



# ***tractable probabilistic modeling with probabilistic circuits***

**antonio vergari** (he/him)

 @nolovedeeplearning

*20th June 2025 - Nordic Prob AI Trondheim*

# *thanks to...*



Lorenzo Loconte  
*U of Edinburgh*



Antonio Mari  
*EPFL*



Rickey Liang  
*Saarland U*



Gennaro Gala  
*TU Eindhoven*



Adrian Javaloy  
*U of Edinburgh*

# *and moar...*

*april*

april-tools.github.io

# *april*

*autonomous &  
provably  
reliable  
intelligent  
learners*

# *april*

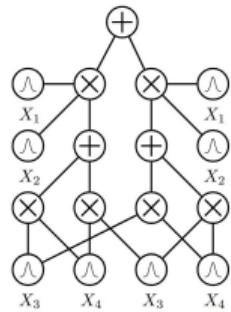
*about  
probabilities  
integrals &  
logic*

# *april*

*april is  
probably a  
recursive  
identifier of a  
lab*

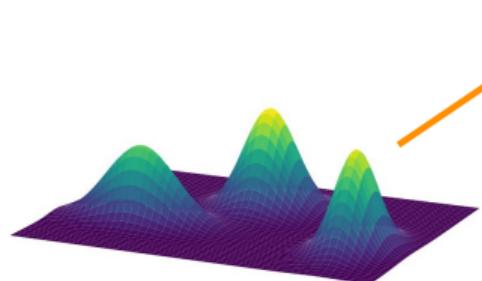
*today's topics...*

# *a language of tractable computational graphs*

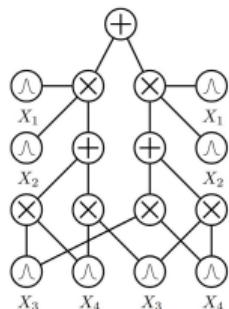


(deep)  
circuits

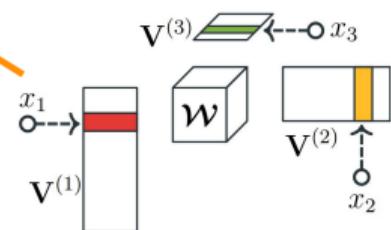
# *a single formalism for many models*



(hierarchical)  
mixtures



(deep)  
circuits



(hierarchical)  
tensor factorizations

*wait...!*

*why tractability?*

# *reasoning about ML models*



q<sub>1</sub>

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



q<sub>2</sub>

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



q<sub>3</sub>

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

# *reasoning about ML models*



**q<sub>1</sub>**  $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$   
*(missing values)*

**q<sub>2</sub>**  $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$   
*(fairness)*

**q<sub>3</sub>**  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

*...in the language of probabilities*

## ***more complex reasoning***



*neuro-symbolic AI*



*probabilistic programming*



*computing      uncertainties  
(Bayesian inference)*

***...and more application scenarios***

# *reasoning about ML models*



**q<sub>1</sub>**  $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$   
*(missing values)*

**q<sub>2</sub>**  $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$   
*(fairness)*

**q<sub>3</sub>**  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

***hard to compute in general!***

# *reasoning about ML models*



**q<sub>1</sub>**  $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$   
*(missing values)*

**q<sub>2</sub>**  $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$   
*(fairness)*

**q<sub>3</sub>**  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

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

# *reasoning about ML models*



**q<sub>1</sub>**  $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$   
*(missing values)*

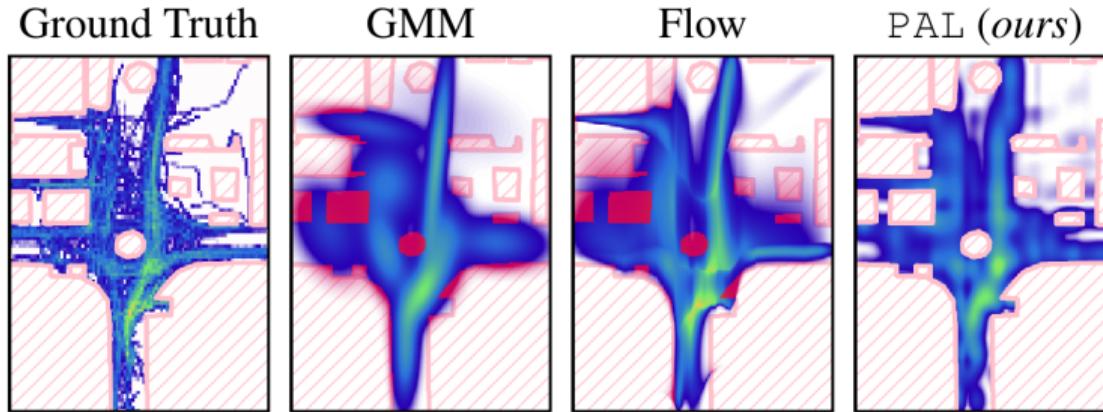
**q<sub>2</sub>**  $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$   
*(fairness)*

**q<sub>3</sub>**  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

***it is crucial we compute them tractably!***

# *why tractable models?*

*exactness can be crucial in **safety-driven** applications*

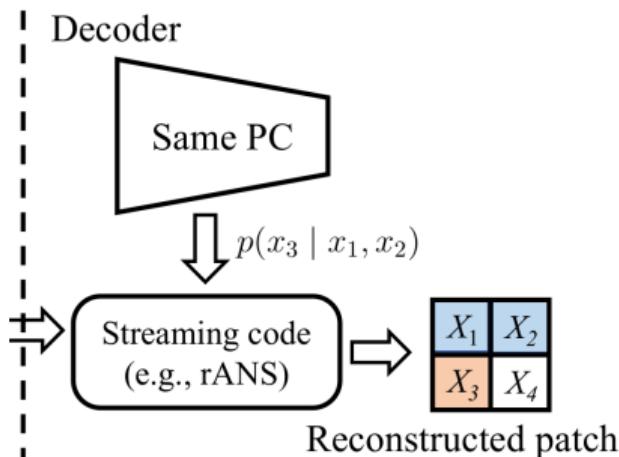


guarantee the satisfaction of given constraints

[Ahmed et al. 2022; Kurscheidt et al. 2025]

# why tractable models?

they can be much **faster** than intractable ones!



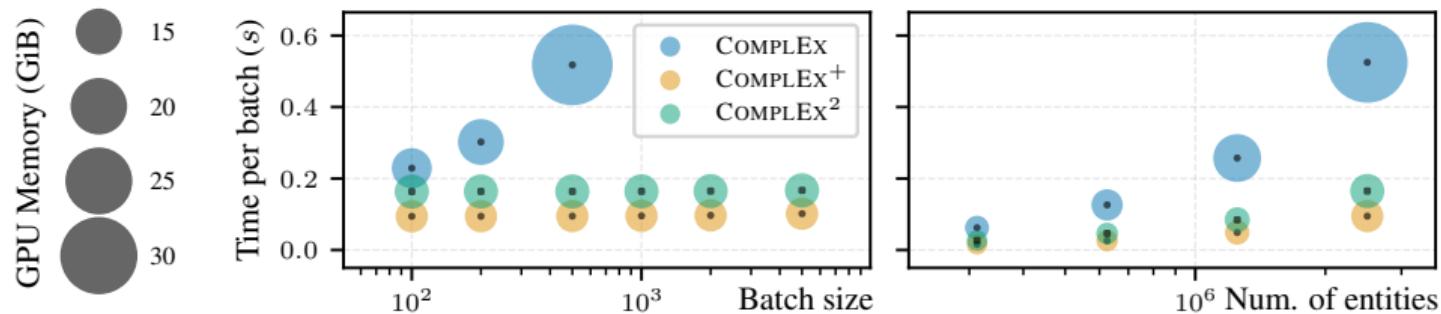
query many times the same model to compute conditional probabilities

Method	MNIST (10,000 test images)		
	Theoretical bpd	Comp. bpd	En- & decoding time
PC (small)	1.26	1.30	<b>53</b>
PC (large)	<b>1.20</b>	<b>1.24</b>	168
IDF	1.90	1.96	880
BitSwap	1.27	1.31	904

# ***why tractable models?***

*they can be much **faster** than intractable ones!*

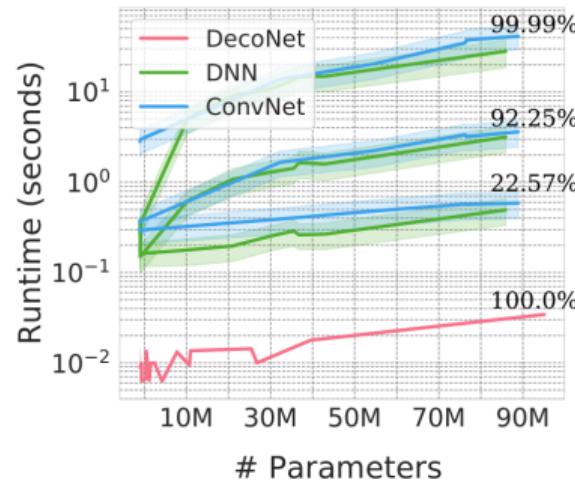
useful to scale to graph datasets with millions of entities



# why tractable models?

they can be much **faster** than intractable ones!

one-shot computation of an **exact expectation** versus Monte Carlo estimates



q3  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

**why?**

***tractable computations?***

**why?**

*we always perform  
**tractable computations**  
in the end,  
even when performing approximations!*

$$\min_{\mathbf{q} \in \mathcal{Q}} \text{KL}(\mathbf{q} || p)$$

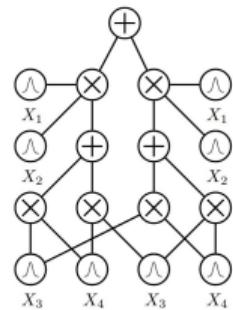
we have to pick a **tractable** variational distribution  $\mathbf{q}$ ...otherwise we cannot compute its density, mode, marginals later

$\Rightarrow$  e.g., Gaussian, GMM, HMM, normalizing flow, etc

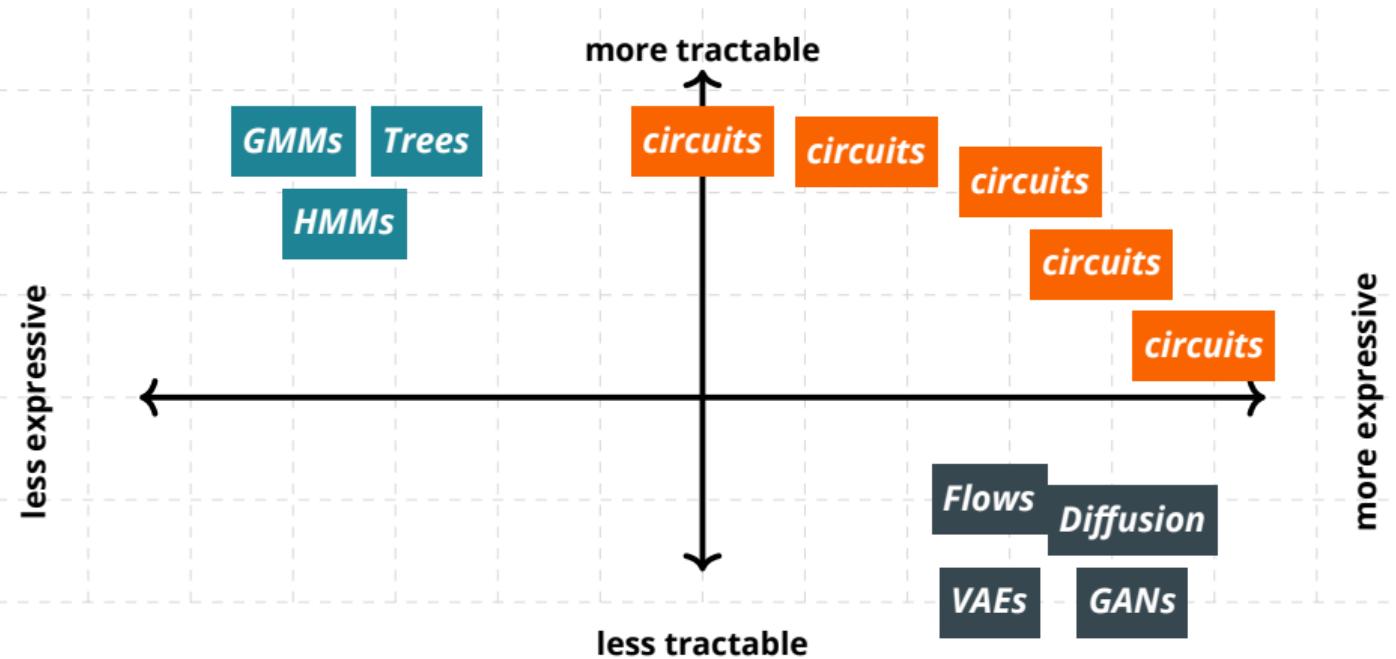
$\Rightarrow$  then, what is the largest tractable class we can use???

## *Goal*

*“Can we find  
a **middle ground**  
**between**  
**tractability and expressiveness?**”*

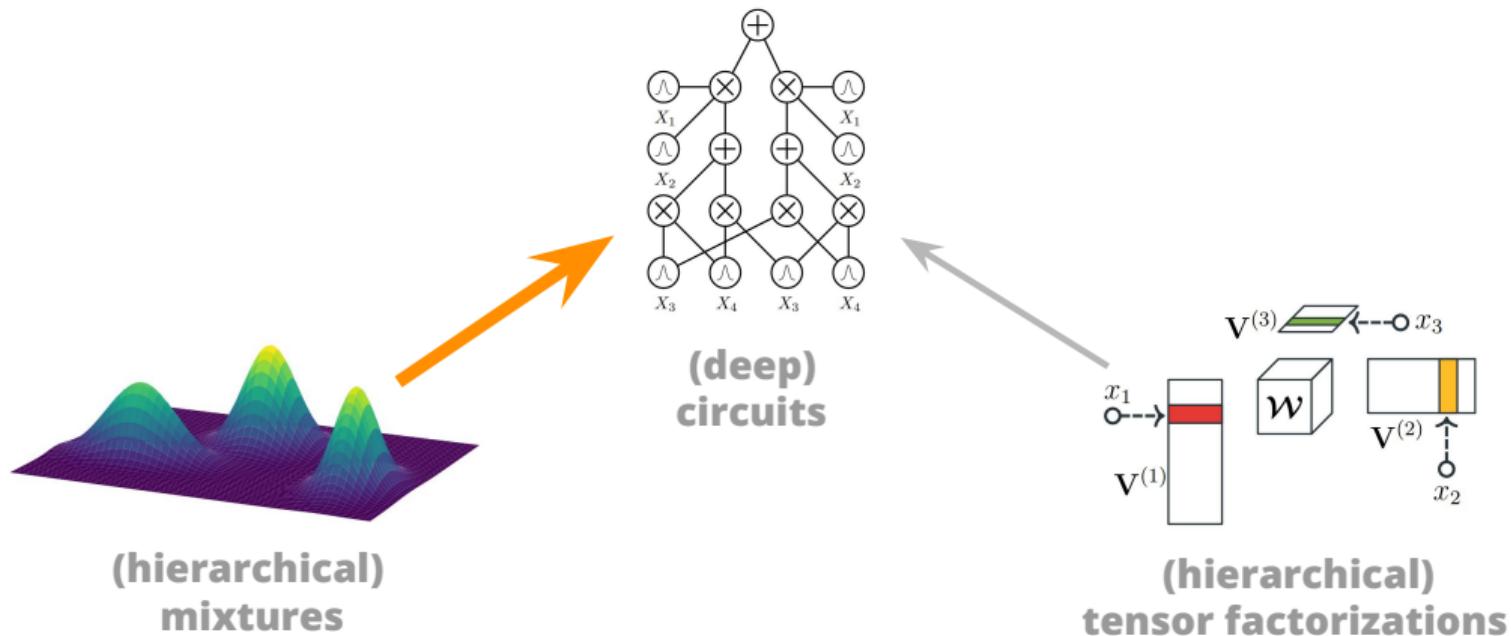


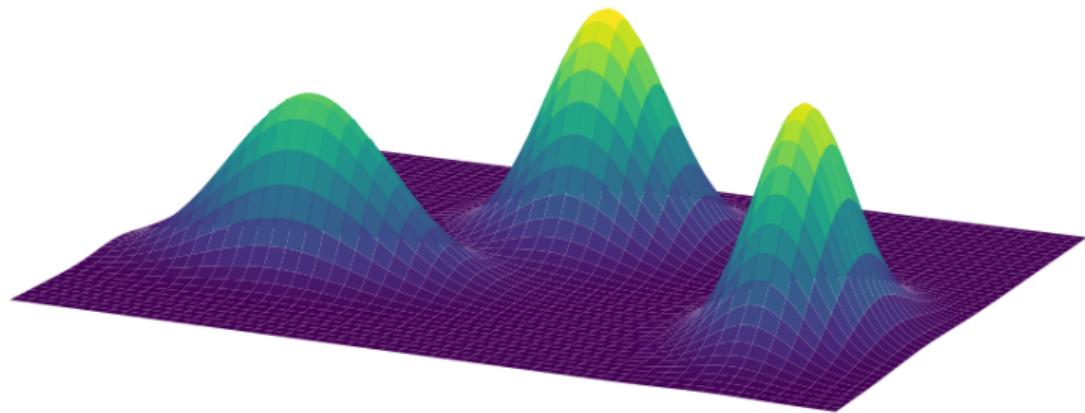
(deep)  
circuits



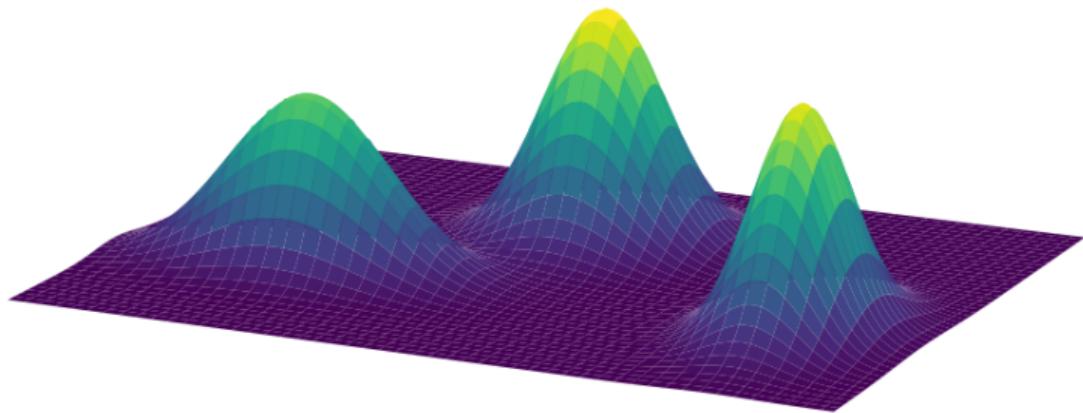
*navigate the trade-off  
between tractability and expressiveness*

*compile mixtures into circuits...*





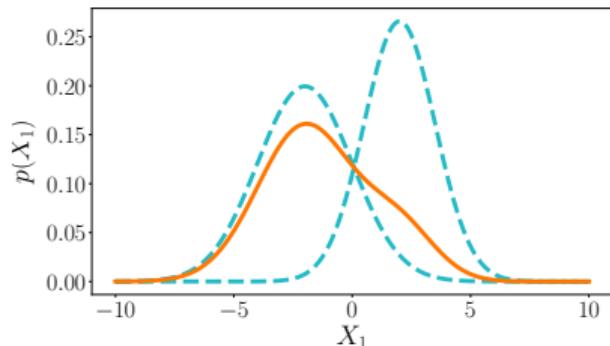
***who knows mixture models?***



***who loves mixture models?***

# GMMS

as computational graphs

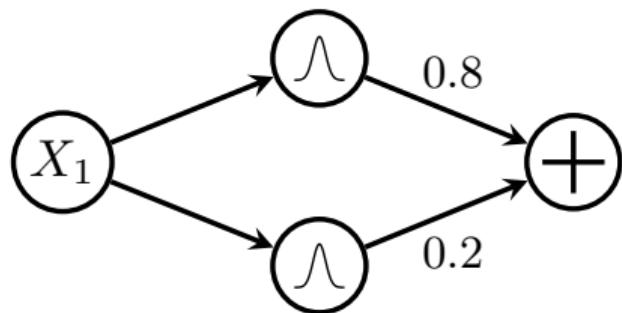


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

⇒ translating inference to data structures...

# GMMs

as computational graphs

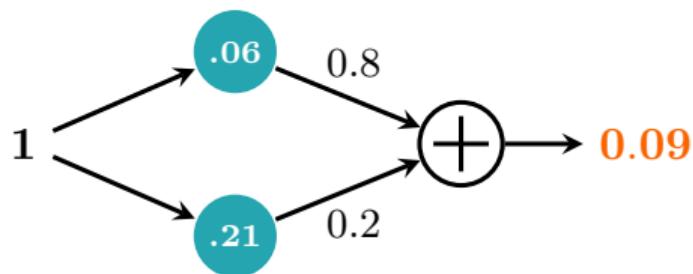


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

# GMMS

as computational graphs

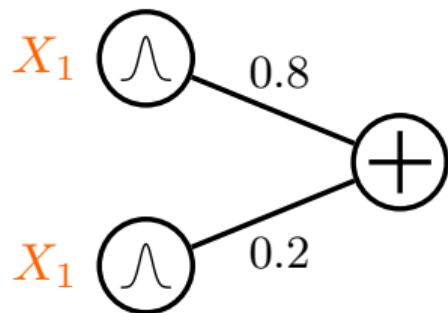


$$p(X = 1) = 0.2 \cdot p_1(X_1 = 1) + 0.8 \cdot p_2(X_1 = 1)$$

⇒ inference = feedforward evaluation

# GMMs

as computational graphs

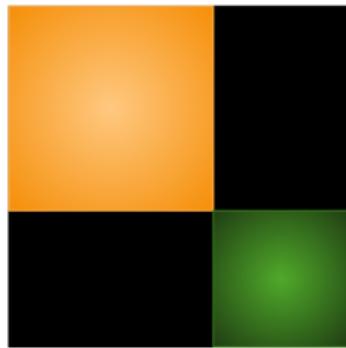
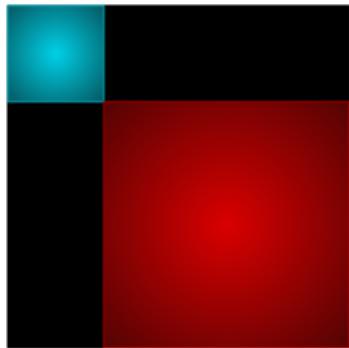


A simplified notation:

- ⇒ **scopes** attached to inputs
- ⇒ edge directions omitted

# GMMS

as computational graphs

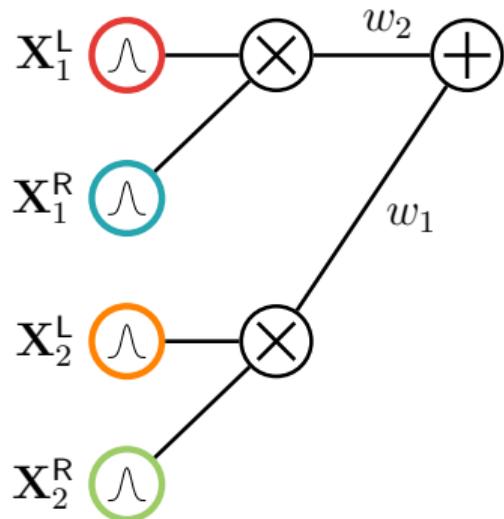


$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}_1^L) \cdot p_1(\mathbf{X}_1^R) + \\ w_2 \cdot p_2(\mathbf{X}_2^L) \cdot p_2(\mathbf{X}_2^R)$$

⇒ local factorizations...

