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...!
neural networks are everywhere...!
Why?

struggle with uncertainty  be unfair  be not robust

**but they can...**
Why?

q1. “What is the probability of a treatment for a patient with unavailable records?”

q2. “How fair is the prediction is a certain protected attribute changes?”

q3. “Can we certify no adversarial examples exist?”

how can we reason about their behavior?
Why

“What is the probability of a treatment for a patient with unavailable records?”

“How fair is the prediction is a certain protected attribute changes?”

“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 \textit{deep} learning models that can reliably reason?”

\textit{expressive} and \textit{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?”
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

q1 “What is the probability of a treatment for a patient with unavailable records?”

q2 “How fair is the prediction is a certain protected attribute changes?”

q3 “Can we certify no adversarial examples exist?”
Reasoning about ML models

\[ q_1 \int p(x_o, x_m) dX_m \]  
(missing values)

\[ q_2 \mathbb{E}_{x_c \sim p(x_c|X_s=0)} [f_0(x_c)] - \mathbb{E}_{x_c \sim p(x_c|X_s=1)} [f_1(x_c)] \]  
(fairness)

\[ q_3 \mathbb{E}_{e \sim N(0, \sigma^2 I_D)} [f(x + e)] \]  
(adversarial robust)

...in the language of probabilities
Inspecting behaviors

$q_1 \int p(x_o, x_m) dX_m$

*(missing values)*

$q_2 \mathbb{E}_{x_c \sim p(x_c|x_s=0)} [f_0(x_c)] - \mathbb{E}_{x_c \sim p(x_c|x_s=1)} [f_1(x_c)]$

*(fairness)*

$q_3 \mathbb{E}_{e \sim \mathcal{N}(0, \sigma^2 I_D)} [f(x + e)]$

*(adversarial robust.)*

*it is crucial we compute them exactly and in polytime!*
Inspecting behaviors

$q_1 \int p(x_o, x_m) dX_m$

(missing values)

$q_2 \mathbb{E}_{x_c \sim p(x_c|X_s=0)} [f_0(x_c)] - \mathbb{E}_{x_c \sim p(x_c|X_s=1)} [f_1(x_c)]$

(fairness)

$q_3 \mathbb{E}_{e \sim \mathcal{N}(0, \sigma^2 I_D)} [f(x + e)]$

(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?
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

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

⇒ translating inference to data structures...
GMMs as computational graphs

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

⇒ ...e.g., as a weighted sum unit over Gaussian input distributions
GMMs

as computational graphs

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

\[
\Rightarrow \text{ inference } = \text{ feedforward evaluation}
\]
GMMs as computational graphs

A simplified notation:

- **scopes** attached to inputs
- edge directions omitted
GMMs
as computational graphs

\[
p(X) = w_1 \cdot p_1(X_1^L) \cdot p_1(X_1^R) + w_2 \cdot p_2(X_2^L) \cdot p_2(X_2^R)
\]

⇒ local factorizations...
GMMs as computational graphs

\[
p(X) = w_1 \cdot p_1(X^L_1) \cdot p_1(X^R_1) + w_2 \cdot p_2(X^L_2) \cdot p_2(X^R_2)
\]

⇒ ...are product units
HMMs

as computational graphs

\[ Z_1 \xrightarrow{\cdots} Z_2 \xrightarrow{\cdots} \cdots \]

\[ X_1 \xrightarrow{\cdots} X_2 \xrightarrow{\cdots} \cdots \\times \\times \times \]
Probabilistic Circuits (PCs)
A grammar for tractable computational graphs

1. 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

\[ \text{\textbf{I. A simple tractable function is a circuit}} \]
\[ \text{\textbf{II. A weighted combination of circuits is a circuit}} \]
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, stddev=1.0, scope=1) *
                   Categorical(p=[0.4, 0.6], scope=2))
           + 0.7 * (Gaussian(mean=-1.0, stddev=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, stddev=0.1, scope=1) *
          Categorical(p=[0.4, 0.6], scope=2))

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

\[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
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...

