbf{x}, \mathbf{y} \models \mathsf{K}\}$

Ahmed et al., "Semantic Probabilistic Layers for Neuro-Symbolic Learning", 2022

SPL

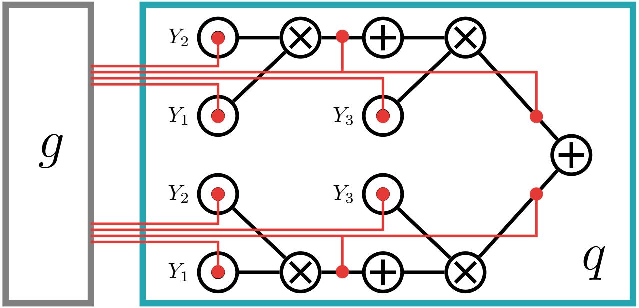


$$p(\mathbf{y} \mid \mathbf{x}) = q_{\boldsymbol{\Theta}}(\mathbf{y} \mid g(\mathbf{z})) \cdot c_{\mathsf{K}}(\mathbf{x}, \mathbf{y}) / \mathcal{Z}(\mathbf{x})$$

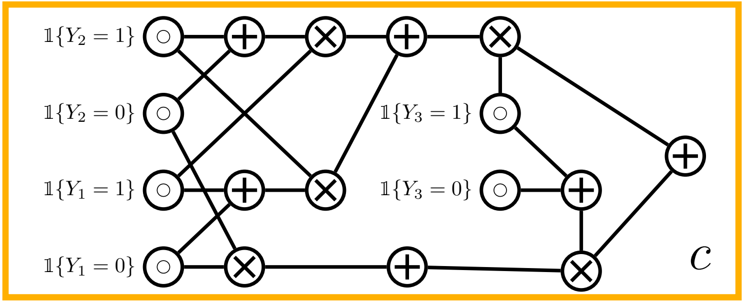
$$\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}} q_{\boldsymbol{\Theta}}(\mathbf{y} \mid \mathbf{x}) \cdot c_{\mathsf{K}}(\mathbf{x}, \mathbf{y})$$

Ahmed et al., "Semantic Probabilistic Layers for Neuro-Symbolic Learning", 2022

SPL

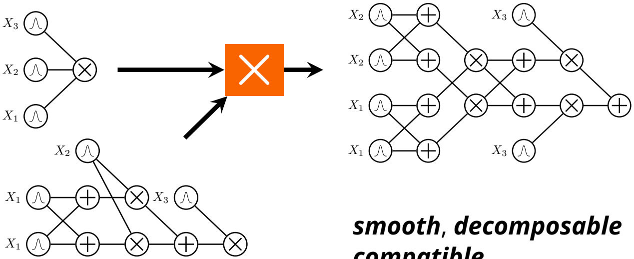


a conditional circuit $\boldsymbol{q}(\mathrm{y}; \boldsymbol{\Theta} = g(\mathbf{z}))$



and a logical circuit $c(\mathrm{y}, \mathrm{x})$ **encoding** K

Tractable products



smooth, decomposable compatible

exactly compute $\mathcal{Z}$ in time $O(|\boldsymbol{q}||\boldsymbol{c}|)$

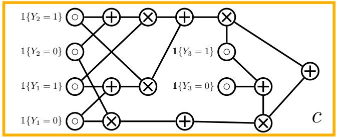
SPL recipe

$$\mathsf{K}: (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \quad (Y_2 = 1 \implies Y_3 = 1)$$

1) Take any
logical constraint

SPL recipe

$$
\begin{aligned}
\mathsf{K}: &\ (Y_1 = 1 \implies Y_3 = 1) \\
\wedge &\ (Y_2 = 1 \implies Y_3 = 1)
\end{aligned}
$$

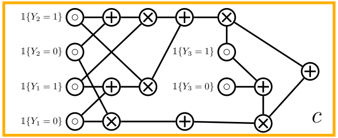


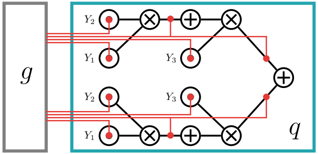
1) Take any logical constraint

2) Compile it into a constraint circuit

SPL recipe

$$\mathsf{K}: (Y_1 = 1 \implies Y_3 = 1)$$
$$\land \quad (Y_2 = 1 \implies Y_3 = 1)$$