# GMMs

as computational graphs



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

⇒ ...are product units

# **probabilistic circuits (PCs)**

*a grammar for tractable computational graphs*

I. A simple tractable function is a circuit

⇒ e.g., a multivariate Gaussian or small  
neural network



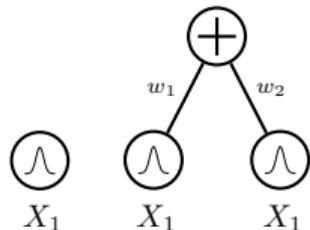
$X_1$

# **probabilistic circuits (PCs)**

*a grammar for tractable computational graphs*

I. A simple tractable function is a circuit

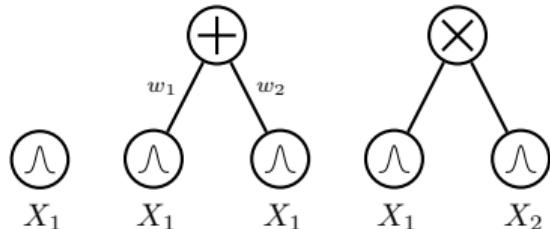
II. A weighted combination of circuits is a circuit



# **probabilistic circuits (PCs)**

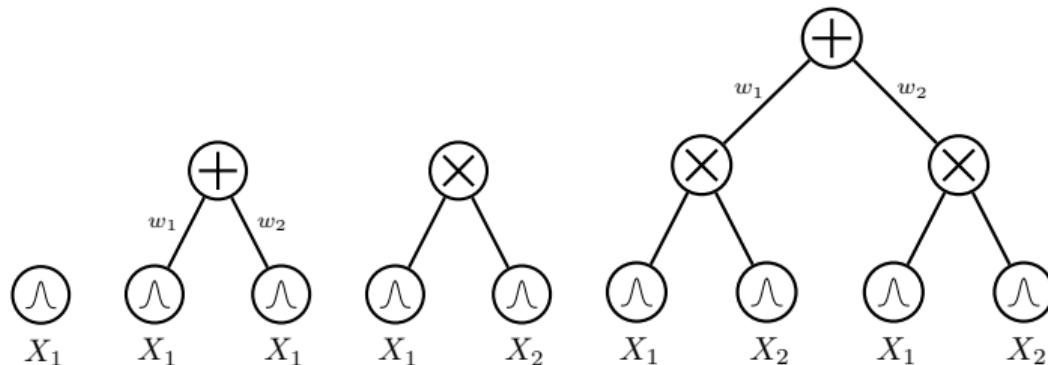
*a grammar for tractable computational graphs*

- 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



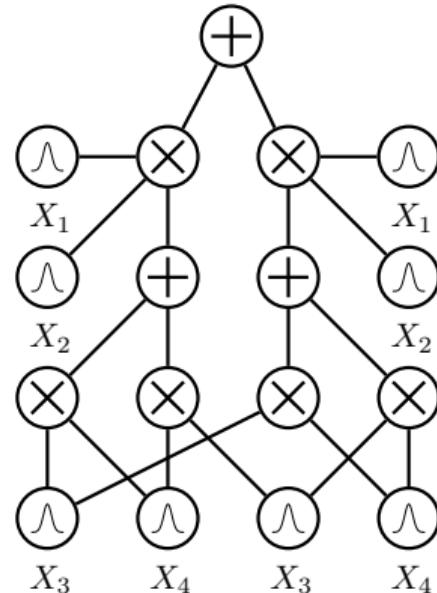
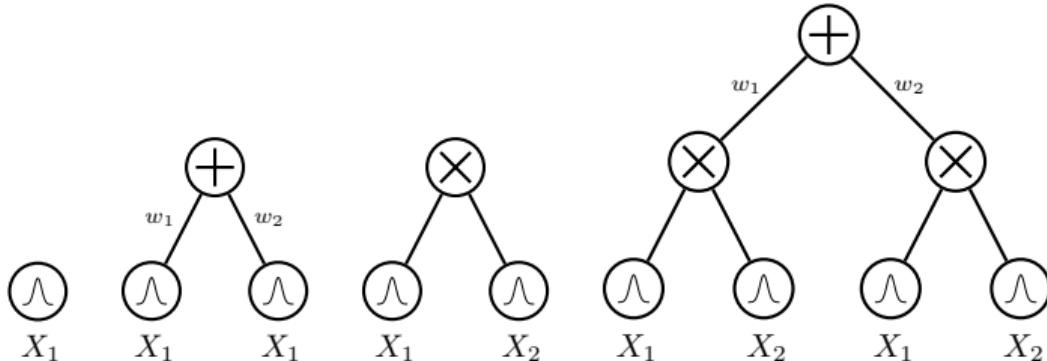
# **probabilistic circuits (PCs)**

*a grammar for tractable computational graphs*



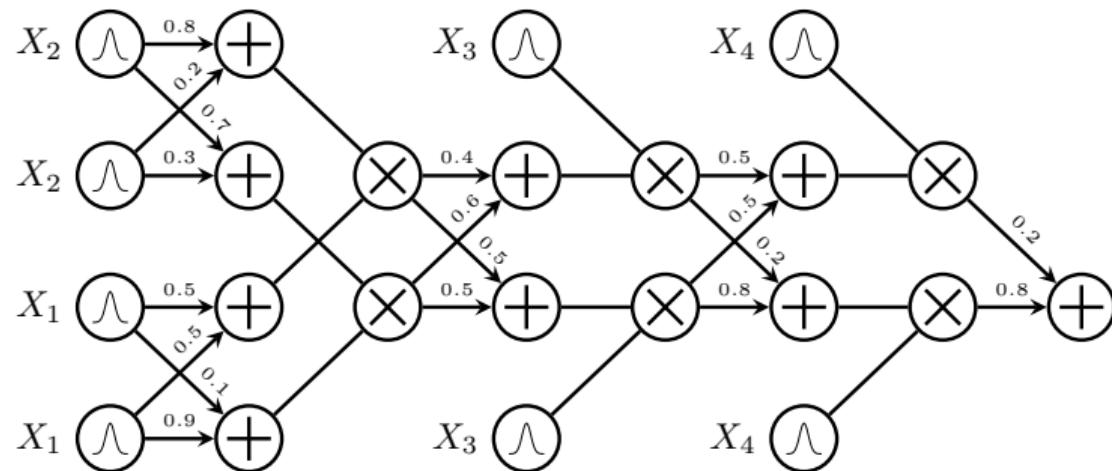
# **probabilistic circuits (PCs)**

*a grammar for tractable computational graphs*



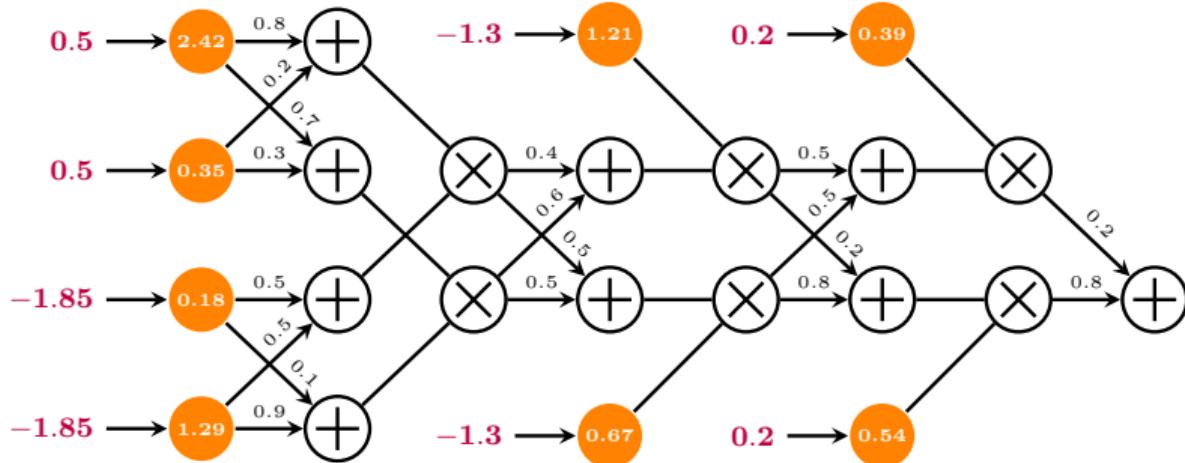
## **probabilistic queries** = **feedforward** evaluation

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



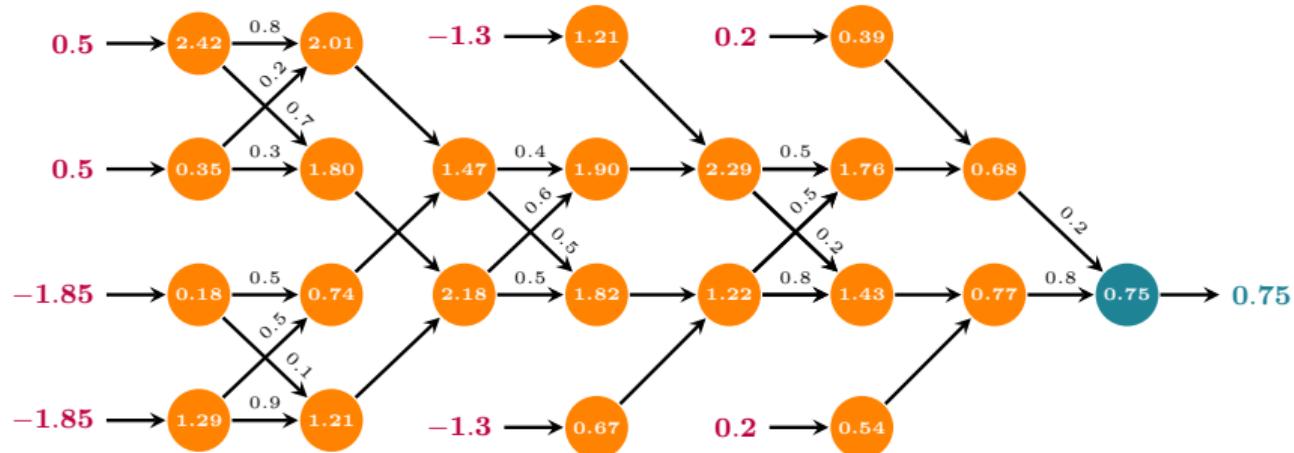
## **probabilistic queries** = **feedforward** evaluation

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



## **probabilistic queries** = **feedforward** evaluation

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



# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer



# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer

$$c(\mathbf{x}) = \mathbf{W}l(\mathbf{x})$$



# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer

$$c(\mathbf{x}) = \mathbf{W}l(\mathbf{x})$$



# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer
- III. The product of two layers is a circuit layer

$$c(\mathbf{x}) = \mathbf{l}(\mathbf{x}) \odot \mathbf{r}(\mathbf{x}) \quad // \text{ Hadamard}$$



# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer
- III. The product of two layers is a circuit layer

$$c(\mathbf{x}) = \mathbf{l}(\mathbf{x}) \odot \mathbf{r}(\mathbf{x}) \quad // \text{ Hadamard}$$

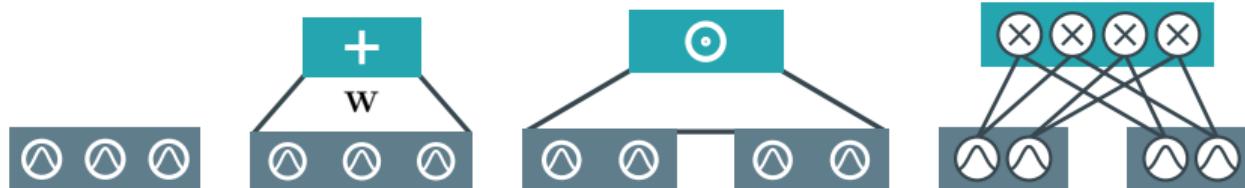


# **probabilistic circuits (PCs)**

*a tensorized definition*

- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer
- III. The product of two layers is a circuit layer

$$c(\mathbf{x}) = \text{vec}(\mathbf{l}(\mathbf{x})\mathbf{r}(\mathbf{x})^\top) \quad // \text{ Kronecker}$$

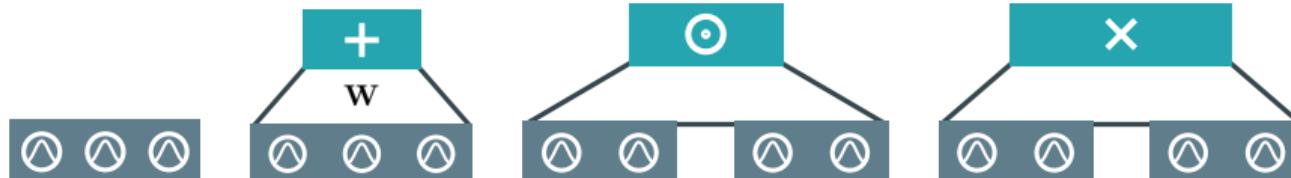


# **probabilistic circuits (PCs)**

*a tensorized definition*

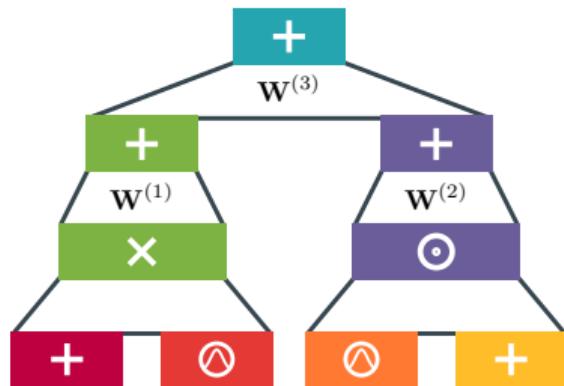
- I. A set of tractable functions is a circuit layer
- II. A linear projection of a layer is a circuit layer
- III. The product of two layers is a circuit layer

$$c(\mathbf{x}) = \text{vec}(\mathbf{l}(\mathbf{x})\mathbf{r}(\mathbf{x})^\top) \quad // \text{ Kronecker}$$



# **probabilistic circuits (PCs)**

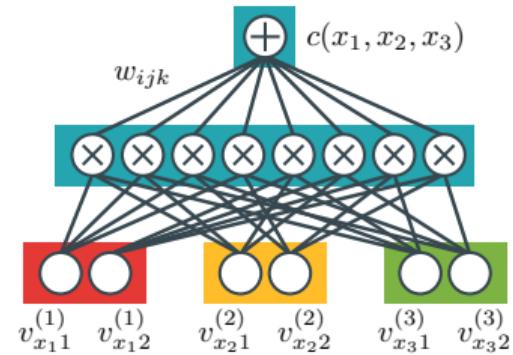
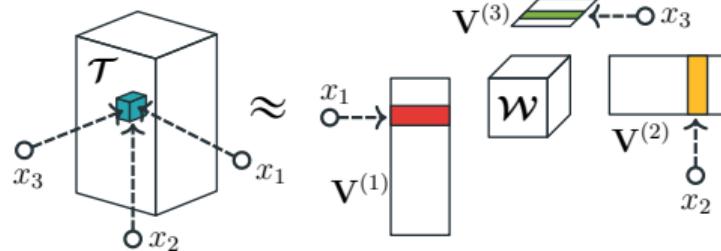
*a tensorized definition*



- I. A set of tractable functions is a circuit layer
  - II. A linear projection of a layer is a circuit layer
  - III. The product of two layers is a circuit layer
- stack layers to build a deep circuit!**

# *tensor factorizations*

*as circuits*



***wait...!***

***how do we learn them?***

# **which parameters?**

*how to reparameterize circuits*

Input distributions.

Sum unit parameters.

# **which parameters?**

*how to reparameterize circuits*

**Input distributions.** Each input can be a different parametric distribution

⇒ *Bernoullis, Categoricals, Gaussians, exponential families, small NNs, ...*

**Sum unit parameters.**

# which parameters?

how to reparameterize circuits

**Input distributions.** Each input can be a different parametric distribution

**Sum unit parameters.** Enforce them to be non-negative, i.e.,  $w_i \geq 0$  but unnormalized

$$w_i = \exp(\alpha_i), \quad \alpha_i \in \mathbb{R}, \quad i = 1, \dots, K$$

and renormalize the loss

$$\min_{\theta} - \left( \sum_{i=1}^N \log \tilde{p}_{\theta}(\mathbf{x}^{(i)}) - \log \int \tilde{p}_{\theta}(\mathbf{x}^{(i)}) d\mathbf{X} \right)$$

or just renormalize the weights, i.e.,  $\sum_i w_i = 1$

$$\mathbf{w} = \text{softmax}(\boldsymbol{\alpha}), \quad \boldsymbol{\alpha} \in \mathbb{R}^K$$

***wait...!***

***how do we learn them?***

***wait...!***

***how do we **learn them?*****

***just SGD your way as usual!***

⇒ ***or any other gradient-based optimizer***



***learning & reasoning with circuits in pytorch***

[github.com/april-tools/cirkit](https://github.com/april-tools/cirkit)

The screenshot shows a GitHub repository page for 'april-tools / cirkit'. The 'Code' tab is selected. A specific file, 'cirkit / notebooks / learning-a-gaussian-mixture-model.ipynb', is open. The notebook has a commit by 'adrianjav' titled 'Add CP diagram' made 20 hours ago. The notebook content starts with a section titled 'Learning a Gaussian Mixture Model' which describes the purpose of the notebook.

```
In this notebook, we show how we can create a symbolic circuit with cirkit to create a simple Gaussian mixture model, compile it into a regular Pytorch model, and learn the cluster assignments using Adam.  
Note that this is an illustrative example to show how to build symbolic circuits manually, and there are better ways of fitting Gaussian mixture models than with stochastic first-order optimization.
```

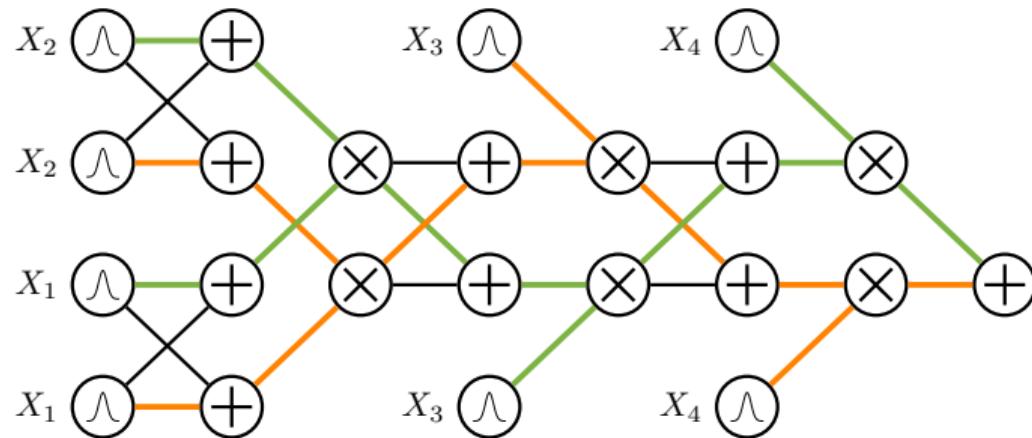


## *a notebook on learning GMMs as circuits*

<https://github.com/april-tools/cirkit/blob/main/notebooks/learning-a-gaussian-mixture-model.ipynb>

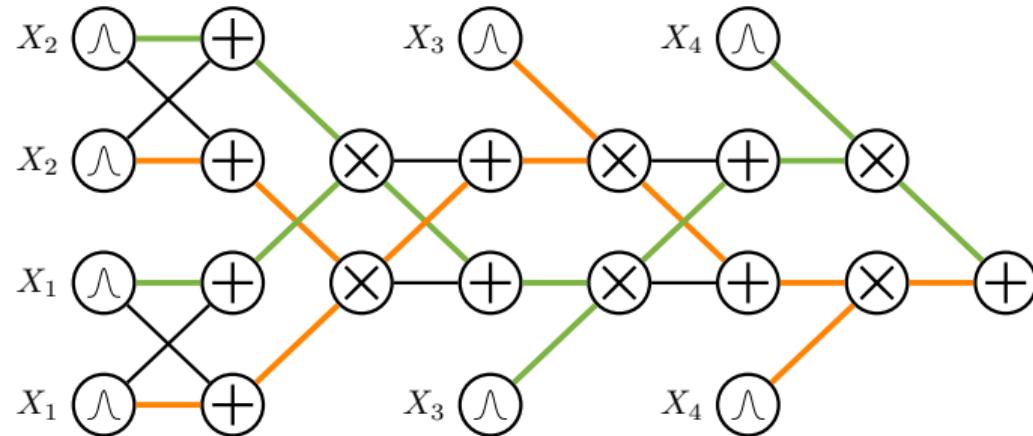
*what about deep circuits?*

## deep mixtures



$$p(\mathbf{x}) = \sum_{\mathcal{T}} \left( \prod_{w_j \in \mathbf{w}_{\mathcal{T}}} w_j \right) \prod_{l \in \text{leaves}(\mathcal{T})} p_l(\mathbf{x})$$

## *deep mixtures*



*an exponential number of mixture components!*

# **...why PCs?**

## **1. A grammar for tractable models**

One formalism to represent many probabilistic models

⇒ #HMMs #Trees #XGBoost, Tensor Networks, ...

# **...why PCs?**

## **1. A grammar for tractable models**

One formalism to represent many probabilistic models

⇒ #HMMs #Trees #XGBoost, Tensor Networks, ...

## **2. Tractability == structural properties!!!**

Exact computations of reasoning tasks are certified by guaranteeing certain structural properties. #marginals #expectations #MAP, #product ...