...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?

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

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

competitive with Flows and VAEs!
How scalable?

up to billions of parameters

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!

⇒ structural properties needed for tractability
1. A grammar for tractable models
One formalism to represent many models. #GMMs #HMMs #Trees #XGBoost, ...

2. Increase expressiveness
Stacking millions latent variables. #hierarchical #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

Structural properties

**smoothness**

**decomposability**

**compatibility**

**determinism**

the inputs of sum units are defined over the same variables

---

**Structural properties**

- smoothness
- decomposability
- compatibility
- determinism

The inputs of prod units are defined over disjoint variable sets.

\[ X_1 \times \cdots \times X_3 \]

*decomposable circuit*  
*non-decomposable circuit*

---

smooth + decomposable circuits = ...

allow for the tractable computation of arbitrary integrals

\[
p(y) = \int_{\text{val}(Z)} p(z, y) \, dZ, \quad \forall Y \subseteq X, \quad Z = X \setminus Y
\]

⇒ sufficient and necessary conditions for a single feedforward evaluation

⇒ can marginalize out any missing values

Probabilistic queries = feedforward evaluation

\[ p(X_1 = -1.85, X_4 = 0.2) \]
Probabilistic queries = feedforward evaluation

\[ p(X_1 = -1.85, X_4 = 0.2) \]
Computing arbitrary integrations (or summations)

$\Rightarrow$ linear in circuit size!

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

$$\int p(x) dx$$
**smooth** + **decomposable** circuits = ...

If \( p(x) = \sum_i w_i p_i(x) \), (**smoothness**):

\[
\int p(x) \, dx = \int \sum_i w_i p_i(x) \, dx = \sum_i w_i \int p_i(x) \, dx
\]

⇒ *integrals are “pushed down” to inputs*
smooth + decomposable circuits = ...

If \( p(x, y, z) = p(x)p(y)p(z) \), (decomposability):

\[
\int \int \int p(x, y, z) \, dx \, dy \, dz = \\
= \int \int \int p(x)p(y)p(z) \, dx \, dy \, dz = \\
= \int p(x) \, dx \int p(y) \, dy \int p(z) \, dz \\
\Rightarrow \text{integrals decompose into easier ones}
\]
Analogously, for arbitrary conditional queries:

\[ p(q \mid e) = \frac{p(q, e)}{p(e)} \]

1. evaluate \( p(q, e) \) \( \Rightarrow \) one feedforward pass
2. evaluate \( p(e) \) \( \Rightarrow \) another feedforward pass
   \( \Rightarrow \) ...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 $x$ by smoothing it by computing

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

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

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

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

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

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

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

$\Rightarrow$ expectations are “pushed down” to inputs
If \( f(x, y, z) = f(x)f(y)f(z) \), (decomposability):

\[
\int N(e_x)N(e_y)N(e_y)f(x + e_x, y + e_y, z + e_z)de_xde_yde_z
\]

\[
E_{ex}[f(x + e_x)] \cdot E_{ey}[f(y + e_y)] \cdot E_{ez}[f(z + e_z)]
\]

\( \Rightarrow \) 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:

$$\int p(x)f(x)dX$$
Integrals involving two or more functions:

\[ \int p(x) f(x) dX \]

represent both \( p \) and \( f \) as circuits...but with which structural properties? E.g.,
General expectations

Integrals involving two or more functions:

\[
\int p(x) f(x) \, dX
\]

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

\[
\mathbb{E}_{x_c \sim p(x_c | x_s = 0)} [f_0(x_c)] - \mathbb{E}_{x_c \sim p(x_c | x_s = 1)} [f_1(x_c)]
\]
Structural properties

- smoothness
- decomposability
- compatibility
- determinism

**Structural properties**

**smoothness**

**decomposability**

**compatibility**

**determinism**

---

Structural properties

- smoothness
- decomposability
- compatibility
- determinism

Tractable products

\[ \int p(x) f(x) \, dX \] in time \( O(|p| |f|) \)

Which structural properties
for complex reasoning

smooth + decomposable  smooth + compatible  decomposable
Structural properties

smoothness
decomposability
compatibility
determinism

stay tuned!