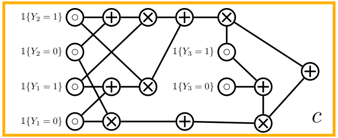
1) Take any logical constraint

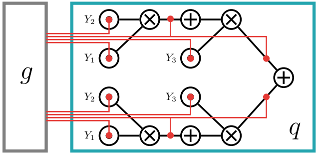
2) Compile it into a constraint circuit

3) Multiply it by a circuit distribution

SPL recipe

$$\mathsf{K}: (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \quad (Y_2 = 1 \implies Y_3 = 1)$$





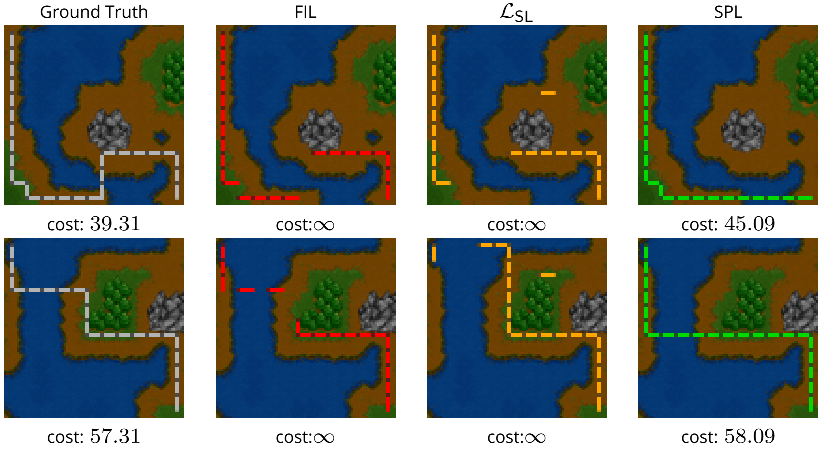
1) Take any
logical constraint

2) Compile it into
a constraint circuit

3) Multiply it
by a circuit distribution

4) train end-to-end by sgd!

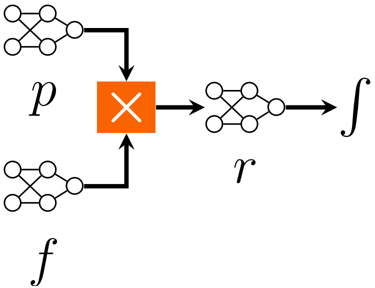
Guaranteeing consistency



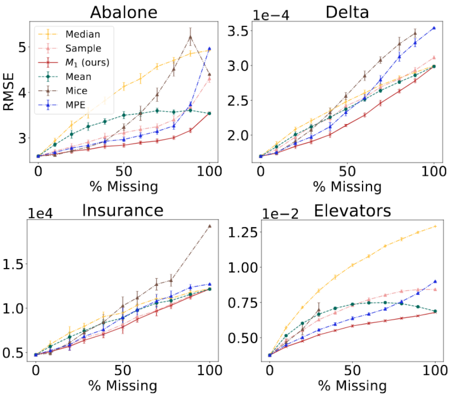
Expected predictions

Reasoning about the output of a classifier or regressor $\boldsymbol{f}$ given a distribution $\boldsymbol{p}$ over the input features

$$\mathbb{E}_p[f] = \int_{\mathsf{val}(\mathbf{X})} p(\mathbf{x}) \times f(\mathbf{x})\, d\mathbf{X}$$



Handling missing values at test time



Given a partial observation $\mathbf{x}^o$, what is the expected output from f?

$$\mathop{\mathbb{E}}_{\mathbf{x}^m \sim p(\mathbf{x}^m|\mathbf{x}^o)} \left[f(\mathbf{x}^m, \mathbf{x}^o) \right]$$

Khosravi et al., "On Tractable Computation of Expected Predictions", 2019

Fairness analysis

```julia
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);
```

q: *Is the predictive model biased by gender?*

```julia
groups = make_observations([["male"], ["female"]])
exps, _ = Expectation(pc, rc, groups);
println("Female  : \$ $(exps[2])");
println("Male    : \$ $(exps[1])");
println("Diff    : \$ $(exps[2] - exps[1])");
Female  : $ 14170.125469335406
Male    : $ 13196.548926381849
Diff    : $ 973.5765429535568
```