# *structural properties*

***smoothness***

***decomposability***

***compatibility***

***determinism***

***the combination of certain structural properties guarantees tractable computation of certain query classes***

# *structural properties*

**property A**

*the combination of certain  
structural properties  
**guarantees***

**property B**

*tractable computation of  
certain query classes*

**property C**

**property D**

# *structural properties*

**property A**

*tractable* computation of *arbitrary integrals*

**property B**

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

**property C**

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

**property D**

$\Rightarrow$  *tractable partition function*  
 $\Rightarrow$  *also any conditional* is tractable

# *structural properties*

**smoothness**

*tractable* computation of *arbitrary integrals*

**decomposability**

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

**property C**

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

**property D**

$\Rightarrow$  *tractable partition function*  
 $\Rightarrow$  *also any conditional* is tractable

# *structural properties*

**smoothness**

smoothness  $\wedge$  decomposability  $\implies$  multilinearity

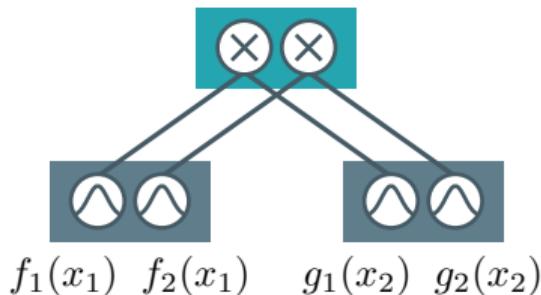
**decomposability**

**property C**

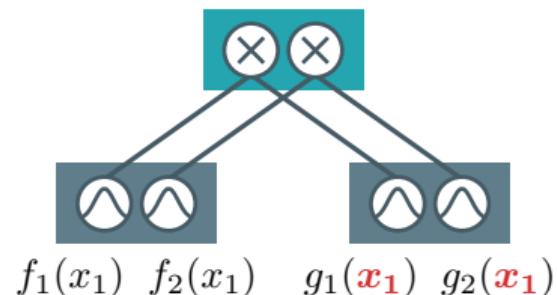
**property D**

# multilinearity

the inputs of product units are defined over disjoint sets of variables



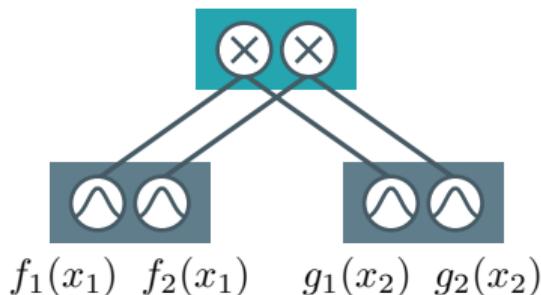
✓ **multilinear**



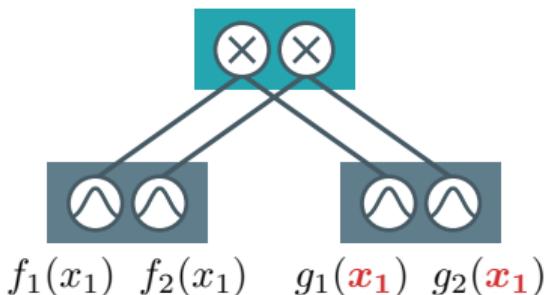
✗ **not multilinear**

# multilinearity

the inputs of product units are defined over disjoint sets of variables



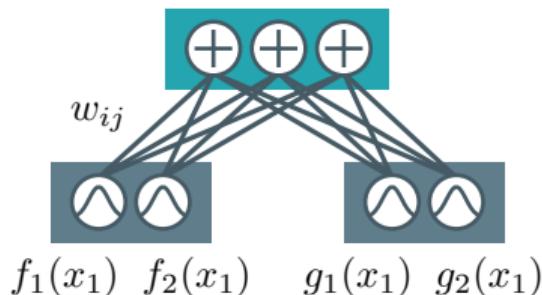
**decomposable circuit**



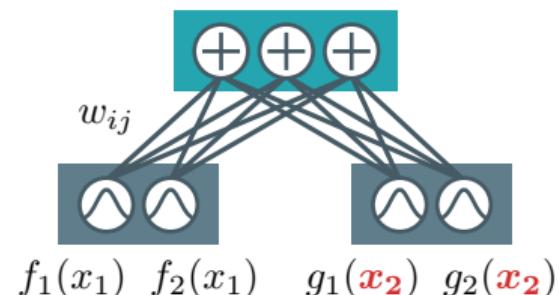
**non-decomposable circuit**

# multilinearity

the inputs of sum units are defined over the same variables



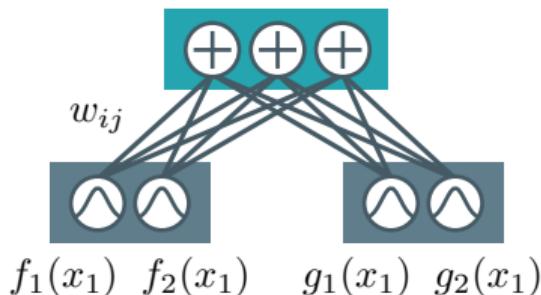
✓ **multilinear**



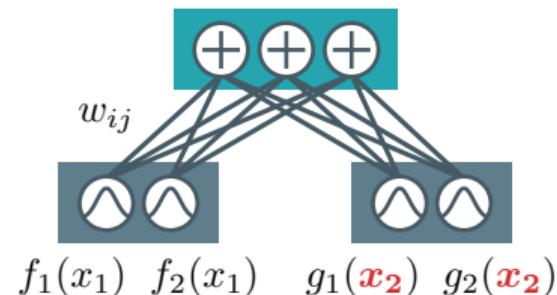
✗ **not multilinear**

# multilinearity

the inputs of sum units are defined over the same variables



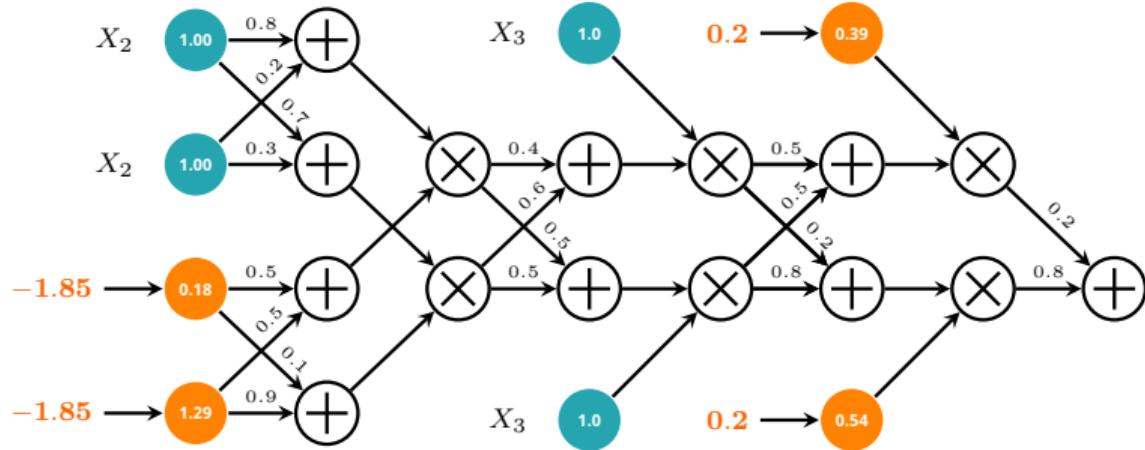
**smooth circuit**



**non-smooth circuit**

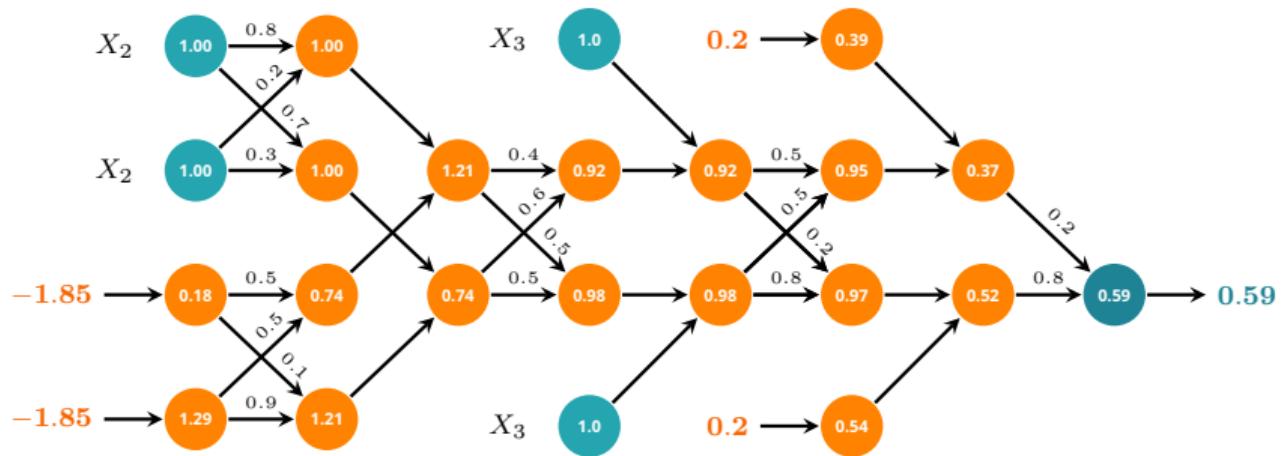
## **marginal queries** = *feedforward* evaluation

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



## **marginal queries** = *feedforward* evaluation

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

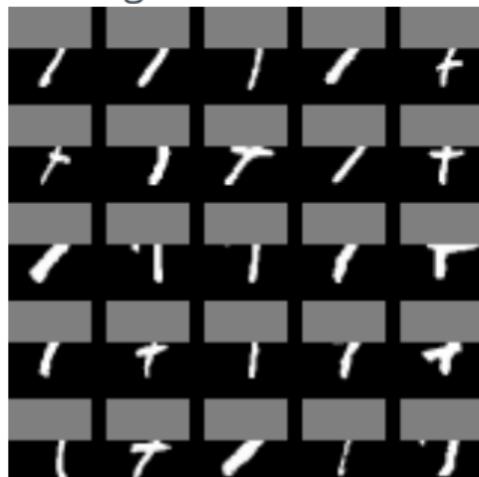


## *tractable marginals on PCs*

Original



Missing

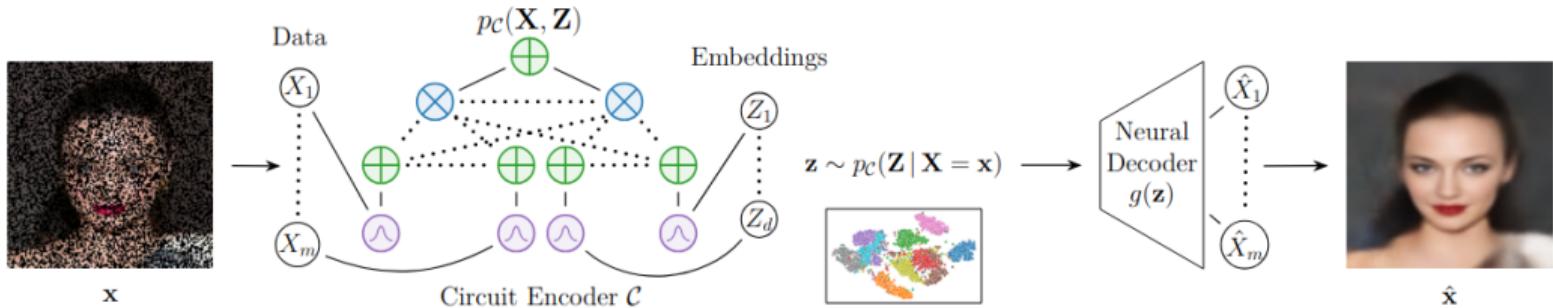


Conditional sample



***use tractable models  
inside intractable pipelines  
where it matters!***

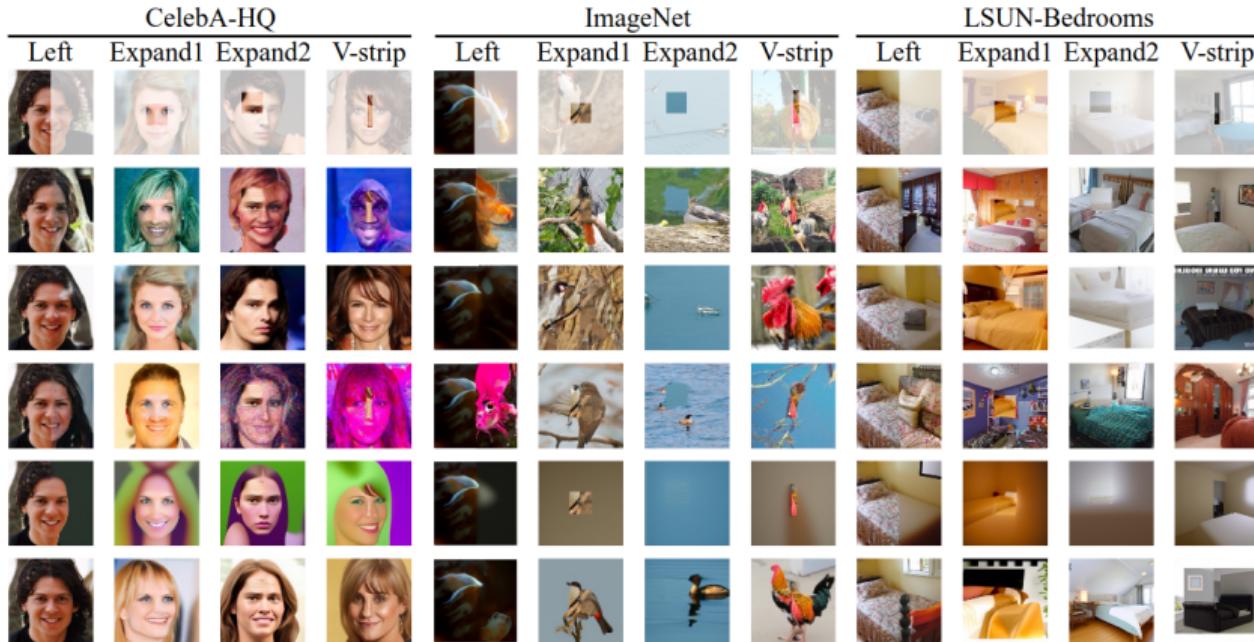
# *tractable + intractable*



***tractable conditioning over every missing mask***  
(under submission)

	MNIST					CIFAR					CelebA					LSUN				
	0%	50%	80%	Vert	Ctr	0%	50%	80%	Vert	Ctr	0%	50%	80%	Vert	Ctr	0%	50%	80%	Vert	Ctr
Data																				
mF																				
SPAE																				
VAE																				
MIWAE																				
APC																				

***better than (V)AEs for missing values***  
*(under submission)*



# *structural properties*

**smoothness**

Integrals involving two or more functions:  
e.g., expectations

**decomposability**

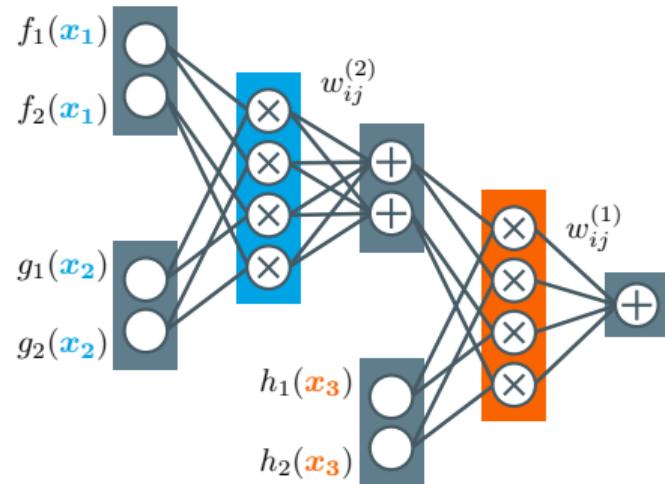
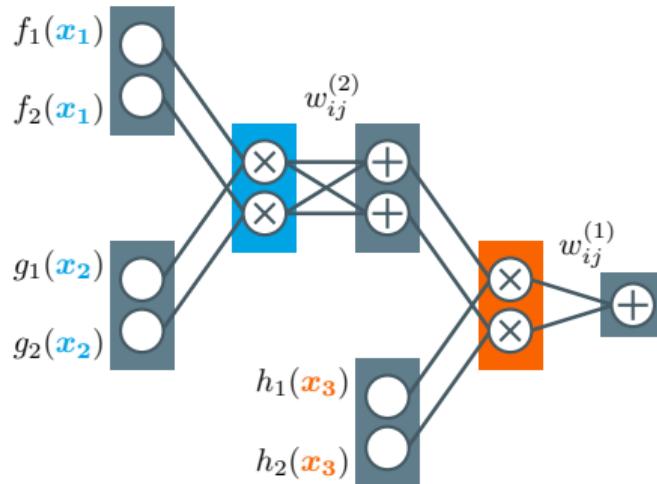
$$\mathbb{E}_{\mathbf{x} \sim p} f(\mathbf{x}) = \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

**compatibility**

when both  $p(\mathbf{x})$  and  $f(\mathbf{x})$  are circuits

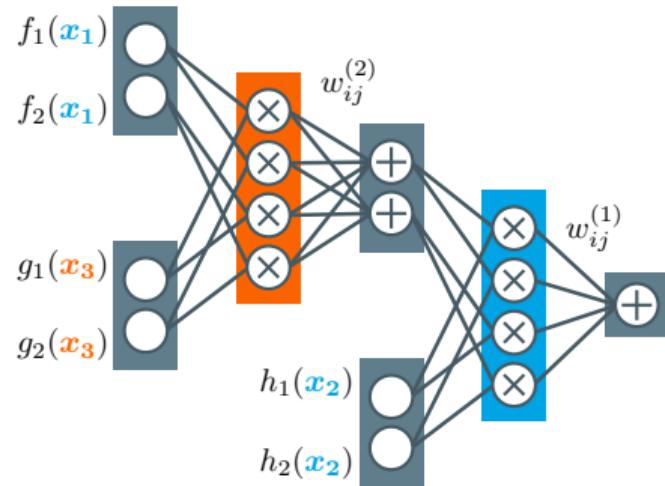
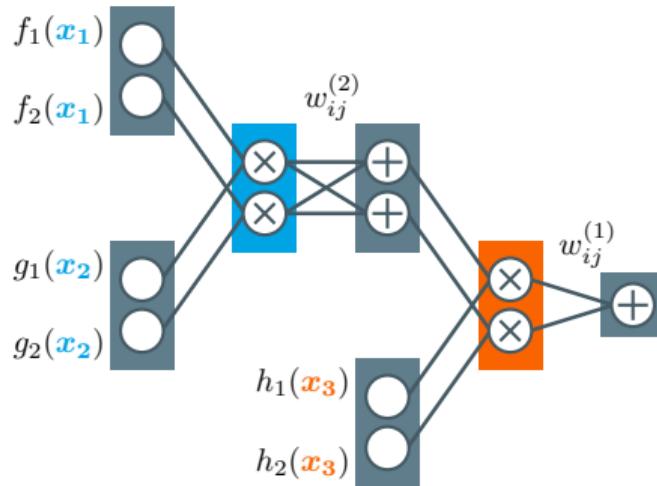
**property D**

# *compatibility*



**compatible circuits**

# *compatibility*



**non-compatible circuits**

# *structural properties*

**smoothness**

**decomposability**

**compatibility**

**property D**

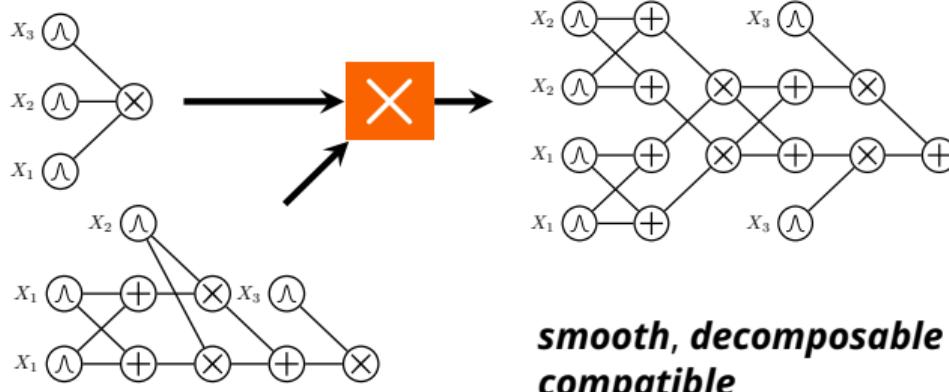
**compatibility**



**smoothness  $\wedge$  decomposability**

**compatibility**  $\Rightarrow$  tractable expectations

# *tractable products*



**compute**  $\mathbb{E}_{\mathbf{x} \sim p} f(\mathbf{x}) = \int p(\mathbf{x}) f(\mathbf{x}) \, d\mathbf{x}$  **in**  $O(|p| |f|)$

# **which structural properties**

*for complex reasoning*



**q<sub>1</sub>**  $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$   
*(missing values)*

*properties A+B*

**q<sub>2</sub>**  $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$   
*(fairness)*

*property C*

**q<sub>3</sub>**  $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} [f(\mathbf{x} + \mathbf{e})]$   
*(adversarial robust.)*

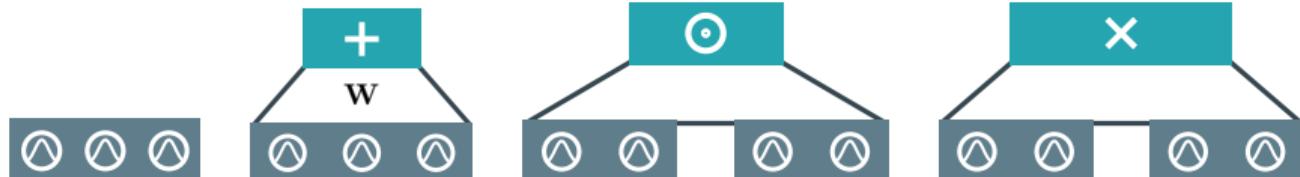
*property A*

```
1 from cirkit.symbolic.circuit import Circuit
2 from cirkit.symbolic.functional import (
3     integrate, multiply)
4
5 # Circuits expectation  $\int [p(x) f(x)] dx$ 
6 def expectation(p: Circuit, f: Circuit) -> Circuit:
7     i = multiply(p, f)
8     return integrate(i)
9
10 # Squared loss  $\int [p(x)-q(x)]^2 dx = E_p[p] + E_q[q] - 2E_p[q]$ 
11 #           =  $\int p^2(x) dx + \int q^2(x) dx - 2\int p(x)q(x) dx$ 
12 def squared_loss(p: Circuit, q: Circuit) -> Circuit:
13     p2 = multiply(p, p)
14     q2 = multiply(q, q)
15     pq = multiply(p, q)
16     return integrate(p2) + integrate(q2) - 2 * integrate(pq)
```