Which structural properties for complex reasoning

reason with constraints  expected predictions  computing uncertainties
structure + expressiveness
Building Probabilistic Circuits
Information  Prior Knowledge  Data

domain assumptions
constraints
other models

compilation

learning

Circuits  Structure  Parameters

decomposability
smoothness
determinism
compatibility

$\theta, \omega$
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)
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 incredibly fast:

```julia
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:

```python
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
```python
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

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

---

Peharz et al., “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”, 2020
Bayesian parameter learning

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

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

- 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 ($R$) and partitions ($P$)
- Input and output nodes of the RG are regions
- Regions have a scope (receptive field), denoted as $sc(R) \subseteq X$
- For every partition $P$ it holds that

$$sc(R_{out}) = \bigcup_{R_{in} \in inputs(P)} sc(R_{in})$$

$$sc(R') \cap sc(R'') = \emptyset, \quad R' \neq R'' \in inputs(P)$$
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

Often, output region has only a single sum
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
 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 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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LVD (ours)</th>
<th>HCLT</th>
<th>EiNet</th>
<th>RAT-SPN</th>
<th>Glow</th>
<th>RealNVP</th>
<th>BIVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet32</td>
<td>4.38</td>
<td>4.82</td>
<td>5.63</td>
<td>6.90</td>
<td>4.09</td>
<td>4.28</td>
<td>3.96</td>
</tr>
<tr>
<td>ImageNet64</td>
<td>4.12</td>
<td>4.67</td>
<td>5.69</td>
<td>6.82</td>
<td>3.81</td>
<td>3.98</td>
<td>-</td>
</tr>
<tr>
<td>CIFAR</td>
<td>4.37</td>
<td>4.61</td>
<td>5.81</td>
<td>6.95</td>
<td>3.35</td>
<td>3.49</td>
<td>3.08</td>
</tr>
</tbody>
</table>

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 \( X \) 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

(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

(Peharz et al. 2020)
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

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

will be Lipschitz, guaranteeing the non-existence of adversarial examples within a certain $\ell_2$-ball around $x$ (depending on the class margin and $\sigma$).

How to compute $\bar{f}(x)$ for neural networks? *Monte Carlo* seems to only solution, which is never exact and requires *many evaluations for a single 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.
Data-driven structure learning

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

Expand regions with **clustering**

\[ X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \]
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”

Success $\rightarrow$ 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)

\[ X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \]
Data-driven structure learning

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

Success $\rightarrow$ \textit{partition} into new regions

\[
\begin{array}{cccc}
X_1 & X_2 & X_3 & X_4 & X_5 \\
\end{array}
\]

\[\text{Diagram showing partitioning of data into regions.}\]
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 → input region
Data-driven structure learning

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

Expand regions with clustering
Data-driven structure learning

“Recursive data slicing”

Number of clusters = number of partitions

And so on...
Data-driven structure learning

“Recursive data slicing”

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

(Gens et al. 2013)
LearnSPN

Selected references

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