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

Conclusions

Probabilistic Circuits

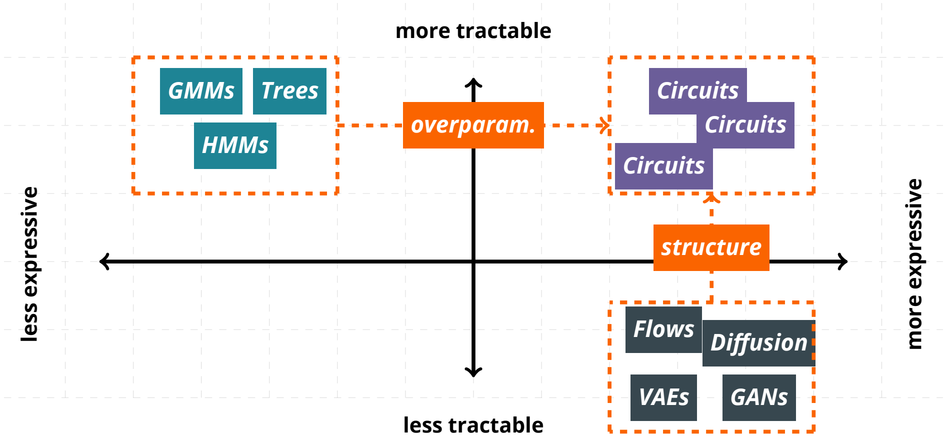
a grammar for structured tractable deep learning models

Building Circuits

imposing structure and learning parameters from data and prior knowledge

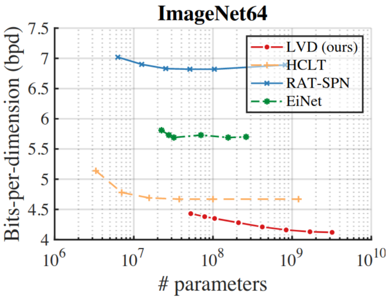
Advanced Reasoning

how do structure and reasoning interplay for real-world applications

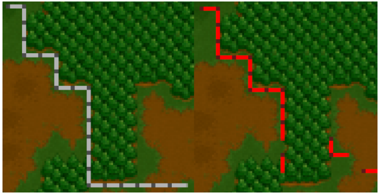
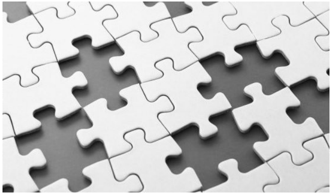
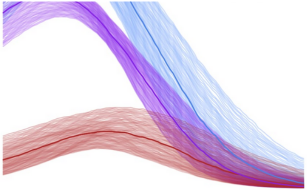


expressiveness and tractability without compromises

Dataset	TPMs				DGMs		
	LVD (ours)	HCLT	EiNet	RAT-SPN	Glow	RealNVP	BIVA
ImageNet32	**4.39**±0.01	4.82	5.63	6.90	4.09	4.28	3.96
ImageNet64	**4.12**±0.00	4.67	5.69	6.82	3.81	3.98	-
CIFAR	**4.38**±0.02	4.61	5.81	6.95	3.35	3.49	3.08



takeaway #2: we can learn circuits with billions of parameters

takeaway# 3: a unified framework for complex reasoning

		× $p \times q$	+ $p + q$	$\mathrm{pow}_{\mathbb{N}}$ p^n	$\mathrm{pow}_{\mathbb{R}}$ p^α	/ p/q	log $\log p$	exp $\exp p$
smoothness	SMO	✗	✓	✓	✓	✓	✓	✓
decomposability	DEC	✗	✗	✗	✓	✗	✗	✗
determinism	DET	✗	✗	✗	✓	✓	✓	✗
compatibility	CMP	✗	✓	✓	✗	✓	✗	✗

takeaway# 3.1: a compositional framework for reasoning

Challenge #1

scaling tractable learning

Learn tractable models
*on **billions of datapoints***
*and **thousands of features***
in tractable time!

Challenge #2

more structure!

Inject and enforce
symmetries and other
real-world biases!

Challenge #3

advanced and automated reasoning

Move beyond single reasoning tasks
*towards **fully automated reasoning**!*

Readings

Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models

`starai.cs.ucla.edu/papers/ProbCirc20.pdf`

Foundations of Sum-Product Networks for probabilistic modeling

`tinyurl.com/w65po5d`

Slides for this tutorial

`nolovedeeplearning.com/slides/pc-neurips22.pdf`

Code

Juice.jl advanced logical+probabilistic inference with circuits in Julia
github.com/Juice-jl/ProbabilisticCircuits.jl

SPFlow easy and extensible python library for SPNs
`github.com/SPFlow/SPFlow`

Are you looking for a PhD/Postdoc?