*how to build **deep** circuits?*

# ***probabilistic circuits (PCs)***

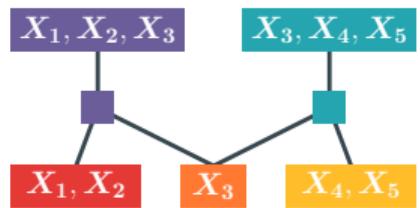
*the layer-wise definition*



```
1 from cirkit.templates import circuit_templates  
2  
3 symbolic_circuit = circuit_templates.image_data(  
4     (1, 28, 28),                      # The shape of MNIST  
5     region_graph='quad-graph',  
6     input_layer='categorical',          # input distributions  
7     sum_product_layer='cp',            # CP, Tucker, CP-T  
8     num_input_units=64,                # overparameterizing  
9     num_sum_units=64,  
10    sum_weight_param=circuit_templates.Parameterization(  
11        activation='softmax',  
12        initialization='normal'  
13    )  
14 )
```

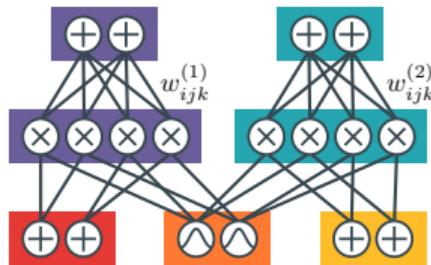
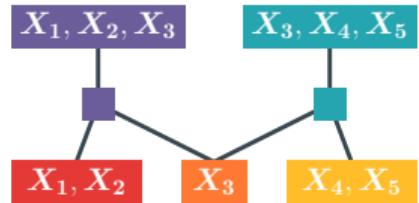
```
1 from cirkit.templates import circuit_templates  
2  
3 symbolic_circuit = circuit_templates.image_data(  
4     (1, 28, 28),  
5     region_graph='quad-graph',  
6     input_layer='categorical',    # input distributions  
7     sum_product_layer='cp',      # CP, Tucker, CP-T  
8     num_input_units=64,          # overparameterizing  
9     num_sum_units=64,  
10    sum_weight_param=circuit_templates.Parameterization(  
11        activation='softmax',  
12        initialization='normal'  
13    )  
14 )
```

# *learning recipe*



1) Build a *region graph*

# *learning recipe*

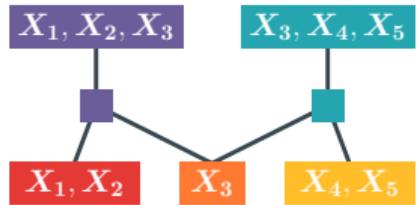


1) Build a *region graph*

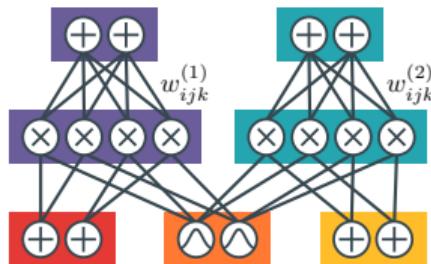
2) Overparameterize

- 2.1) pick a (composite) layer type**
- 2.2) choose how many units per layer**

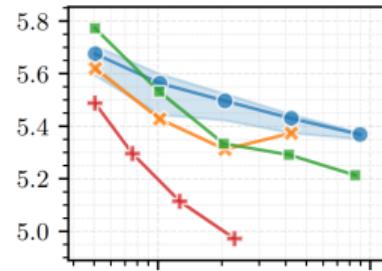
# *learning recipe*



1) Build a *region graph*



2) Overparameterize

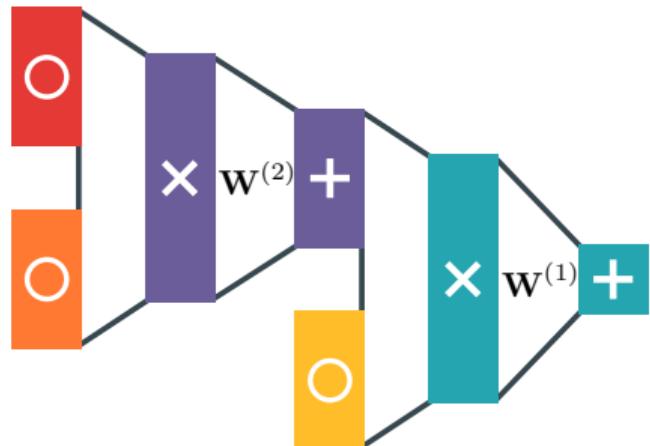
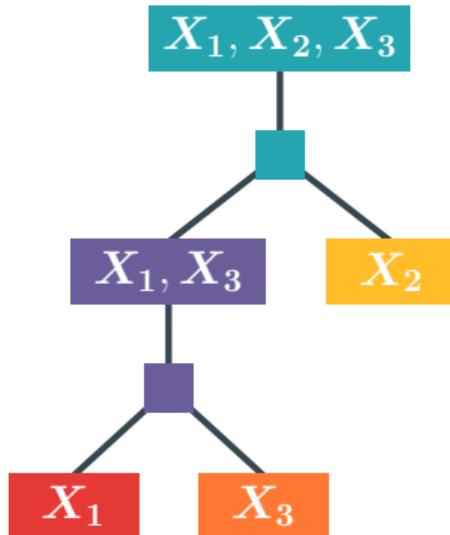


3) Learn parameters

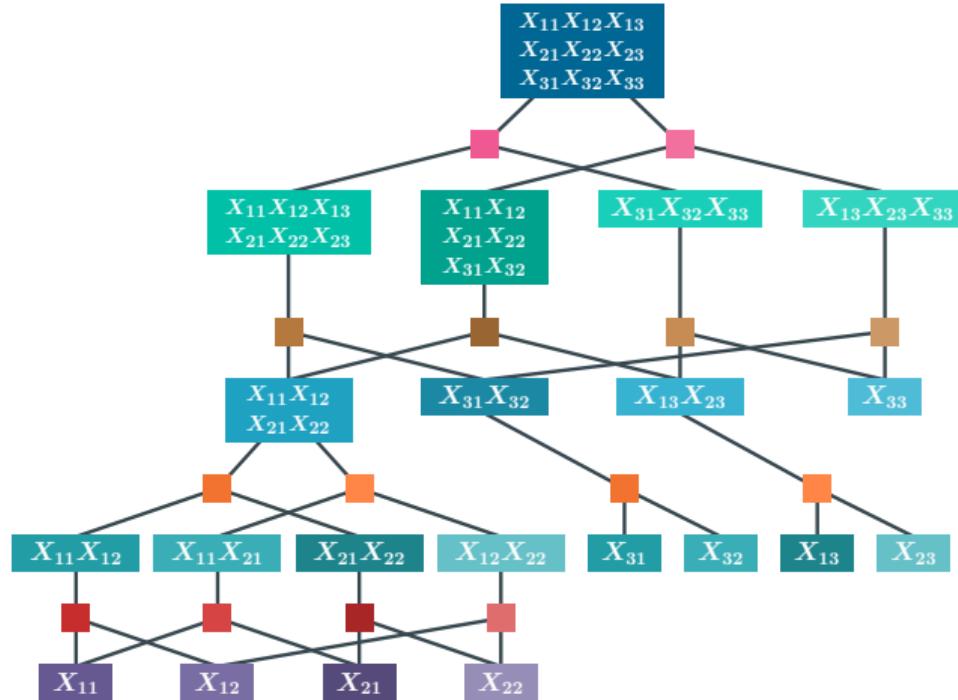
*use any optimizer in pytorch*

# *region graphs*

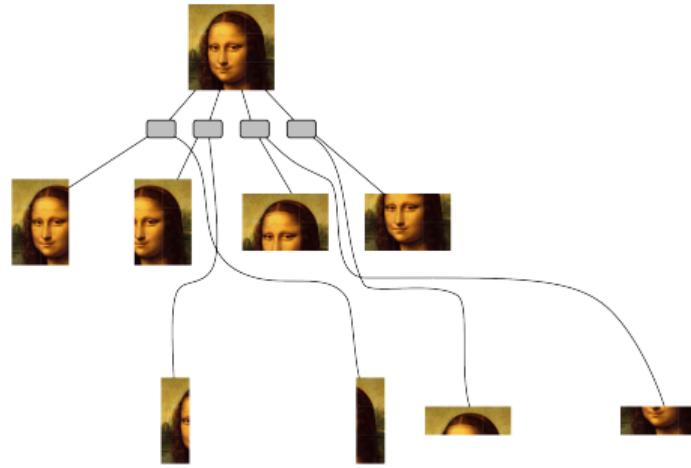
*a template for smooth&decomposable PCs*

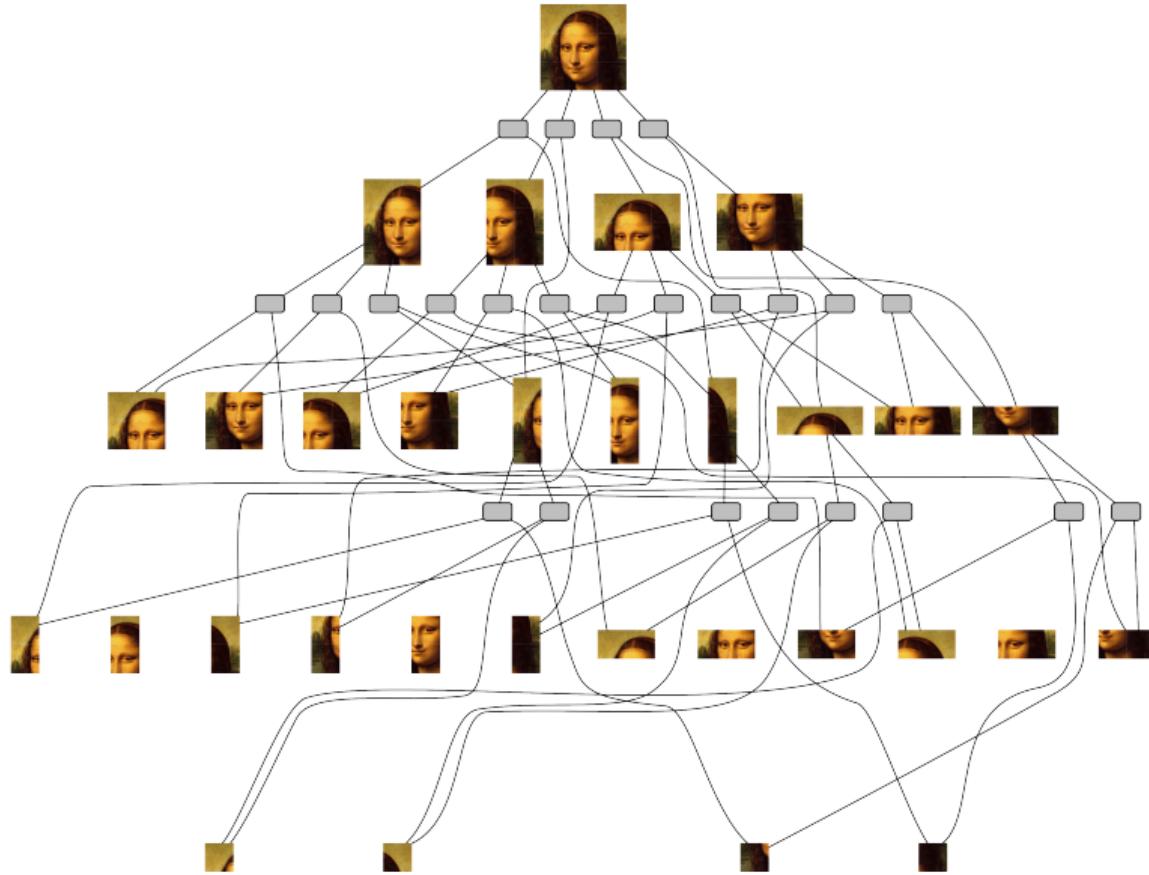


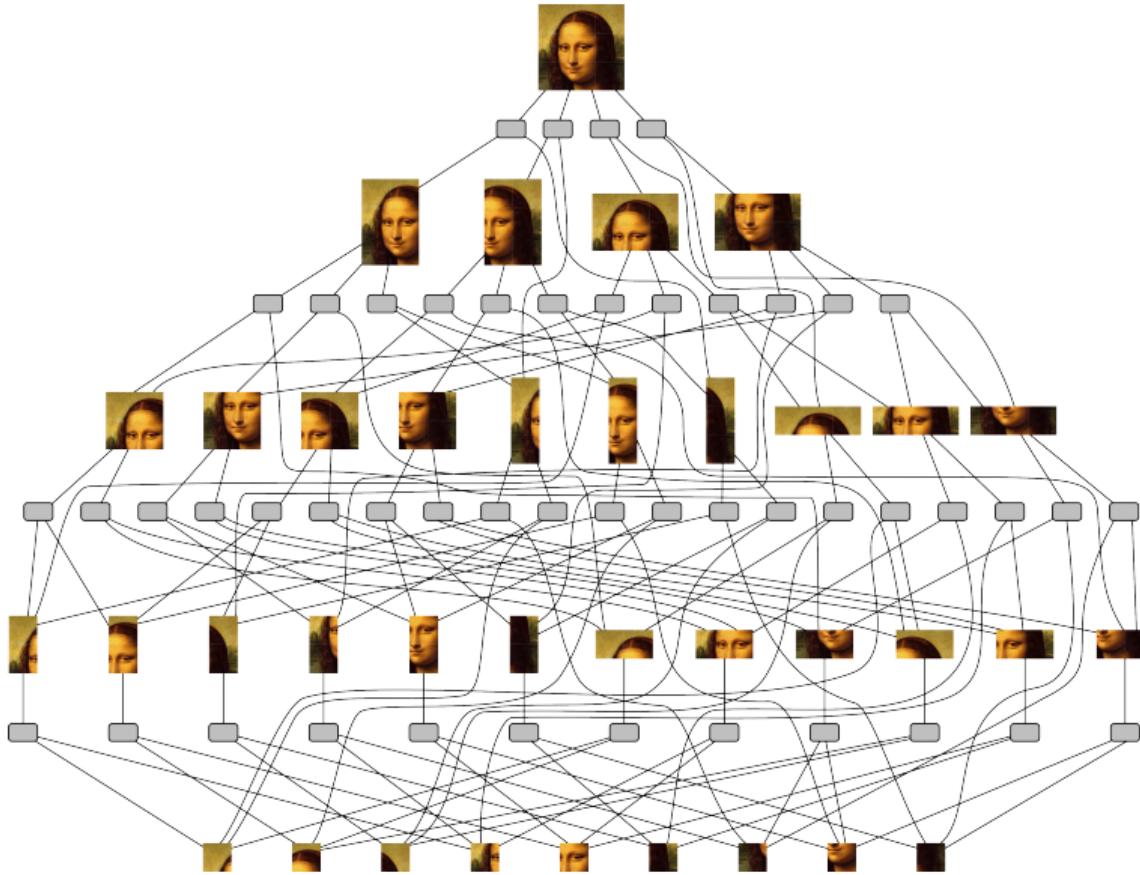
# *which region graph?*







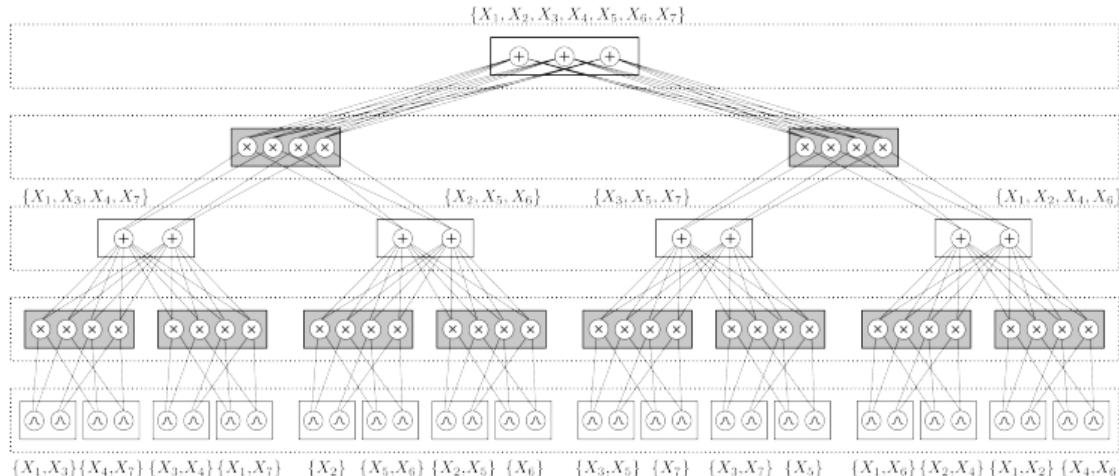




# *random regions graphs*

The “no-learning” option

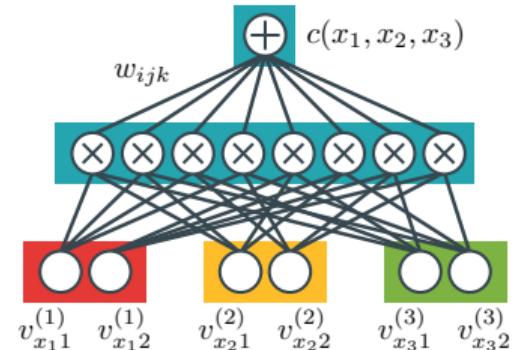
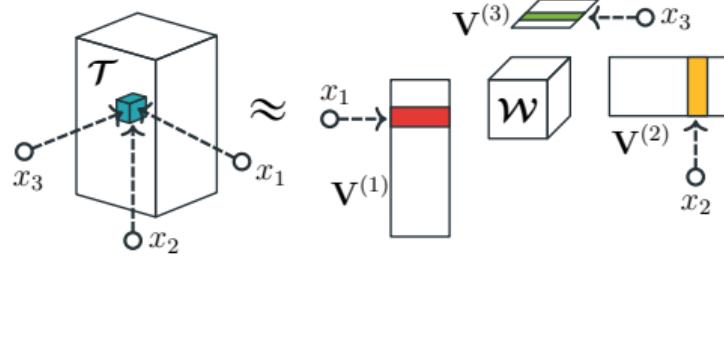
Generating a random region graph, by recursively splitting  $\mathbf{X}$  into two random parts:



```
1 from cirkit.templates import circuit_templates  
2  
3 symbolic_circuit = circuit_templates.image_data(  
4     (1, 28, 28),                      # The shape of MNIST  
5     region_graph='quad-graph',  
6     input_layer='categorical',          # input distributions  
7     sum_product_layer='cp',            # CP, Tucker, CP-T  
8     num_input_units=64,                # overparameterizing  
9     num_sum_units=64,  
10    sum_weight_param=circuit_templates.Parameterization(  
11        activation='softmax',  
12        initialization='normal'  
13    )  
14 )
```

# *circuits layers*

*as tensor factorizations*

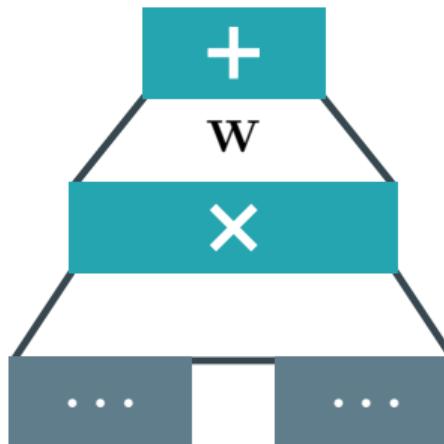
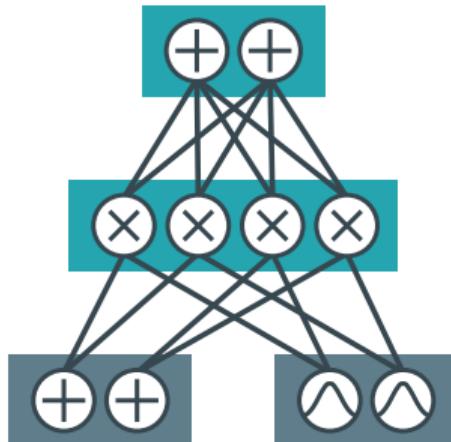


---

loconte2024relationship, loconte2024relationship, loconte2024relationship,  
loconte2024relationship

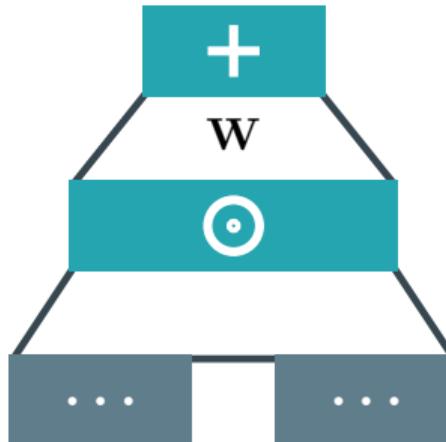
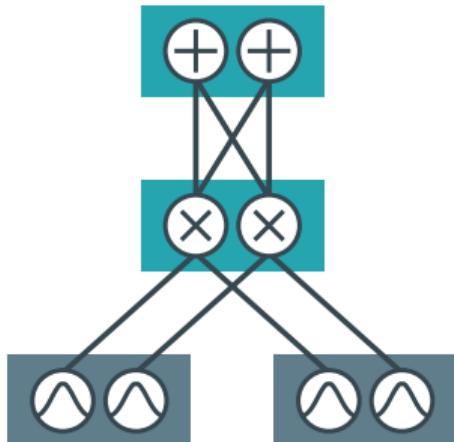
# **more layers**

*Tucker decomposition*

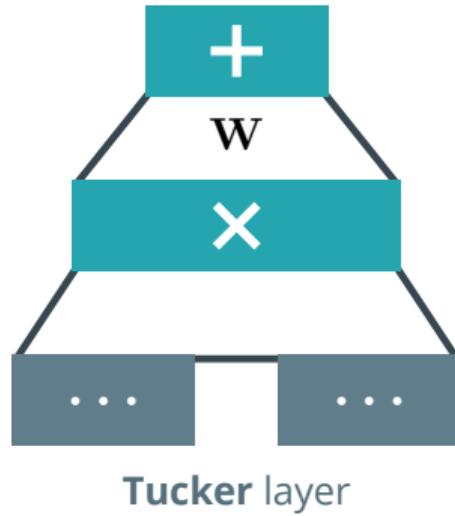
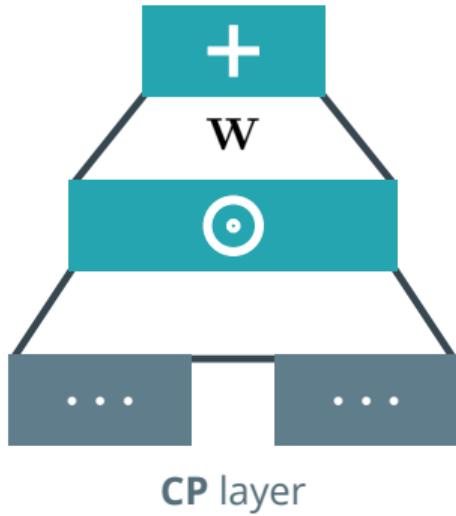


# **more layers**

*Candecomp Parafac (CP) decomposition*



## *more layers*

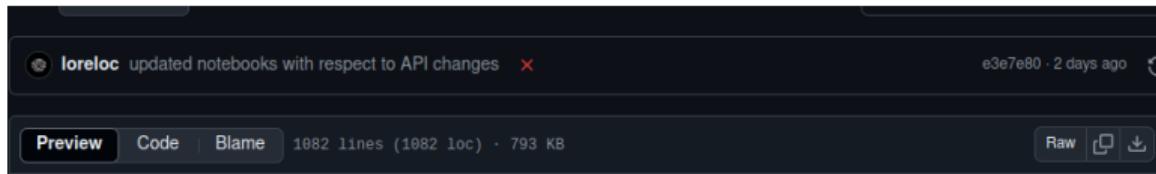


The screenshot shows a GitHub repository page for 'april-tools/cirkit'. The main navigation bar includes 'Code', 'Issues (30)', 'Pull requests (5)', 'Discussions', 'Actions', 'Projects (4)', 'Security', and 'Insights'. Below the navigation is a search bar with placeholder 'Type / to search'. The repository path 'cirkit / notebooks / learning-a-circuit.ipynb' is displayed. A commit by 'adrianjav' with the message 'fix relative links and toc of notebooks' is shown, dated 0137ad1 - 4 months ago. The notebook preview shows the title 'Learning and Evaluating a Probabilistic Circuit' and a descriptive text about learning and evaluating a probabilistic circuit for MNIST images using PyTorch.