Circuits
- decomposability
- smoothness
- determinism
- compatibility

Structure

Parameters
- $\theta$, $w$
- generative
- discriminative
- Bayesian
- credal

Compilation

Learning
Advanced Reasoning with Probabilistic Circuits
Reasoning about ML models

q1. “What is the probability of a treatment for a patient with unavailable records?”

\[ \int p(x_o, x_m) \, dx_m \]

q2. “How fair is the prediction is a certain protected attribute changes?”

\[ \mathbb{E}_{x_c \sim p(x_c | x_s = 0)} [f_0(x_c)] - \mathbb{E}_{x_c \sim p(x_c | x_s = 1)} [f_1(x_c)] \]

q3. “Can we certify no adversarial examples exist?”

\[ \mathbb{E}_{e \sim \mathcal{N}(0, \sigma^2 I_D)} [f(x + e)] \]
**Maximum-a-posteriori (MAP) Inference**

*aka Most probable explanation (MPE)*

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

$$\arg\max_y p(y \mid x)$$

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

---


Determinism
aka support-decomposability

A sum unit is deterministic if its inputs have disjoint supports

\[ X_1 \leq \theta \quad X_2 \quad X_1 > \theta \quad X_2 \]

**deterministic circuit**

**non-deterministic circuit**

Darwiche and Marquis, “A knowledge compilation map”, 2002
Determinism + decomposability = tractable MAP

Computing maximization with arbitrary evidence $e$ ⇒ *linear in circuit size!*

E.g., suppose we want to compute:

$$\max_{q} p(q \mid e)$$
Determinism + decomposability = tractable MAP

If \( p(q, e) = \sum_i w_i p_i(q, e) = \max_i w_i p_i(q, e) \),
(deterministic sum unit):

\[
\max_q p(q, e) = \max_q \sum_i w_i p_i(q, e) \\
= \max_q \max_i w_i p_i(q, e) \\
= \max_i \max_q w_i p_i(q, e)
\]

\( \Rightarrow \) one non-zero term, thus sum is max
Determinism + decomposability = tractable MAP

If \( p(q, e) = p(q_x, e_x, q_y, e_y) = p(q_x, e_x)p(q_y, e_y) \)
(decomposable product unit):

\[
\max_q p(q \mid e) &= \max_q p(q, e) \\
&= \max_{q_x, q_y} p(q_x, e_x, q_y, e_y) \\
&= \max_{q_x} p(q_x, e_x) \cdot \max_{q_y} p(q_y, e_y)
\]

\( \Rightarrow \) 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 \( \text{max}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4) \)
3. retrieve max activations in backward pass
4. compute \( \text{MAP states} \) 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
   \( \text{max}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4) \)
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 \( \text{max}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4) \)
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 \( \arg\max_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4) \):
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 $Q$ for discrete polynomial log-densities, i.e. 

$$\arg\max_{q \in Q} \mathbb{E}_{x \sim q} \left[ \log w(x) \right] + H(q)$$

Closed-form computation for the entropy $H$ (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?
Integral expressions that can be formed by composing these operators:

\[ +, \times, \text{pow}, \log, \exp \text{ and } / \]

\[ \Rightarrow \text{many divergences and information-theoretic queries} \]
A language for queries

Integral expressions that can be formed by composing these operators:

\[ +, \times, \text{pow}, \log, \exp \text{ and } / \]

\[ \Rightarrow \text{many divergences and information-theoretic queries} \]

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

\[ \Rightarrow \text{re-using intermediate transformations across queries} \]
\[ \text{KLD}(p \parallel q) = \int_{\text{val}(X)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX \]
\[ \text{KLD}(p \parallel q) = \int_{\text{val}(x)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX \]
$$\text{KLD}(p \parallel q) = \int_{\text{val}(x)} p(x) \times \log (p(x)/q(x)) \, dX$$
\( \text{KLD}(p \parallel q) = \int_{\text{val}(x)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX \)
\[ \text{XENT}(p \ || \ q) = \int p(x) \times \log q(x) \, dX \]
Tractable operators

smooth, decomposable compatible
Tractable operators

\[
\begin{align*}