YooJung is hiring! yj.choi@asu.edu

yoojungchoi.github.io

Antonio is hiring! avergari@ed.ac.uk

nolovedeeplearning.com/buysellexchange.html

References I

- Darwiche, Adnan (2002). “A logical approach to factoring belief networks”. In: KR 2, pp. 409–420.
- Darwiche, Adnan and Pierre Marquis (2002). “A knowledge compilation map”. In: Journal of Artificial Intelligence Research 17, pp. 229–264.
- Darwiche, Adnan (2003). “A Differential Approach to Inference in Bayesian Networks”. In: J.ACM.
- — (2004). “New advances in compiling CNF to decomposable negation normal form”. In: Proc. of ECAI. Citeseer, pp. 328–332.
- — (2009). Modeling and Reasoning with Bayesian Networks. Cambridge.
- Poon, Hoifung and Pedro Domingos (2011). “Sum-Product Networks: a New Deep Architecture”. In: UAI 2011.
- Muise, Christian, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu (2012). “Dsharp: fast d-DNNF compilation with sharpSAT”. In: Canadian Conference on Artificial Intelligence. Springer, pp. 356–361.
- Gens, Robert and Pedro Domingos (2013). “Learning the Structure of Sum-Product Networks”. In: Proceedings of the ICML 2013, pp. 873–880.
- Rahman, Tahrima, Prasanna Kothalkar, and Vibhav Gogate (2014). “Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees”. In: Machine Learning and Knowledge Discovery in Databases. Vol. 8725. LNCS. Springer, pp. 630–645.
- Rooshenas, Amirmohammad and Daniel Lowd (2014). “Learning Sum-Product Networks with Direct and Indirect Variable Interactions”. In: Proceedings of ICML 2014.
- Adel, Tameem, David Balduzzi, and Ali Ghodsi (2015). “Learning the Structure of Sum-Product Networks via an SVD-based Algorithm”. In: Uncertainty in Artificial Intelligence.

References II

- Bova, Simone, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky (2015). "On compiling CNFs into structured deterministic DNNFs". In: International Conference on Theory and Applications of Satisfiability Testing. Springer, pp. 199–214.
- Dennis, Aaron and Dan Ventura (2015). "Greedy Structure Search for Sum-product Networks". In: IJCAI'15. Buenos Aires, Argentina: AAAI Press, pp. 932–938. isbn: 978-1-57735-738-4.
- Vergari, Antonio, Nicola Di Mauro, and Floriana Esposito (2015). "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning". In: ECML-PKDD 2015.
- Friesen, Abram L and Pedro Domingos (2016). "Submodular Sum-product Networks for Scene Understanding". In.
- Jaini, Priyank, Abdullah Rashwan, Han Zhao, Yue Liu, Ershad Banijamali, Zhitang Chen, and Pascal Poupart (2016). "Online Algorithms for Sum-Product Networks with Continuous Variables". In: Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings, pp. 228–239. url: `http://jmlr.org/proceedings/papers/v52/jaini16.html`.
- Peharz, Robert, Robert Gens, Franz Pernkopf, and Pedro M. Domingos (2016). "On the Latent Variable Interpretation in Sum-Product Networks". In: IEEE Transactions on Pattern Analysis and Machine Intelligence PP, Issue 99. url: `http://arxiv.org/abs/1601.06180`.
- Rashwan, Abdullah, Han Zhao, and Pascal Poupart (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.
- Yuan, Zehuan, Hao Wang, Limin Wang, Tong Lu, Shivakumara Palaiahnakote, and Chew Lim Tan (2016). "Modeling spatial layout for scene image understanding via a novel multiscale sum-product network". In: Expert Systems with Applications 63, pp. 231–240.