## *a notebook on learning a deep circuit on MNIST*

<https://github.com/april-tools/cirkit/blob/main/notebooks/learning-a-circuit.ipynb>



## Notebook on Region Graphs and Sum Product Layers

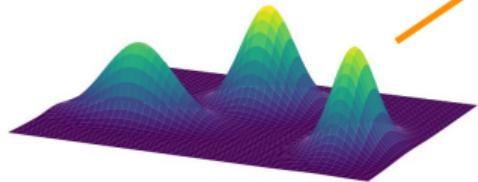
### Goals

By the end of this tutorial you will:

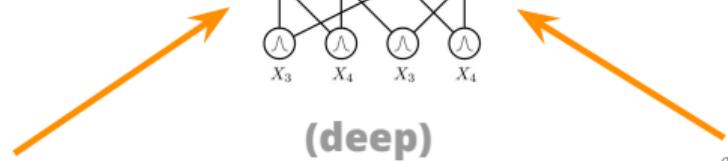
- know what a region graph is
- know how to choose between region graphs for your circuit
- understand how to parametrize a circuit by choosing a sum product layer
- build circuits to tractably estimate a probability distribution over images<sup>1</sup>

***mix& match your RGs and layers***

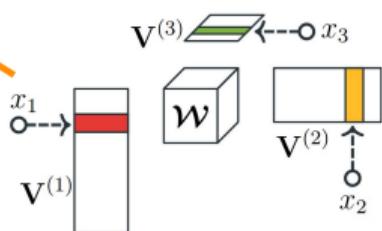
<https://github.com/april-tools/cirkit/blob/main/notebooks/region-graphs-and-parametrisation.ipynb>



(hierarchical)  
mixtures

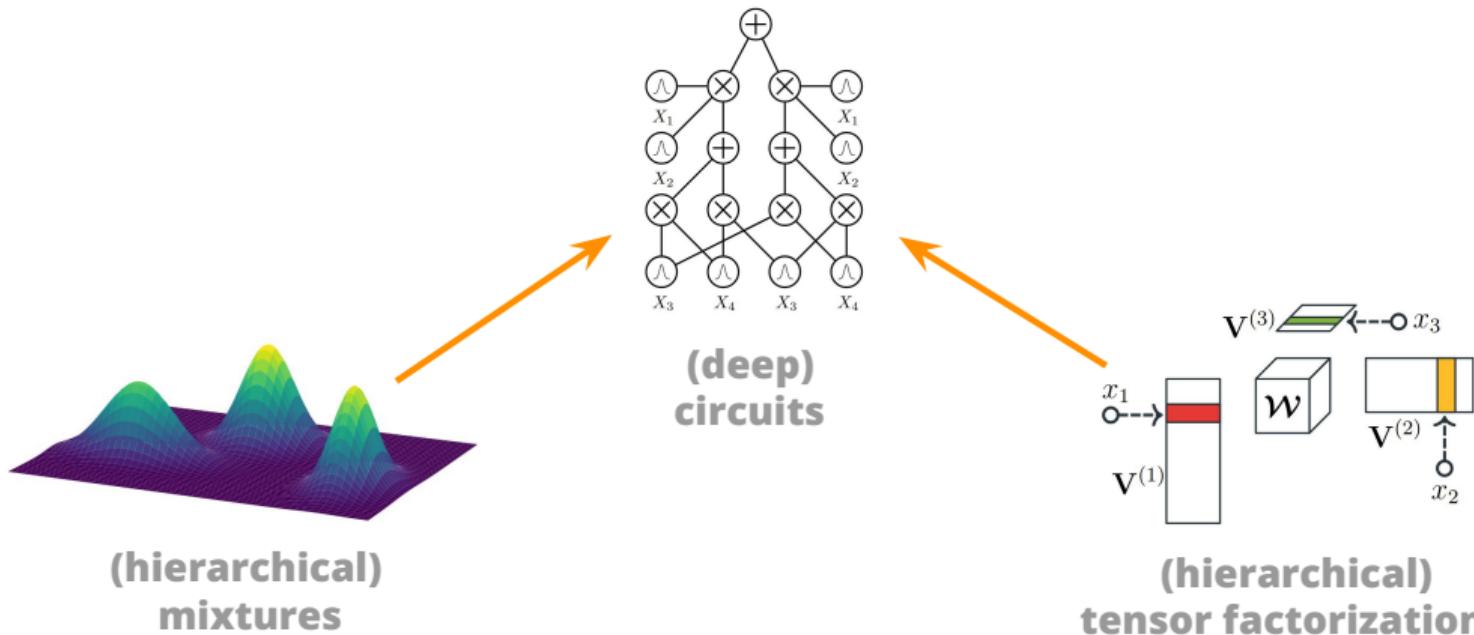


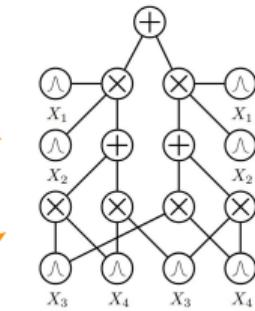
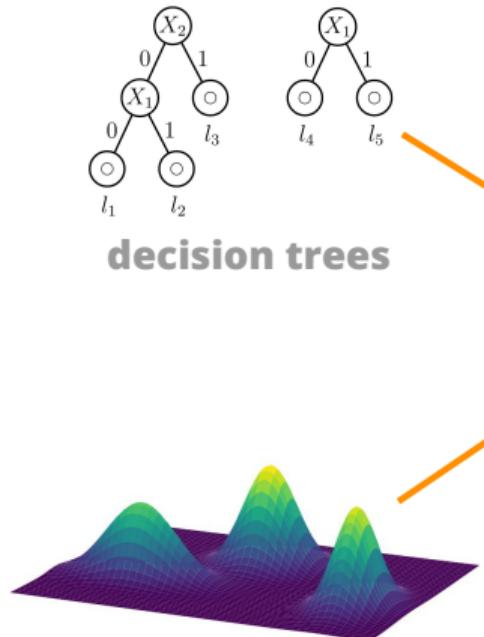
(deep)  
circuits



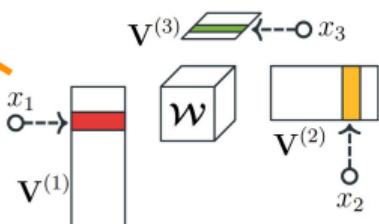
(hierarchical)  
tensor factorizations

*can we represent also non-probabilistic models?...*





**logical  
formulas  
& constraints**



**(hierarchical)  
tensor factorizations**

---

# Semantic Probabilistic Layers for Neuro-Symbolic Learning

---

**Kareem Ahmed**

CS Department  
UCLA

ahmedk@cs.ucla.edu

**Stefano Teso**

CIMeC and DISI  
University of Trento

stefano.teso@unitn.it

**Kai-Wei Chang**

CS Department  
UCLA

kwchang@cs.ucla.edu

**Guy Van den Broeck**

CS Department  
UCLA

guyvdb@cs.ucla.edu

**Antonio Vergari**

School of Informatics  
University of Edinburgh

avergari@ed.ac.uk

***enforce constraints in neural networks at NeurIPS 2022***

# When?



Ground Truth

*e.g. predict shortest path in a map*

# When?



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

Ground Truth

***n*esy structured output prediction (SOP) tasks**

# 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

**nesy structured output prediction (SOP) tasks**

# 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.  $\mathbf{y} \models K$  // e.g., that form a valid path

**nesy structured output prediction (SOP) tasks**

# When?



Ground Truth

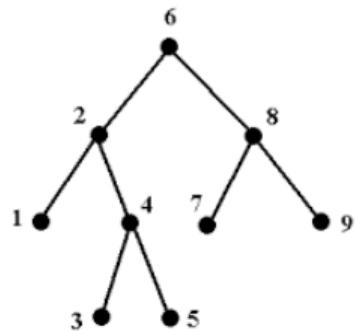
given  $\mathbf{x}$  // e.g. a tile map

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

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

**nesy structured output prediction (SOP) tasks**

# When?



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

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

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

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

**hierarchical multi-label classification**

# When?

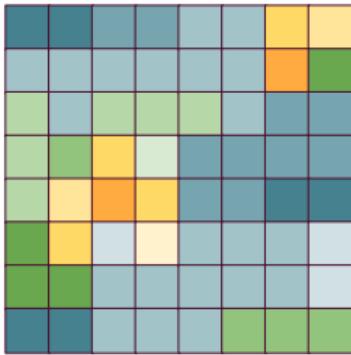


given  $\mathbf{x}$  // e.g. a user preference over  $K - N$  sushi types  
find  $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} p_{\theta}(\mathbf{y} \mid \mathbf{x})$  // e.g. prefs over  $N$  more types  
s.t.  $\mathbf{y} \models K$  // e.g., output valid rankings

## ***user preference learning***

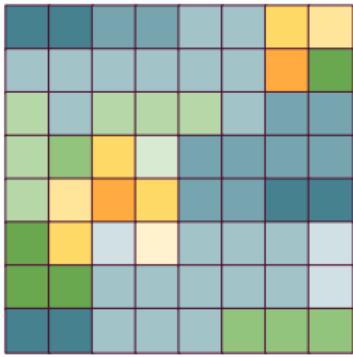
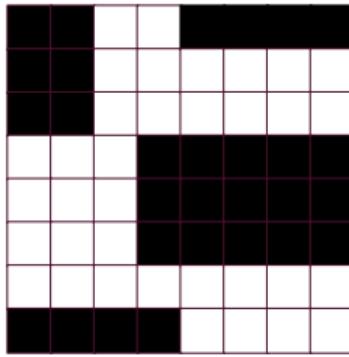
---

*Choi, Van den Broeck, and Darwiche, "Tractable learning for structured probability spaces: A case study in learning preference distributions",  
Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI), 2015*

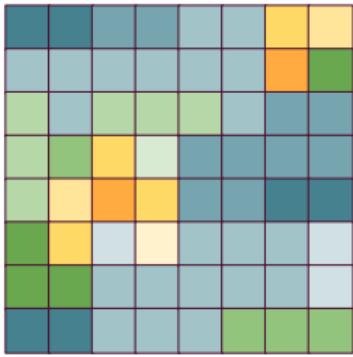
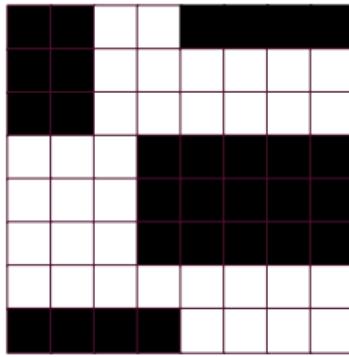
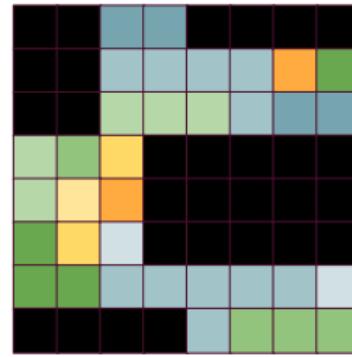


$$q(\mathbf{x})$$

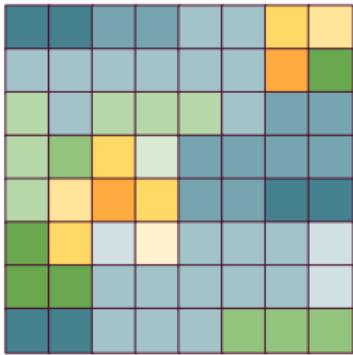
**start from a distribution  $q(\mathbf{x})...$**

 $q(\mathbf{x})$  $c(\mathbf{x})$ 

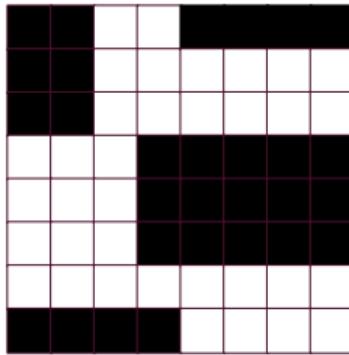
**...and cut its support by a constraint  $c(\mathbf{x})$**

 $q(\mathbf{x})$  $c(\mathbf{x})$  $q(\mathbf{x}) \cdot c(\mathbf{x})$ 

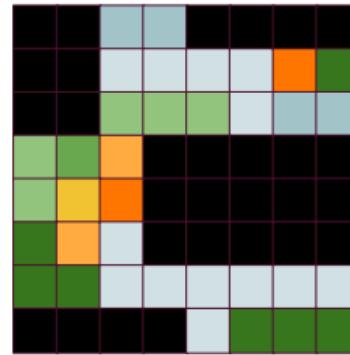
**by multiplying them**  $q(\mathbf{x})c(\mathbf{x})\dots$



$q(\mathbf{x})$

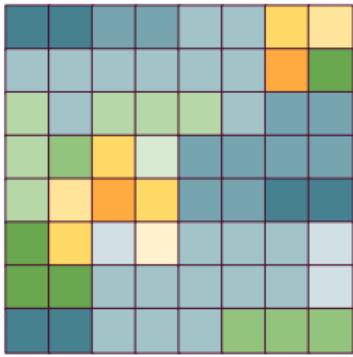
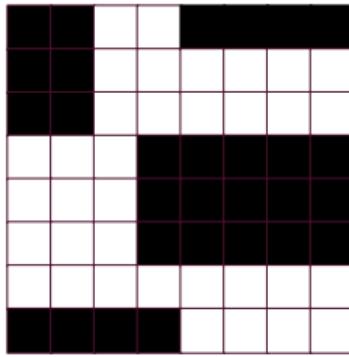
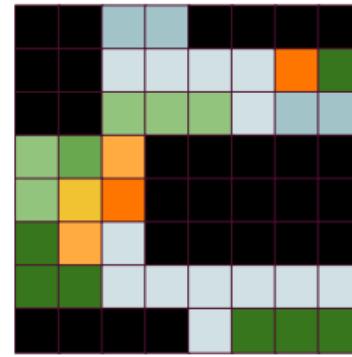


$c(\mathbf{x})$



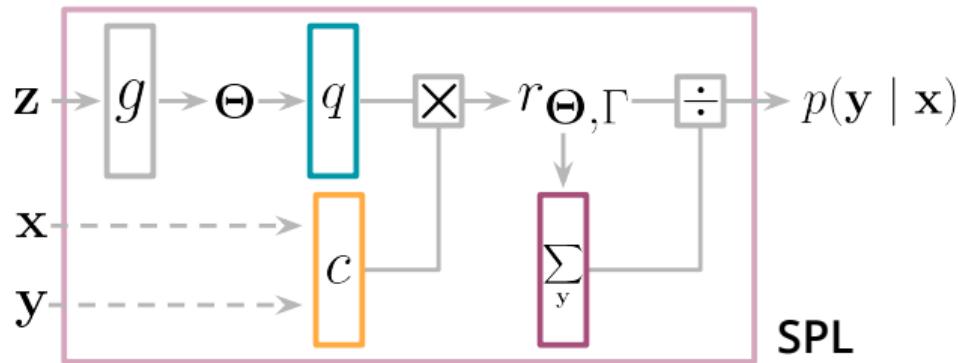
$$\frac{q(\mathbf{x}) \cdot c(\mathbf{x})}{\sum_{\mathbf{x}} q(\mathbf{x}) \cdot c(\mathbf{x})}$$

**and then renormalizing them!**

 $q(\mathbf{x})$  $c(\mathbf{x})$ 

$$\frac{q(\mathbf{x}) \cdot c(\mathbf{x})}{\sum_{\mathbf{x}} q(\mathbf{x}) \cdot c(\mathbf{x})}$$

**states with zero probability will never be predicted  
(nor sampled)**



$$p(\mathbf{y} \mid \mathbf{x}) = \mathbf{q}_{\Theta}(\mathbf{y} \mid g(\mathbf{z})) \cdot \mathbf{c}_K(\mathbf{x}, \mathbf{y}) / \mathcal{Z}(\mathbf{x})$$

$$\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}} q_{\Theta}(\mathbf{y} \mid \mathbf{x}) \cdot c_K(\mathbf{x}, \mathbf{y})$$



Ground Truth



ResNet-18



Semantic Loss



circuits

***predictions guarantee a logical constraint 100% of the time!***

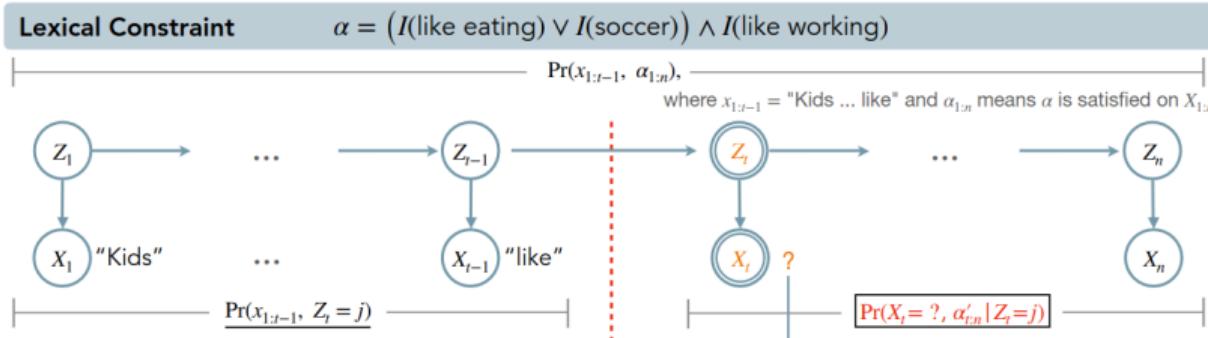
**SPL**  
**(and variants)**  
*everywhere*

---

## Tractable Control for Autoregressive Language Generation

---

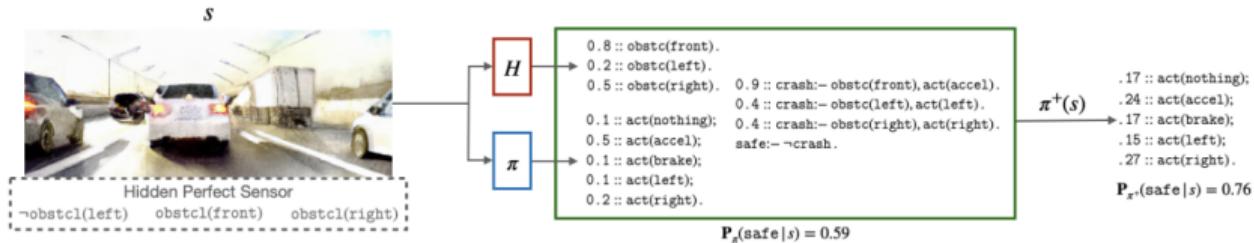
Honghua Zhang<sup>\* 1</sup> Meihua Dang<sup>\* 1</sup> Nanyun Peng<sup>1</sup> Guy Van den Broeck<sup>1</sup>



***constrained text generation with LLMs (ICML 2023)***

# Safe Reinforcement Learning via Probabilistic Logic Shields

Wen-Chi Yang<sup>1</sup>, Giuseppe Marra<sup>1</sup>, Gavin Rens and Luc De Raedt<sup>1,2</sup>



***reliable reinforcement learning (AAAI 23)***

---

# How to Turn Your Knowledge Graph Embeddings into Generative Models

---

**Lorenzo Loconte**  
University of Edinburgh, UK  
l.loconte@sms.ed.ac.uk

**Nicola Di Mauro**  
University of Bari, Italy  
nicola.dimauro@uniba.it

**Robert Peharz**  
TU Graz, Austria  
robert.peharz@tugraz.at

**Antonio Vergari**  
University of Edinburgh, UK  
avergari@ed.ac.uk

*enforce constraints in knowledge graph embeddings  
oral at NeurIPS 2023*

---

# Logically Consistent Language Models via Neuro-Symbolic Integration

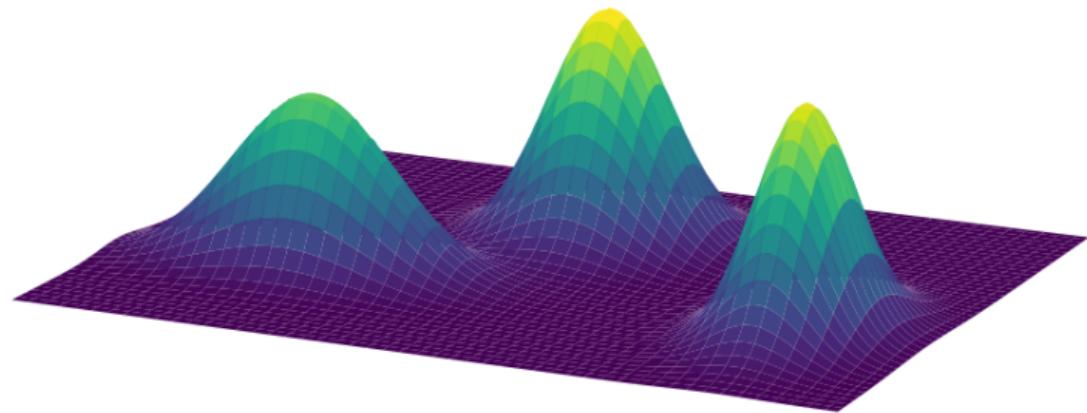
---

The figure displays three panels illustrating logical reasoning:

- Forward Implication**:  $A \rightarrow \neg B$ .  
A: (albatross, isA, bird)  
B: (albatross, isA, fish)  
Is an albatross a bird? (Yes) Yes.  
Is an albatross a fish? (Yes) No.  
Logical: ✕ Factual: ✕
- Reverse Implication**:  $\neg B \rightarrow \neg A$ .  
B: (albatross, isNotA, organism)  
A: (albatross, isNotA, living thing)  
Is it true that an albatross is not an organism? (Yes) Yes.  
Is it true that an albatross is not a living thing? (Yes) No.  
Logical: ✕ Factual: ✕
- Negation**:  $A \oplus \bar{A}$ .  
A: (computer, isA, airplane)  
Ā: (computer, isNotA, airplane)  
Is a computer a airplane? (No) No.  
Is it true that a computer is not a airplane? (Yes) Yes.  
Logical: ✕ Factual: ✕

Legend:  
LLM2 = LoCo-LLMs 2  
LoCo-LLMs 2 = LLaMA 2

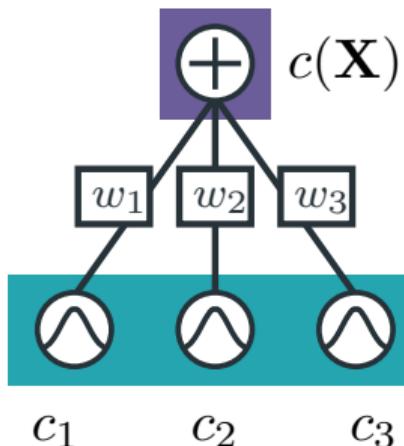
*improving logical (self-)consistency in LLMs at ICLR 2025*



$$c(\mathbf{X}) = \sum_{i=1}^K w_i c_i(\mathbf{X}), \quad \text{with } w_i \geq 0, \quad \sum_{i=1}^K w_i = 1$$

# **additive MMs**

are so cool!



easily represented as shallow PCs

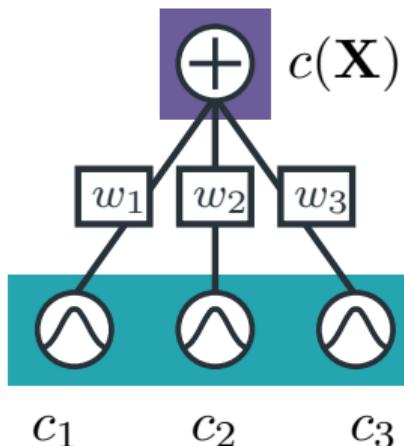
these are **monotonic** PCs

if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

# **additive MMs**

are so cool!



easily represented as shallow PCs

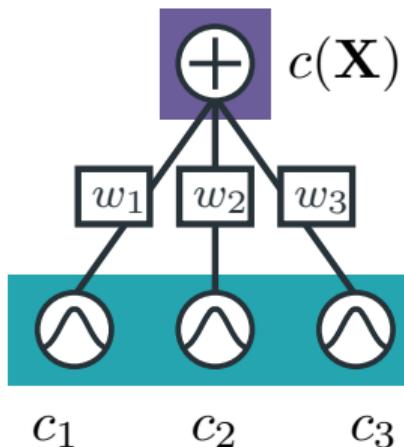
these are **monotonic** PCs

if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

## **additive MMs**

are so cool!



easily represented as shallow PCs

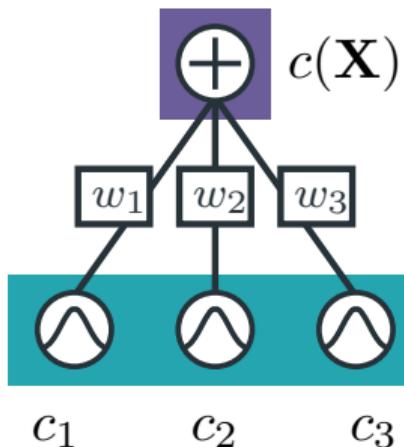
these are **monotonic** PCs

if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

## **additive MMs**

are so cool!



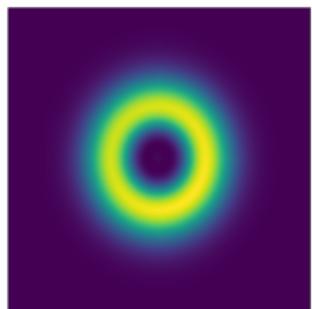
easily represented as shallow PCs

these are **monotonic** PCs

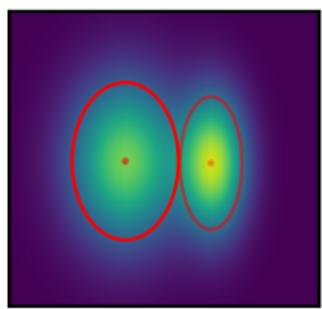
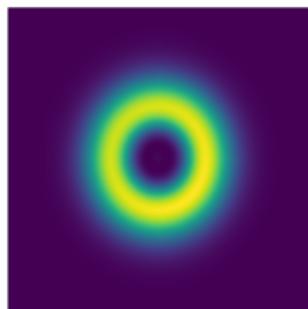
if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

*however...*

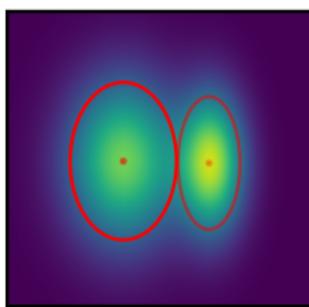
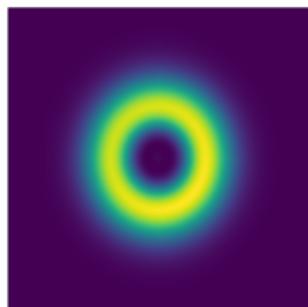


**however...**

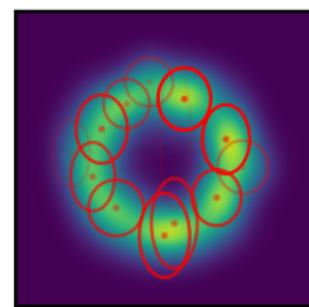


GMM ( $K = 2$ )

*however...*

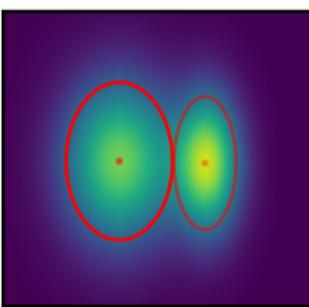
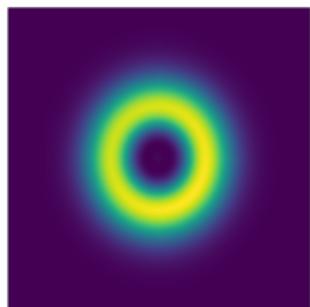


GMM ( $K = 2$ )

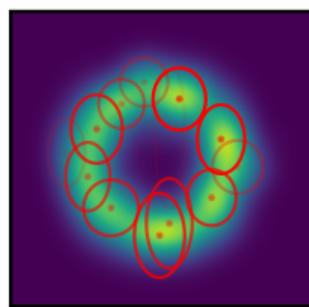


GMM ( $K = 16$ )

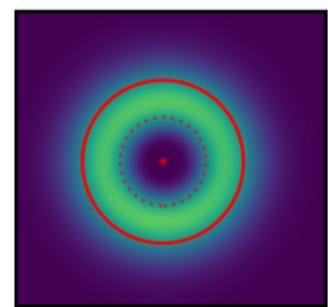
**however...**



GMM ( $K = 2$ )



GMM ( $K = 16$ )

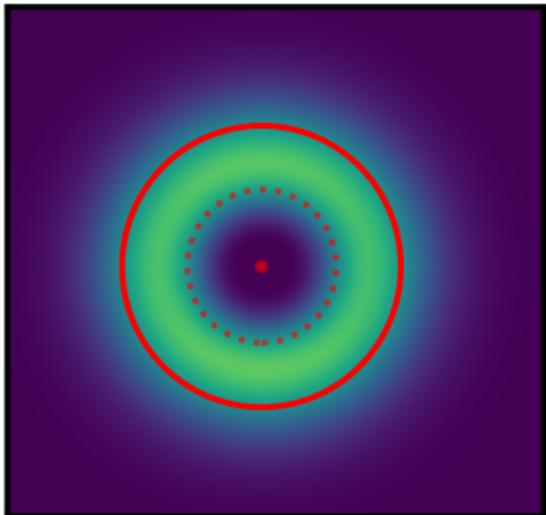


nGMM<sup>2</sup> ( $K = 2$ )

*spoiler*

**shallow mixtures  
with negative parameters  
can be *exponentially more compact* than  
deep ones with positive parameters.**

# **subtractive MMs**



also called negative/signed/**subtractive** MMs

⇒ or **non-monotonic** circuits,...

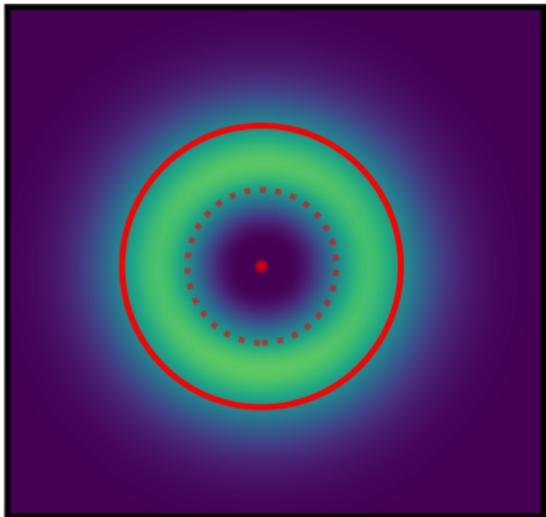
**issue:** how to preserve non-negative outputs?

well understood for simple parametric forms

e.g., Weibulls, Gaussians

⇒ constraints on variance, mean

# **subtractive MMs**



also called negative/signed/**subtractive** MMs

⇒ *or non-monotonic circuits,...*

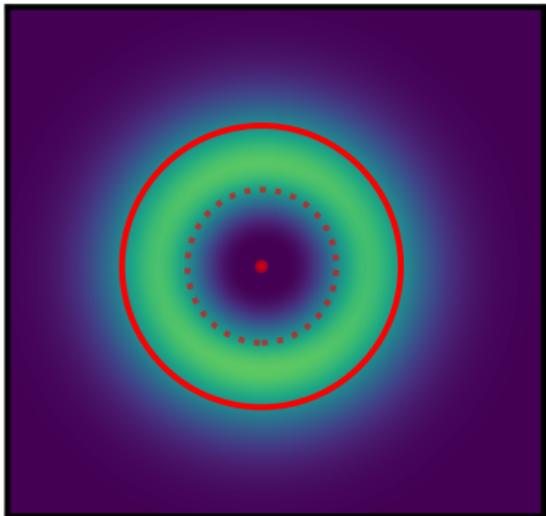
**issue:** how to preserve non-negative outputs?

well understood for simple parametric forms

e.g., Weibulls, Gaussians

⇒ *constraints on variance, mean*

# **subtractive MMs**



also called negative/signed/**subtractive** MMs

⇒ or **non-monotonic** circuits,...

**issue:** how to preserve non-negative outputs?

well understood for simple parametric forms

e.g., Weibulls, Gaussians

⇒ constraints on variance, mean

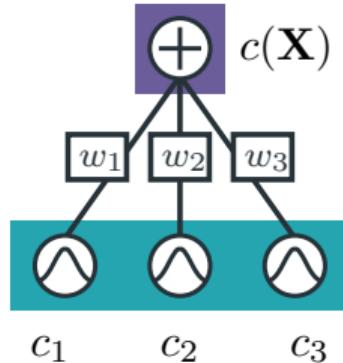
**tl;dr**

***“Understand when and how  
we can use negative parameters  
in deep subtractive mixture models”***

**tl;dr**

***“Understand when and how  
we can use negative parameters  
in deep **non-monotonic circuits**”***

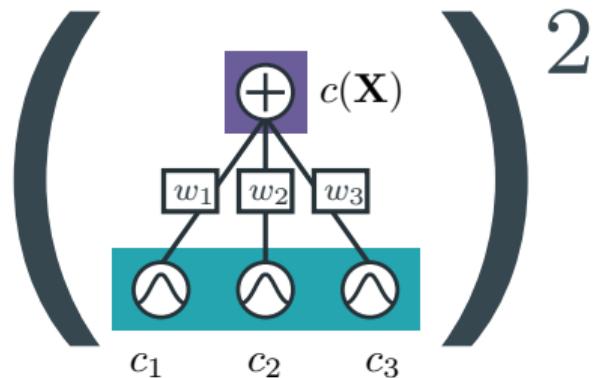
# *subtractive MMs as circuits*



a **non-monotonic** smooth and (structured) decomposable circuit  
⇒ possibly with negative outputs

$$c(\mathbf{X}) = \sum_{i=1}^K w_i c_i(\mathbf{X}), \quad w_i \in \mathbb{R},$$

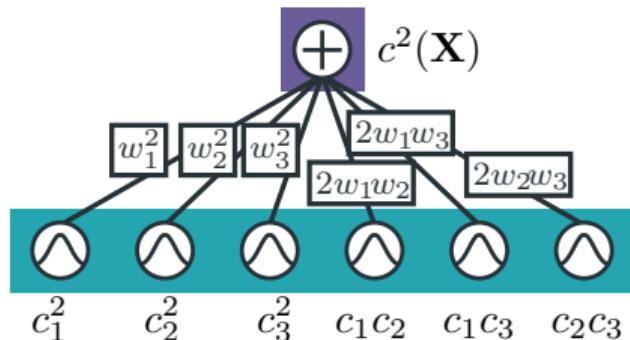
## *squaring shallow MMs*



$$c^2(\mathbf{X}) = \left( \sum_{i=1}^K w_i c_i(\mathbf{X}) \right)^2$$

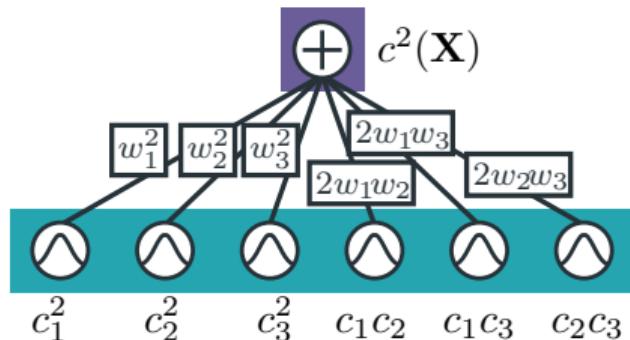
⇒ ensure non-negative output

# *squaring shallow MMs*



$$\begin{aligned} c^2(\mathbf{X}) &= \left( \sum_{i=1}^K w_i c_i(\mathbf{X}) \right)^2 \\ &= \sum_{i=1}^K \sum_{j=1}^K w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X}) \end{aligned}$$

# *squaring shallow MMs*

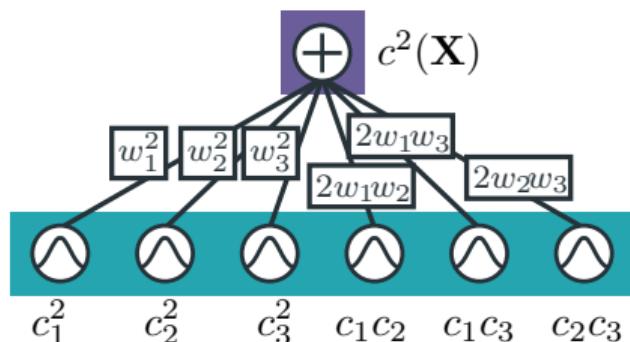


$$\begin{aligned} c^2(\mathbf{X}) &= \left( \sum_{i=1}^K w_i c_i(\mathbf{X}) \right)^2 \\ &= \sum_{i=1}^K \sum_{j=1}^K w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X}) \end{aligned}$$

still a smooth and (str) decomposable PC with  $\mathcal{O}(K^2)$  components!

$\Rightarrow$  but still  $\mathcal{O}(K)$  parameters

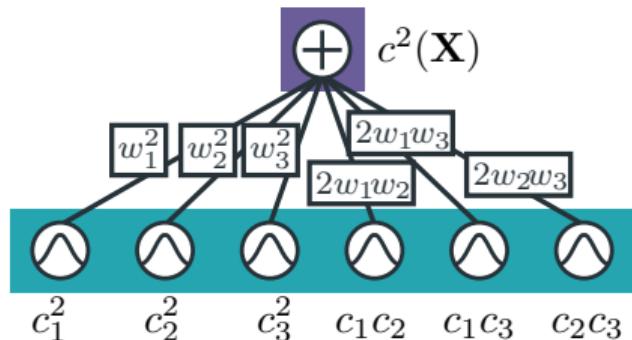
# *squaring shallow MMs*



$$\begin{aligned} c^2(\mathbf{X}) &= \left( \sum_{i=1}^K w_i c_i(\mathbf{X}) \right)^2 \\ &= \sum_{i=1}^K \sum_{j=1}^K w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X}) \end{aligned}$$

to **renormalize**, we have to compute  $\sum_i \sum_j w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x}) d\mathbf{x}$

# *squaring shallow MMs*



$$\begin{aligned} c^2(\mathbf{X}) &= \left( \sum_{i=1}^K w_i c_i(\mathbf{X}) \right)^2 \\ &= \sum_{i=1}^K \sum_{j=1}^K w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X}) \end{aligned}$$

to **renormalize**, we have to compute  $\sum_i \sum_j w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x}) d\mathbf{x}$   
⇒ or we pick  $c_i, c_j$  to be **orthonormal**...!

---

## EigenVI: score-based variational inference with orthogonal function expansions

---

**Diana Cai**  
Flatiron Institute  
[dcai@flatironinstitute.org](mailto:dcai@flatironinstitute.org)

**Chirag Modi**  
Flatiron Institute  
[cmodi@flatironinstitute.org](mailto:cmodi@flatironinstitute.org)

**Charles C. Margossian**  
Flatiron Institute  
[cmargossian@flatironinstitute.org](mailto:cmargossian@flatironinstitute.org)

**Robert M. Gower**  
Flatiron Institute  
[rgower@flatironinstitute.org](mailto:rgower@flatironinstitute.org)

**David M. Blei**  
Columbia University  
[david.blei@columbia.edu](mailto:david.blei@columbia.edu)

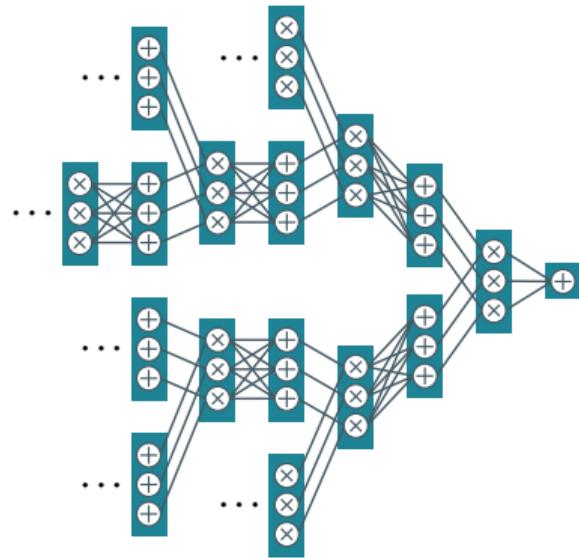
**Lawrence K. Saul**  
Flatiron Institute  
[lsaul@flatironinstitute.org](mailto:lsaul@flatironinstitute.org)

*orthonormal squared mixtures for VI*

*wait...*

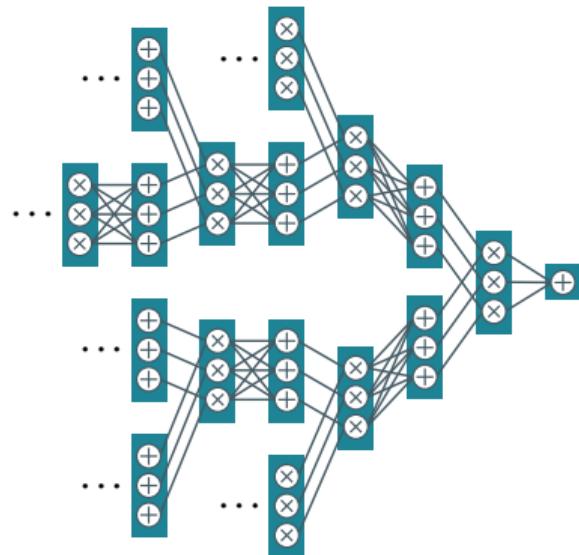
*“do negative parameters  
really boost expressiveness?  
and...always?”*

# theorem



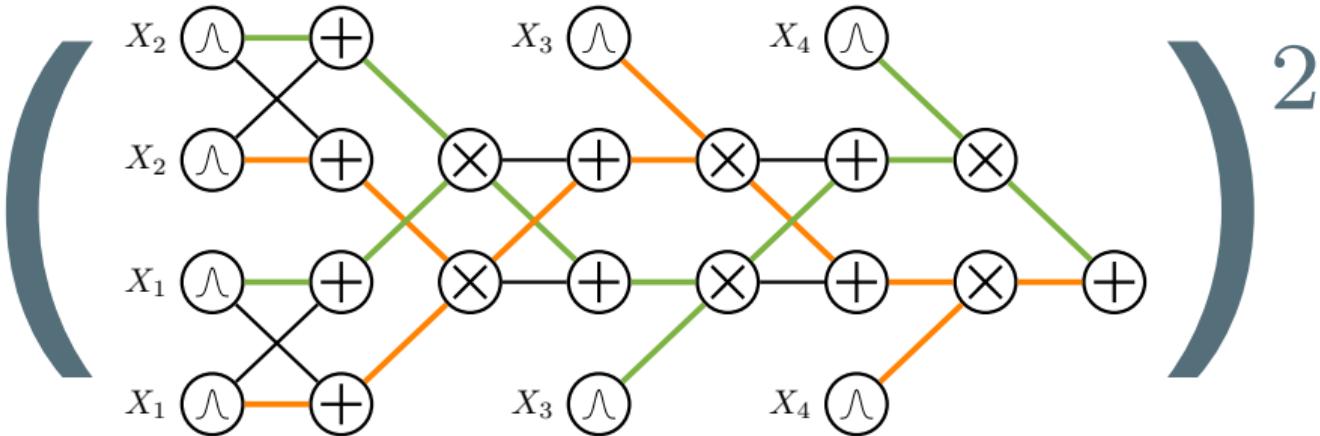
$\exists p$  requiring exponentially large  
monotonic circuits...