[Y < \delta] & \land [X \geq \gamma] & \land [Y \geq \delta] & \land [X < \gamma] \\
\theta_2 & \land p_2 & \theta_1 & \land p_1
\end{align*}
\times
\begin{align*}
\log p_1(X) & \land [X < \gamma] & \land \log \theta_1 & \land [Y \geq \delta] & \land \log p_1(Y) & \land [X \geq \gamma] & \land \log \theta_2 & \land [Y < \delta] & \land \log p_2(X) & \land [X < \gamma] & \land \log \theta_2 & \land [Y \geq \delta] & \land \log p_2(Y)
\end{align*}
\times
\begin{align*}
supp(p_1) & \land \log \theta_1 & \land [Y \geq \delta] & \land \log \theta_1 & \land \log p_1(X) & \land \log \theta_1 & \land \log p_1(Y) & \land \log \theta_1 & \land \log p_1(X) & \land \log \theta_1 & \land \log p_1(Y)
\end{align*}
\times
\begin{align*}
supp(p_2) & \land \log \theta_2 & \land [Y \geq \delta] & \land \log \theta_2 & \land \log p_2(X) & \land \log \theta_2 & \land \log p_2(Y) & \land \log \theta_2 & \land \log p_2(X) & \land \log \theta_2 & \land \log p_2(Y)
\end{align*}

smooth, decomposable deterministic

smooth, decomposable deterministic
Building an atlas of composable \textit{tractable atomic operations}
To perform tractable integration we need \( s \) to be *smooth and decomposable*...
hence we need \( p \) and \( r \) to be smooth, decomposable and compatible...
therefore $q$ must be smooth, decomposable and *deterministic*...
we can compute $\text{XENT}$ tractably if $p$ and $q$ are smooth, decomposable, compatible and $q$ is deterministic...
<table>
<thead>
<tr>
<th>Query</th>
<th>Tract. Conditions</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSS ENTROPY</td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>SHANNON ENTROPY</td>
<td>Sm, Dec, Det</td>
<td>coNP-hard w/o Det</td>
</tr>
<tr>
<td>RéNYI ENTROPY</td>
<td>SD</td>
<td>#P-hard w/o SD</td>
</tr>
<tr>
<td>MUTUAL INFORMATION</td>
<td>Sm, SD, Det*</td>
<td>coNP-hard w/o SD</td>
</tr>
<tr>
<td>KULLBACK-LEIBLER DIV.</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>RéNYI’S ALPHA DIV.</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>ITAKURA-SAITO DIV.</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>CAUCHY-SCHWARZ DIV.</td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
<tr>
<td>SQUARED LOSS</td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
</tbody>
</table>

compositionally derive the tractability of many more queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Tract. Conditions</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSS ENTROPY</td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>SHANNON ENTROPY</td>
<td>Sm, Dec, Det</td>
<td>coNP-hard w/o Det</td>
</tr>
<tr>
<td>RéNYI ENTROPY</td>
<td>SD</td>
<td>#P-hard w/o SD</td>
</tr>
<tr>
<td>MUTUAL INFORMATION</td>
<td>Sm, SD, Det*</td>
<td>coNP-hard w/o SD</td>
</tr>
<tr>
<td>KULLBACK-LEIBLER DIV.</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>RéNYI’S ALPHA DIV.</td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>ITAKURA-SAITO DIV.</td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td>CAUCHY-SCHWARZ DIV.</td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
<tr>
<td>SQUARED LOSS</td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
</tbody>
</table>

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

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

¹https://github.com/UCLA-StarAI/circuit-ops-atlas
Next up...

1. Learning and reasoning with symbolic constraints

2. Expected predictions: handling missing values, fairness

⇒ 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?

Given $x$ // e.g. a tile map

Ground Truth

**structured output prediction (SOP) tasks**

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

given $x$ // e.g. a tile map

find $y^* = \arg\max_y p_\theta(y \mid x)$ // e.g. a configurations of edges in a grid

structured output prediction (SOP) tasks

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

Given $x$ \(\text{// e.g. a tile map}\)

find $y^* = \arg\max_y p_\theta(y \mid x)$ \(\text{// e.g. a configurations of edges in a grid}\)

s.t. $y \models K$ \(\text{// e.g., that form a valid path}\)