References III

- Zhao, Han, Tameem Adel, Geoff Gordon, and Brandon Amos (2016). "Collapsed Variational Inference for Sum-Product Networks". In: In Proceedings of the 33rd International Conference on Machine Learning. Vol. 48.
- Lagniez, Jean-Marie and Pierre Marquis (2017). "An Improved Decision-DNNF Compiler.". In: IJCAI. Vol. 17, pp. 667–673.
- Liang, Yitao and Guy Van den Broeck (Aug. 2017). "Towards Compact Interpretable Models: Shrinking of Learned Probabilistic Sentential Decision Diagrams". In: IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI). url: `http://starai.cs.ucla.edu/papers/LiangXAI17.pdf`.
- Butz, Cory J, Jhonatan S Oliveira, André E Santos, André L Teixeira, Pascal Poupart, and Agastya Kalra (2018). "An Empirical Study of Methods for SPN Learning and Inference". In: International Conference on Probabilistic Graphical Models, pp. 49–60.
- Di Mauro, Nicola, Floriana Esposito, Fabrizio Giuseppe Ventola, and Antonio Vergari (2018). "Sum-Product Network structure learning by efficient product nodes discovery". In: Intelligenza Artificiale 12.2, pp. 143–159.
- Jaini, Priyank, Amur Ghose, and Pascal Poupart (2018). "Prometheus: Directly Learning Acyclic Directed Graph Structures for Sum-Product Networks". In: International Conference on Probabilistic Graphical Models, pp. 181–192.
- Molina, Alejandro, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting (2018). "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains". In: AAAI.
- Oztok, Umut and Adnan Darwiche (2018). "An exhaustive DPLL algorithm for model counting". In: Journal of Artificial Intelligence Research 62, pp. 1–32.
- Khosravi, Pasha, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck (2019). "On Tractable Computation of Expected Predictions". In: Advances in Neural Information Processing Systems, pp. 11167–11178.

References IV

- Molina, Alejandro, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting (2019). "SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks". In: arXiv preprint arXiv:1901.03704.
- Peharz, Robert, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani (2019). "Random sum-product networks: A simple but effective approach to probabilistic deep learning". In: Proceedings of UAI.
- Salman, Hadi, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang (2019). "Provably robust deep learning via adversarially trained smoothed classifiers". In: Advances in Neural Information Processing Systems 32.
- Shao, Xiaoting, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting (2019). "Conditional Sum-Product Networks: Imposing Structure on Deep Probabilistic Architectures". In: arXiv preprint arXiv:1905.08550.
- Trapp, Martin, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani (2019). "Bayesian Learning of Sum-Product Networks". In: Advances in neural information processing systems (NeurIPS).
- Vergari, Antonio, Alejandro Molina, Robert Peharz, Zoubin Ghahramani, Kristian Kersting, and Isabel Valera (2019). "Automatic Bayesian density analysis". In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33, pp. 5207–5215.
- Correia, Alvaro, Robert Peharz, and Cassio P de Campos (2020). "Joints in random forests". In: Advances in neural information processing systems 33, pp. 11404–11415.
- Giunchiglia, Eleonora and Thomas Lukasiewicz (2020). "Coherent hierarchical multi-label classification networks". In: NeurIPS 33, pp. 9662–9673.
- Peharz, Robert, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani (2020). "Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits". In: ICML.

References V

- Shih, Andy and Stefano Ermon (2020). “Probabilistic Circuits for Variational Inference in Discrete Graphical Models”. In: NeurIPS.
- Vlastelica, Marin, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek (2020). “Differentiation of blackbox combinatorial solvers”. In: ICLR.
- Choi, YooJung, Antonio Vergari, and Guy Van den Broeck (2021). “Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling”. In: Technical Report.
- Subramani, Pranav Shankar, Gautam Kamath, Robert Peharz, et al. (2021). “Exact and Efficient Adversarial Robustness with Decomposable Neural Networks”. In: The 4th Workshop on Tractable Probabilistic Modeling.
- **Vergari**, Antonio, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck (2021). “A Compositional Atlas of Tractable Circuit Operations: From Simple Transformations to Complex Information-Theoretic Queries”. In: NeurIPS. arXiv: `2102.06137 [stat.ML]`.
- Yu, Zhongjie, Fabrizio G Ventola, and Kristian Kersting (2021). “Whittle Networks: A Deep Likelihood Model for Time Series”. In: Proceedings of the 38th International Conference on Machine Learning. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 12177–12186. url: `https://proceedings.mlr.press/v139/yu21c.html`.
- Ahmed, Kareem, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari (2022a). “Semantic Probabilistic Layers for Neuro-Symbolic Learning”. In: arXiv preprint arXiv:2206.00426.
- Ahmed, Kareem, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck (2022b). “Neuro-symbolic entropy regularization”. In: Uncertainty in Artificial Intelligence. PMLR, pp. 43–53.
- Dang, Meihua, Anji Liu, and Guy Van den Broeck (2022). “Sparse Probabilistic Circuits via Pruning and Growing”. In: NeurIPS. url: `http://starai.cs.ucla.edu/papers/DangNeurIPS22.pdf`.

References VI

- Liu, Anji, Honghua Zhang, and Guy Van den Broeck (2022). “Scaling Up Probabilistic Circuits by Latent Variable Distillation”. In: arXiv preprint.