# *theorem*



$$\left( \begin{array}{c} \text{...} \\ \text{...} \\ \text{...} \end{array} \right)^2$$

**...but compact  
squared non-monotonic circuits**

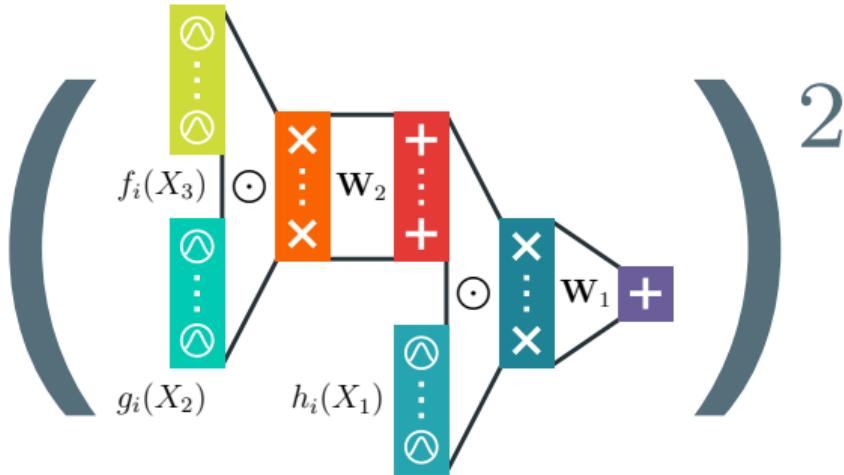


**how to efficiently square (and *renormalize*) a deep PC?**

# *compositional inference I*



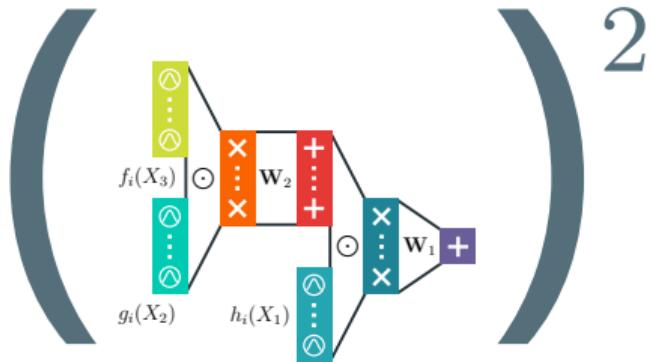
```
1 from cirkit.symbolic.functional import integrate, multiply  
2  
3 #  
4 # create a deep circuit  
5 c = build_symbolic_circuit('quad-tree-4')  
6  
7 #  
8 # compute the partition function of c^2  
9 def renormalize(c):  
10     c2 = multiply(c, c)  
11     return integrate(c2)
```



**how to efficiently square (and *renormalize*) a deep PC?**

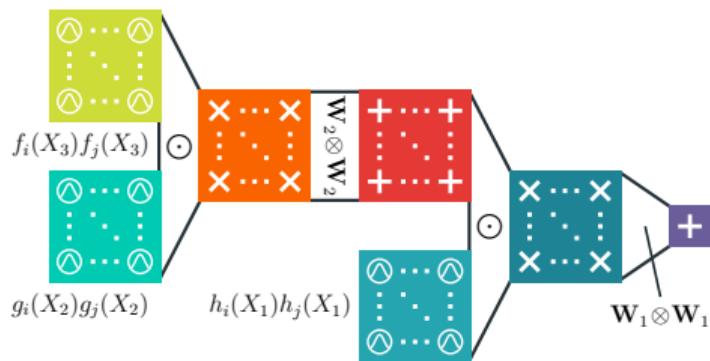
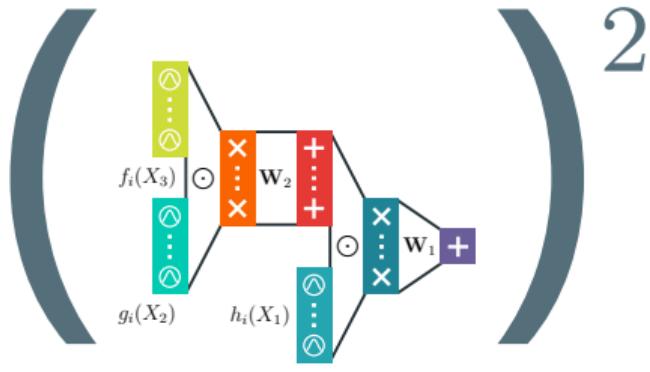
# *squaring deep PCs*

*the tensorized way*



# *squaring deep PCs*

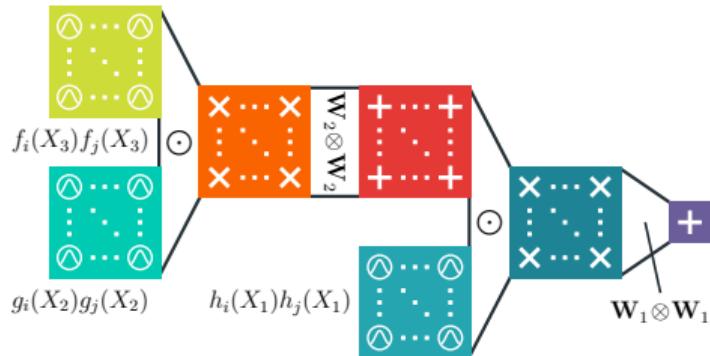
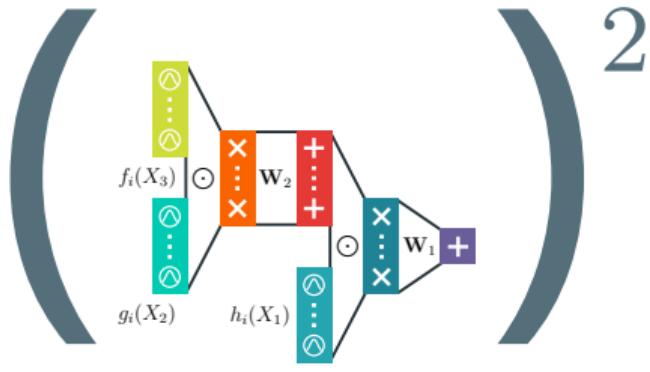
*the tensorized way*



**squaring a circuit = squaring layers**

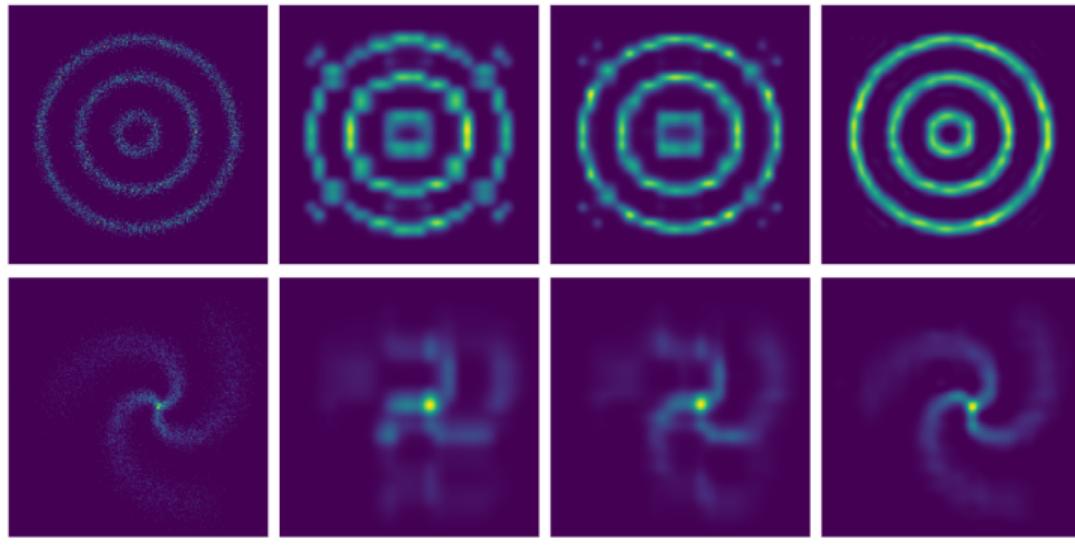
# *squaring deep PCs*

*the tensorized way*



**exactly compute**  $\int c(x)c(x)dX$  **in time**  $O(LK^2)$

# *more expressive?*



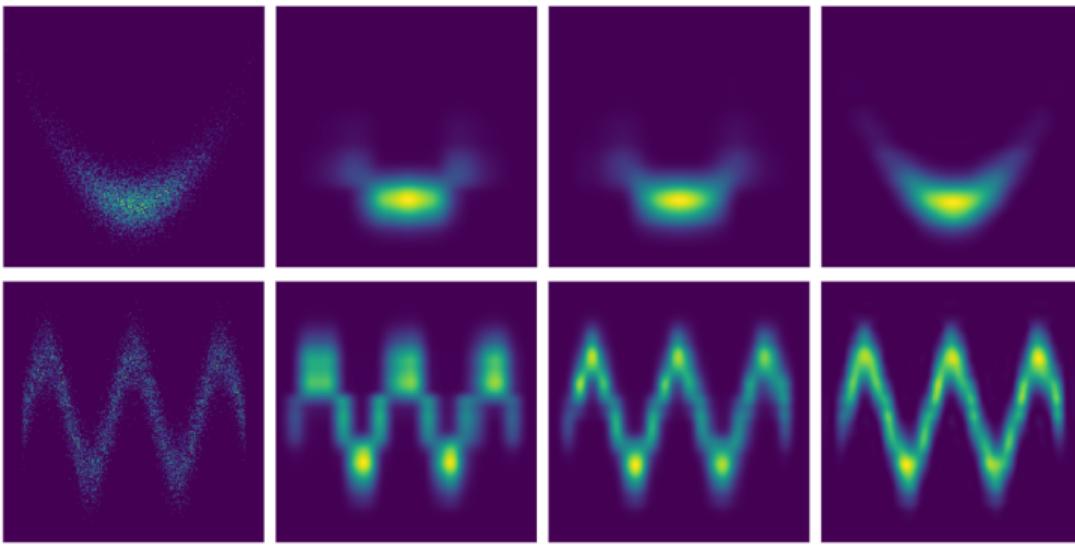
data

monoPC

monoPC<sup>2</sup>

non – monoPC<sup>2</sup>

# *more expressive?*



data

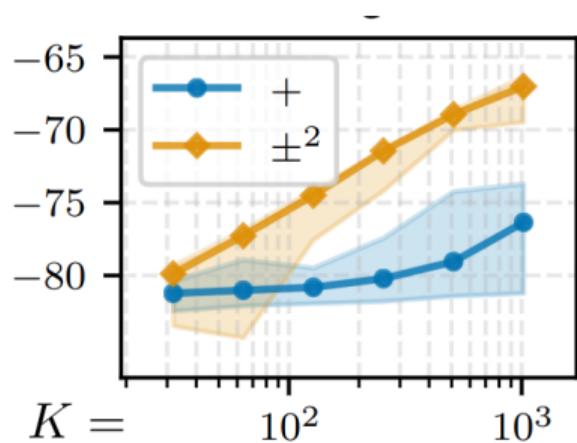
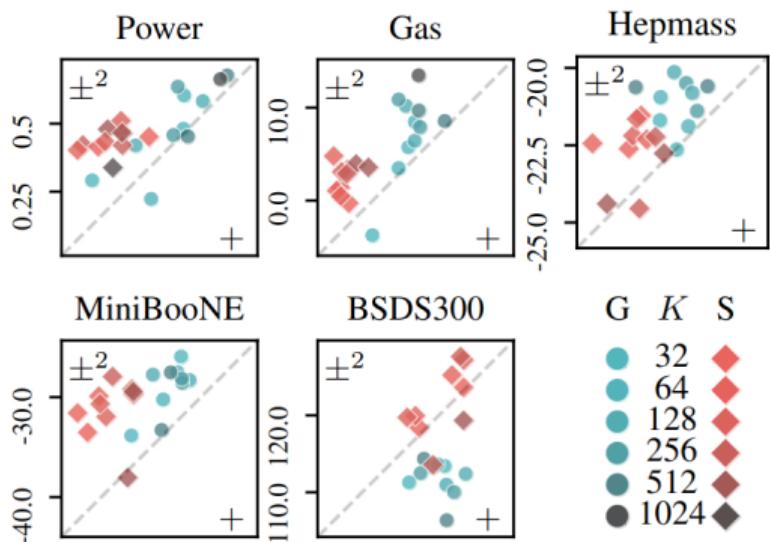
monoPC

monoPC<sup>2</sup>

non – monoPC<sup>2</sup>

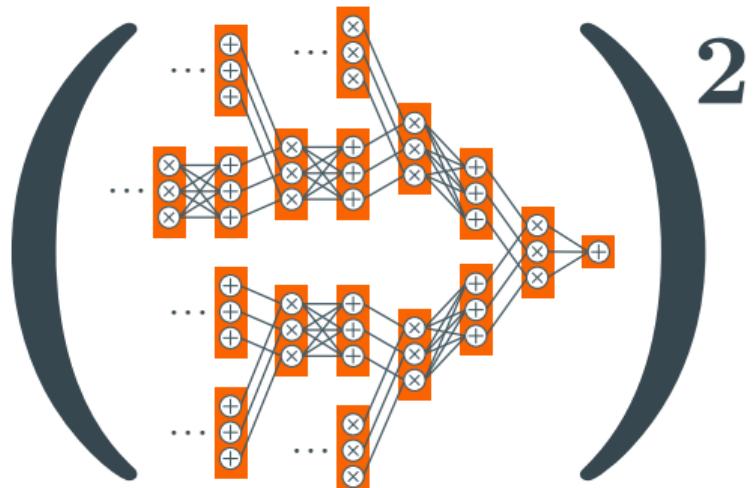
# *how more expressive?*

*real-world data*



# *theorem*

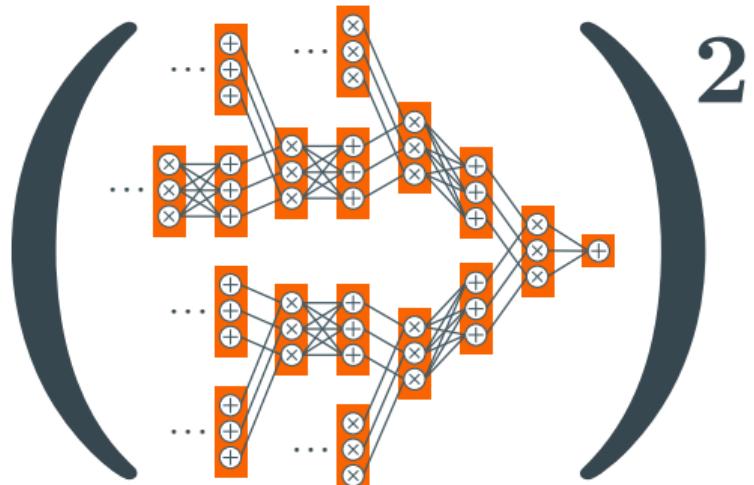
$\exists p$  requiring exponentially large  
squared non-mono circuits...

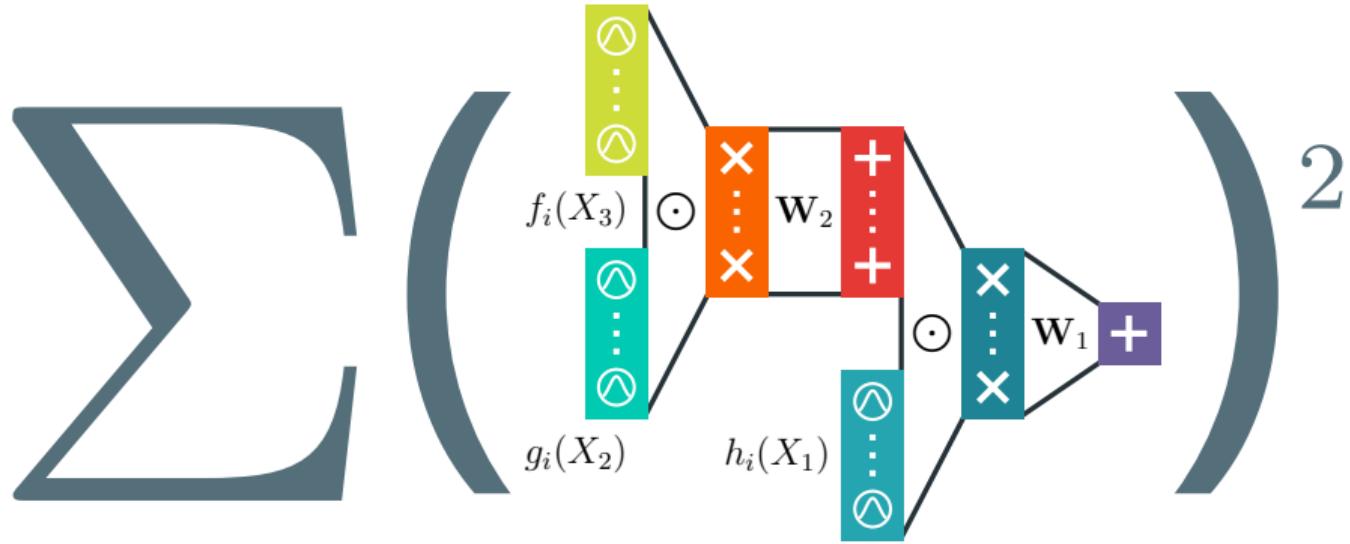


# *theorem*



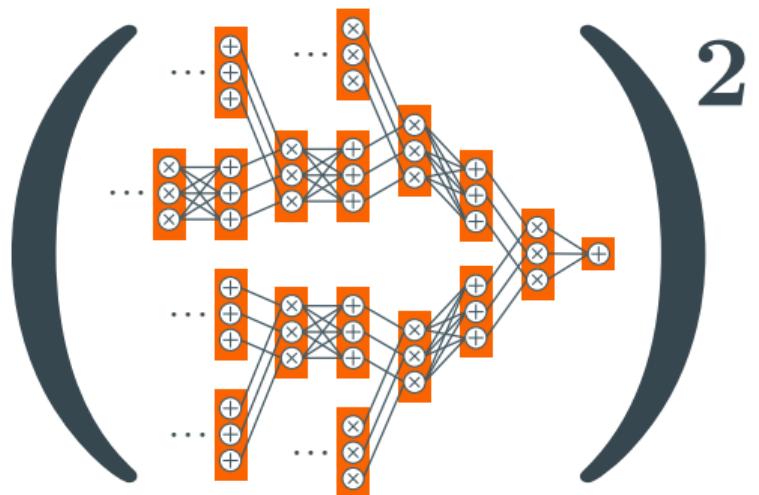
**...but compact  
monotonic circuits...!**





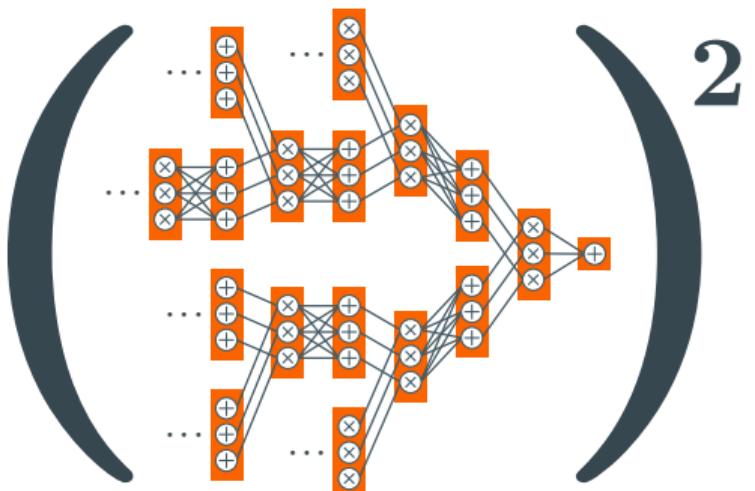
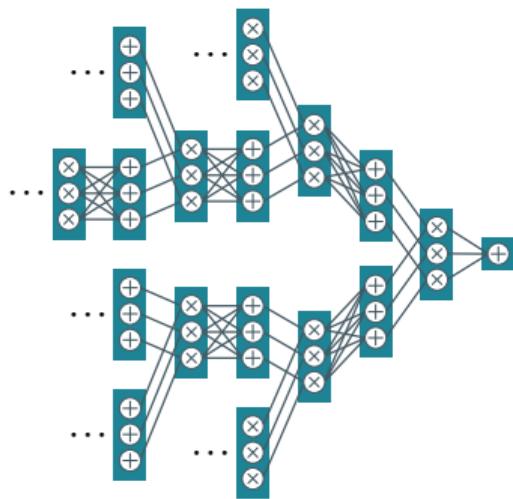
***what if we use more than one square?***

# *theorem*



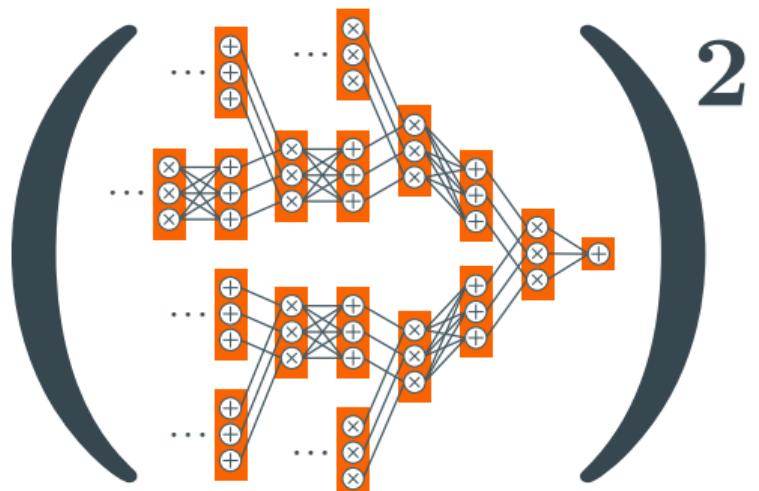
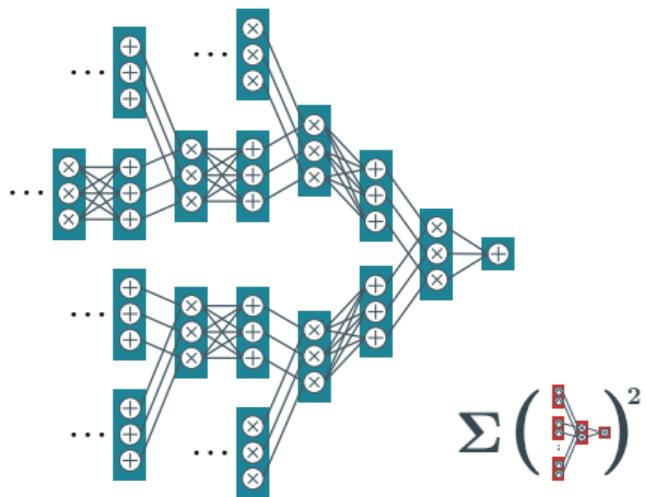
$\exists p$  requiring exponentially large **squared non-mono circuits...**