**structured output prediction (SOP) tasks**

---

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

given $x$ // e.g. a tile map
find $y^* = \arg\max_y p_\theta(y \mid x)$ // e.g. a configurations of edges in a grid
s.t. $y \models K$ // e.g., that form a valid path

// for a $12 \times 12$ 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 $\mathbf{y}^* = \arg\max_y p_\theta(\mathbf{y} \mid \mathbf{x})$ // e.g. labels of classes

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

$$K : (Y_{\text{cat}} \implies Y_{\text{animal}}) \land (Y_{\text{dog}} \implies Y_{\text{animal}})$$

**hierarchical multi-label classification**

Giunchiglia and Lukasiewicz, “Coherent hierarchical multi-label classification networks”, 2020
When?

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
\( q_\Theta(y \mid g(z)) \) is an expressive distribution over labels

\[ c_K(x, y) \text{ encodes the constraint } \mathbb{1} \{ x, y \models K \} \]

\[ p(y \mid x) = q_{\Theta}(y \mid g(z)) \cdot c_{K}(x, y) / \mathcal{Z}(x) \]

\[ \mathcal{Z}(x) = \sum_{y} q_{\Theta}(y \mid x) \cdot c_{K}(x, y) \]

a conditional circuit $q(y; \Theta = g(z))$
and a logical circuit $c(y, x)$ encoding $K$
Tractable products

smooth, decomposable, compatible

exactly compute $Z$ in time $O(\|q\|\|c\|)$
SPL recipe

\[ K : (Y_1 = 1 \implies Y_3 = 1) \]
\[ \land (Y_2 = 1 \implies Y_3 = 1) \]

1) Take any logical constraint
**SPL recipe**

\[ K : (Y_1 = 1 \implies Y_3 = 1) \]
\[ \land \ (Y_2 = 1 \implies Y_3 = 1) \]

1) Take any logical constraint

2) Compile it into a constraint circuit
**SPL recipe**

\[ K : (Y_1 = 1 \implies Y_3 = 1) \land (Y_2 = 1 \implies Y_3 = 1) \]

1) Take any logical constraint

2) Compile it into a constraint circuit

3) Multiply it by a circuit distribution

\[ \begin{array}{c}
1 \{Y_1 = 0\} \\
1 \{Y_1 = 1\} \\
1 \{Y_2 = 0\} \\
1 \{Y_2 = 1\} \\
\times \\
\times \\
\times \\
\times \\
1 \{Y_3 = 1\} \\
1 \{Y_3 = 0\} \\
\times c \\
\end{array} \]
SPL recipe

K : \((Y_1 = 1 \implies Y_3 = 1) \land (Y_2 = 1 \implies Y_3 = 1)\)

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

Ground Truth

FIL

cost: 39.31

cost: ∞

cost: ∞

cost: 45.09

L_{SL}

cost: ∞

cost: ∞

cost: ∞

cost: 58.09

SPL

cost: 57.31

cost: ∞

cost: ∞

cost: 45.09
Expected predictions

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

$$\mathbb{E}_p[f] = \int_{\text{val}(\mathbf{X})} p(\mathbf{x}) \times f(\mathbf{x}) \, d\mathbf{X}$$
Handling missing values at test time

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

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

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?

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
takeaway #2: we can learn circuits with billions of parameters
takeaway# 3: a unified framework for complex reasoning
### Takeaway #3.1: A Compositional Framework for Reasoning

<table>
<thead>
<tr>
<th></th>
<th>$p + q$</th>
<th>$p \times q$</th>
<th>$p^n$</th>
<th>$p^a$</th>
<th>$p/q$</th>
<th>$\log p$</th>
<th>$\exp p$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SMO</strong></td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td><strong>DET</strong></td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

**Properties:**
- **Smoothness (SMO):** For $p + q, p \times q, p^n, p^a, p/q, \log p, \exp p$
- **Decomposability (DEC):** For $p + q, p \times q, p^n, p^a, p/q, \log p, \exp p$
- **Determinism (DET):** For $p + q, p \times q, p^n, p^a, p/q, \log p, \exp p$
- **Compatibility (CMP):** For $p + q, p \times q, p^n, p^a, p/q, \log p, \exp p$

**Note:** The table indicates whether each property holds or not for each operation.
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
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


Darwiche, Adnan (2003). “A Differential Approach to Inference in Bayesian Networks”. In: J.ACM.


References II


References III


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 Rolínek (2020). “Differentiation of blackbox combinatorial solvers”. In: ICLR.