# *theorem*



**...exponentially large monotonic circuits...**

# theorem



...but compact **SOS circuits...**!

$$\pm_{\text{sd}} = \Delta \Sigma_{\text{cmp}}^2$$

(Theorem 5)

$$\Sigma_{\text{cmp}}^2 = \text{psd}$$

(Proposition 2)

$+$ <sub>sd</sub>

•  
Open Question 1

•  
Open Question 2

$$\pm_{\mathbb{R}}^2$$

• UDISJ  
(Theorem 0)

• UPS  
(Theorem 2)

• UTQ  
(Theorem B.3)

***a hierarchy of subtractive mixtures***

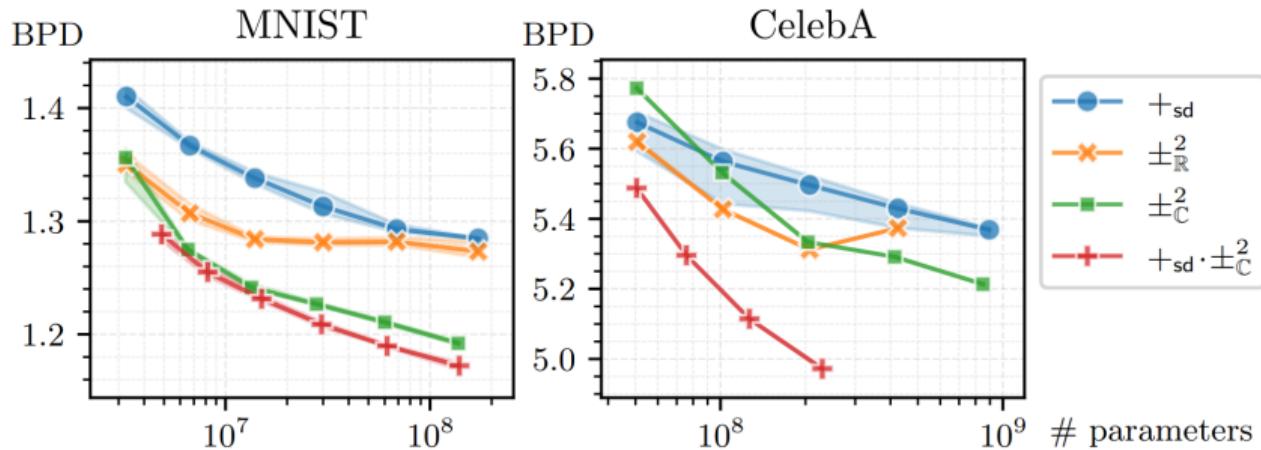
we can define circuits (and hence mixtures) over the Complex:

$$c^2(\mathbf{x}) = c(\mathbf{x})^\dagger c(\mathbf{x}), \quad c(\mathbf{x}) \in \mathbb{C}$$

and then we can note that they can be written as a SOS form

$$c^2(\mathbf{x}) = r(\mathbf{x})^2 + i(\mathbf{x})^2, \quad r(\mathbf{x}), i(\mathbf{x}) \in \mathbb{R}$$

***complex circuits are SOS (and scale better!)***



***complex circuits are SOS (and scale better!)***

*takeaway*

*“use squared mixtures  
over complex numbers  
and you get a SOS for free”*

## *takeaway*

*“use squared mixtures  
over complex numbers  
and you get a SOS for free”*

⇒    *but how to implement them?*

# *compositional inference I*



```
1 from cirkit.symbolic.functional import integrate, multiply,
2     conjugate
3
4 # create a deep circuit with complex parameters
5 c = build_symbolic_complex_circuit('quad-tree-4')
6
7 # compute the partition function of c^2
8 def renormalize(c):
9     c1 = conjugate(c)
10    c2 = multiply(c, c1)
11    return integrate(c2)
```

---

## A Quantum Information Theoretic Approach to Tractable Probabilistic Models

---

Pedro Zuidberg Dos Martires<sup>1</sup>

<sup>1</sup>Örebro University, Sweden

***negative weights without squaring***

# **approximate inference**

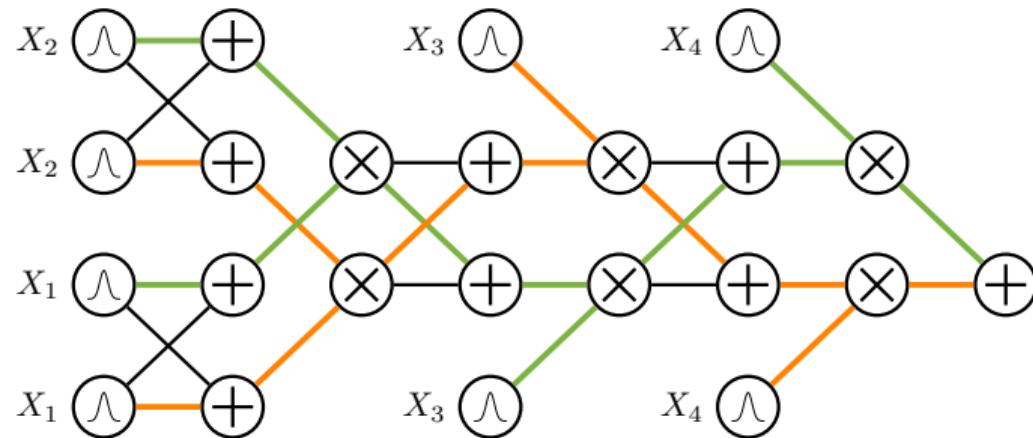
e.g., via sampling

Can we use a subtractive mixture model to approximate expectations?

$$\mathbb{E}_{\mathbf{x} \sim q(\mathbf{x})} [f(\mathbf{x})] \approx \frac{1}{S} \sum_{i=1}^S f(\mathbf{x}^{(i)}) \quad \text{with} \quad \mathbf{x}^{(i)} \sim q(\mathbf{x})$$

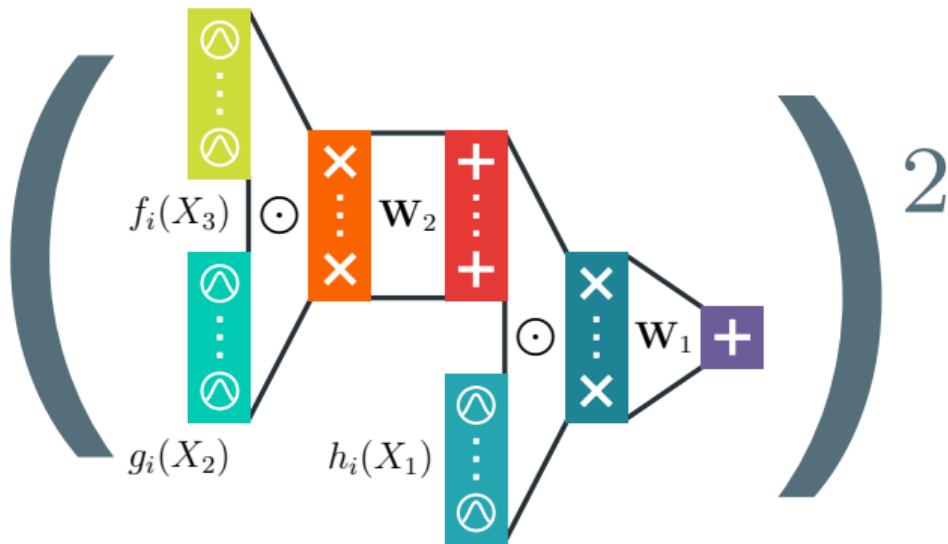
$\Rightarrow$  but how to sample from  $q$ ?

**wait...!**

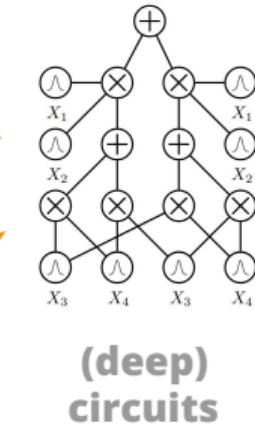
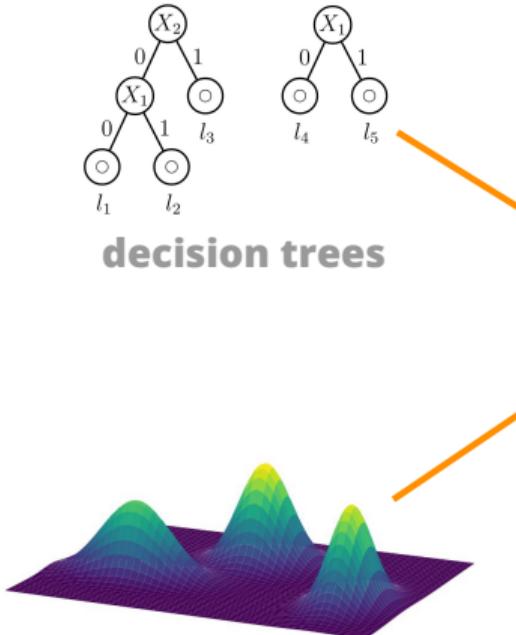


**how to sample from a *monotonic* deep PC?**

***wait...!***

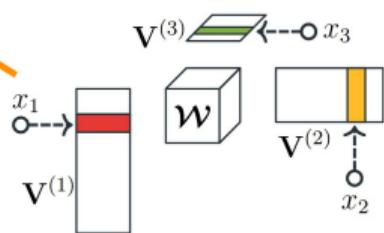


***how to sample from a **non-monotonic** deep PC?***

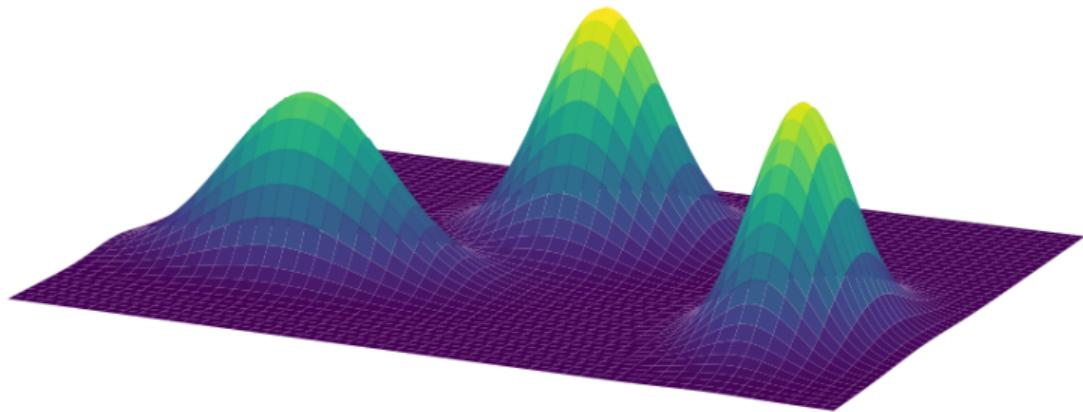


$(d \rightarrow b) \wedge (e \rightarrow b)$   
 $\vdash (\neg d \vee b) \wedge (\neg e \vee b)$   
 $\vdash b \vee (\neg d \wedge \neg e)$

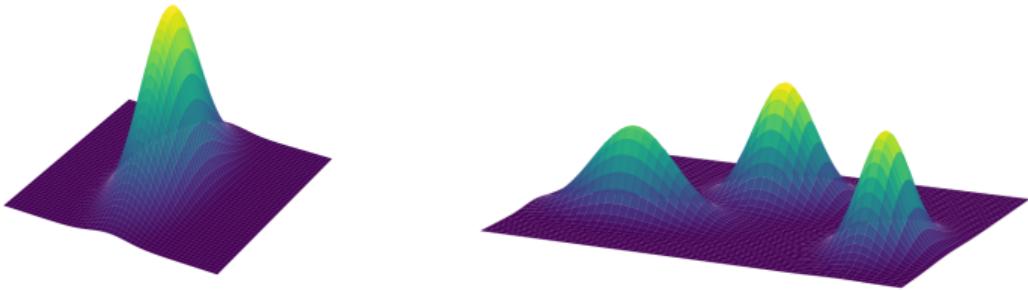
**logical formulas & constraints**



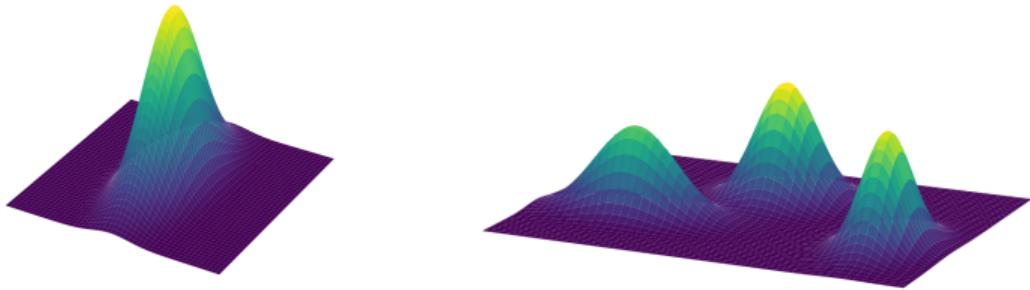
**(hierarchical) tensor factorizations**



*oh mixtures, you're so fine you blow my mind!*



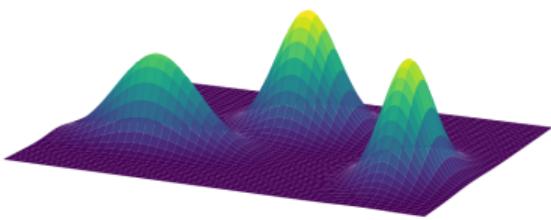
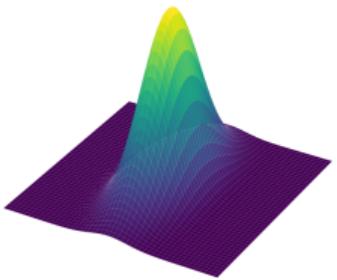
$$p(\mathbf{X}) \quad \xrightarrow{\text{orange arrow}} \quad \sum_{i=1}^K w_i p_i(\mathbf{X}) \quad w_i > 0$$



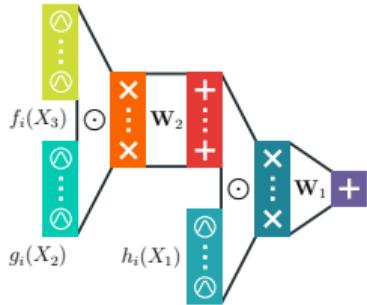
$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^K w_i p_i(\mathbf{X}) \quad w_i > 0$$

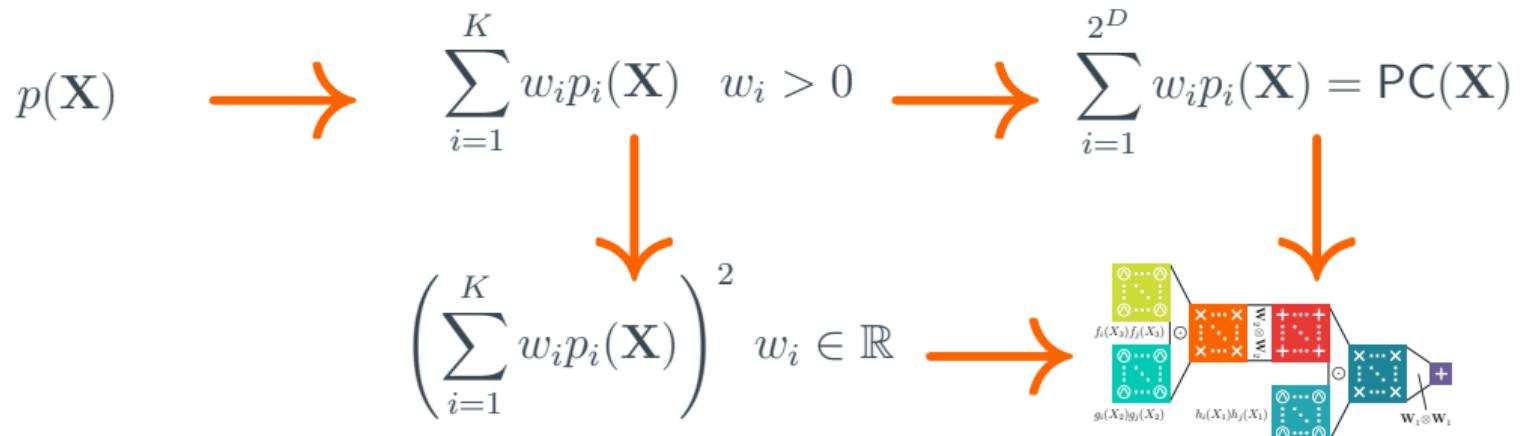
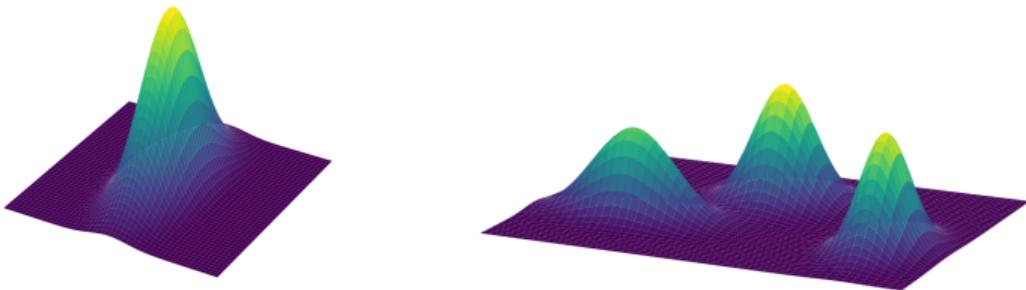
*"if someone publishes a paper on **model A**, there will be a paper about  
**mixtures of A** soon, with high probability"*

A. Vergari



$$p(\mathbf{X}) \rightarrow \sum_{i=1}^K w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \text{PC}(\mathbf{X})$$

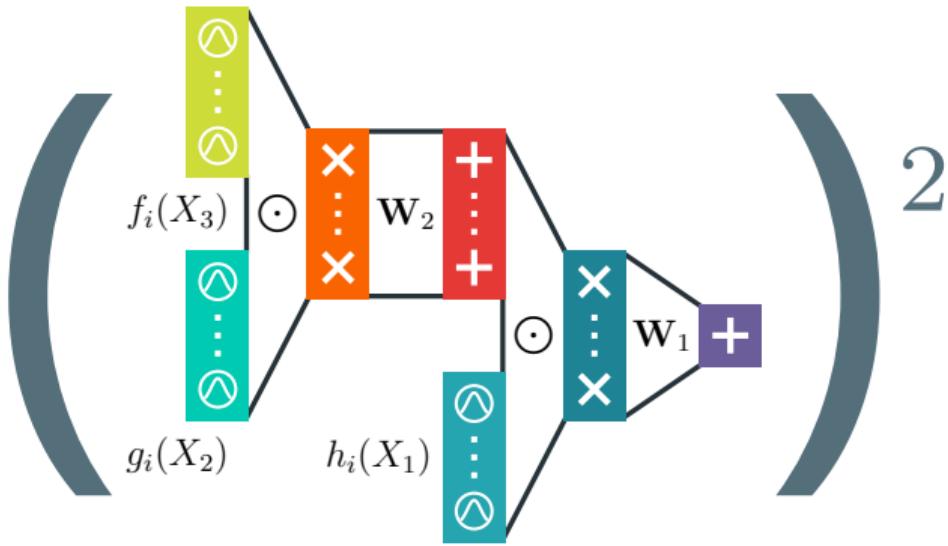






***learning & reasoning with circuits in pytorch***

[github.com/april-tools/cirkit](https://github.com/april-tools/cirkit)



**questions?**

# *structural properties*

*smoothness*

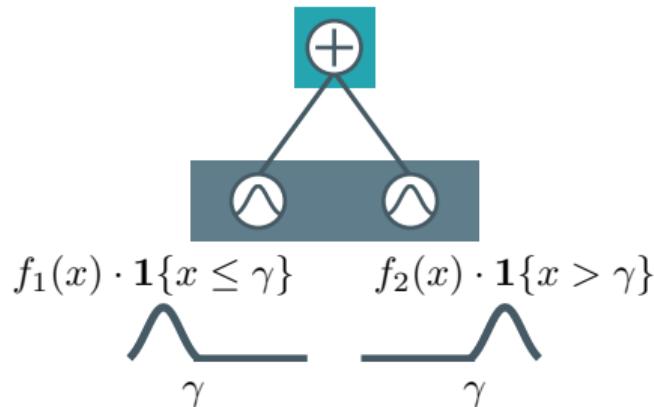
*decomposability*

*compatibility*

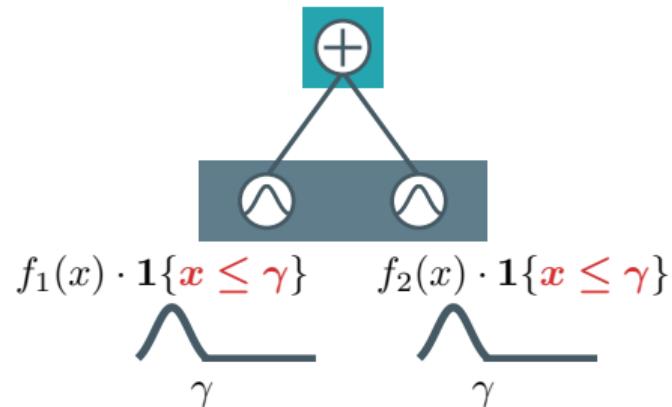
*determinism*

# determinism

the inputs of sum units are defined over disjoint supports



**deterministic circuit**



**non-deterministic circuit**