tractable probabilistic modeling with probabilistic circuits

antonio vergari (he/him)

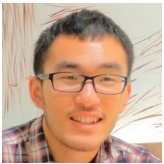
@nolovedeeplearning

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

***a**utonomous &*
***p**rovably*
***r**eliable*
***i**ntelligent*
***l**earners*

april

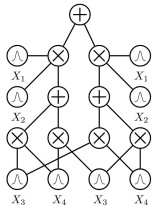
about
probabilities
integrals &
logic

april

april is probably a recursive identifier of a lab

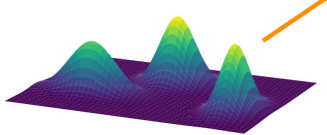
today's topics...

a language of tractable computational graphs

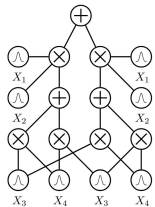


(deep)
circuits

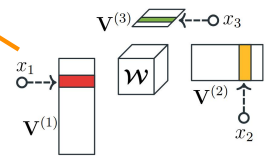
a single formalism for many models



(hierarchical)
mixtures



(deep)
circuits



(hierarchical)
tensor factorizations

wait...!

why tractability?

reasoning about ML models

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

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

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

reasoning about ML models

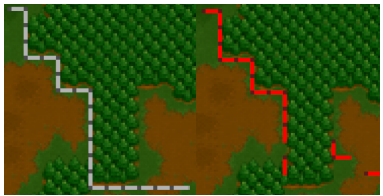
q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

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

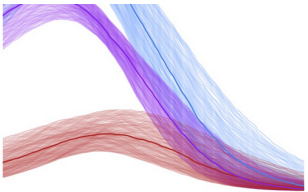
...in the language of probabilities

more complex reasoning



neuro-symbolic AI

probabilistic programming



computing uncertainties (Bayesian inference)

...and more application scenarios

reasoning about ML models

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

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

hard to compute in general!

reasoning about ML models

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

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

it is crucial we compute them exactly and in polytime!

reasoning about ML models

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

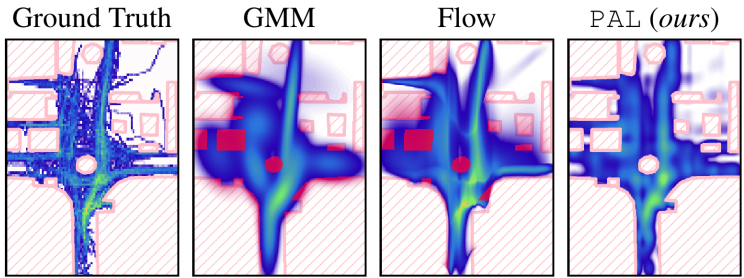
q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{x}_c | X_s = 1)} [f_1(\mathbf{x}_c)]$
(fairness)

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

it is crucial we compute them tractably!

why tractable models?

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

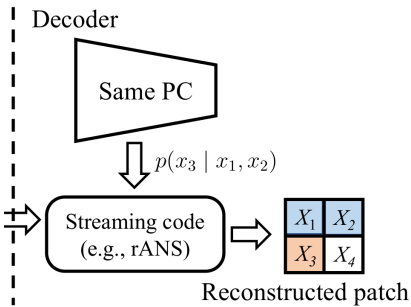


guarantee the satisfaction of given constraints

[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

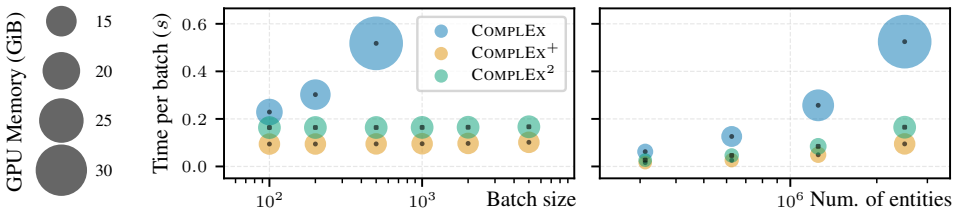
Method	MNIST (10,000 test images)		
	Theoretical bpd	Comp. bpd	En- & decoding time
PC (small)	1.26	1.30	**53**
PC (large)	**1.20**	**1.24**	168
IDF	1.90	1.96	880
BitSwap	1.27	1.31	904

[Liu, Mandt, and Broeck 2022]

why tractable models?

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

useful to scale to graph datasets with millions of entities

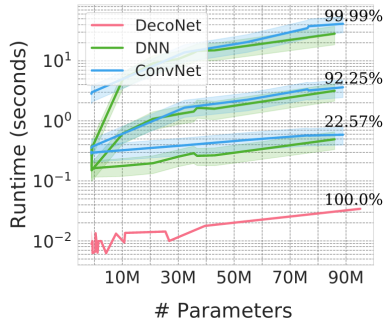


Loconte et al., "How to turn your knowledge graph embeddings into generative models", , 2023

why tractable models?

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

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



$\mathbf{q}_3$ $\mathbb{E}_{\mathbf{e}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I}_D)}\left[f(\mathbf{x}+\mathbf{e})\right]$

(adversarial robust.)

why?

tractable computations?

why?

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

VI

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

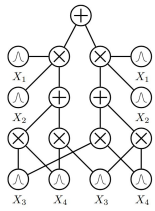
we have to pick a ***tractable*** variational distribution q...otherwise we cannot compute its density, mode, marginals later

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

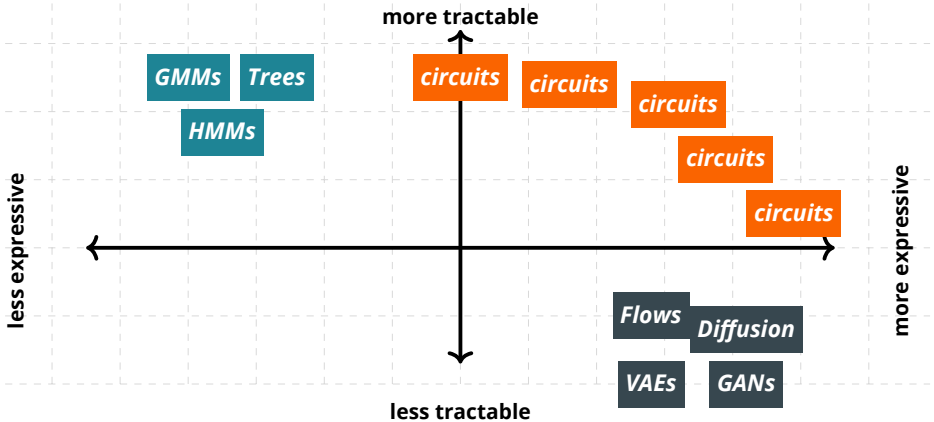
$\Rightarrow$ *then, what is the largest tractable class we can use???*

Goal

"Can we find
a middle ground
between
tractability and expressiveness?"

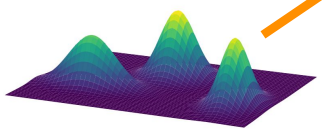


(deep) circuits

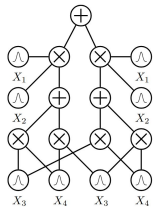


navigate the trade-off
between tractability and expressiveness

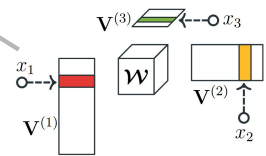
compile mixtures into circuits…



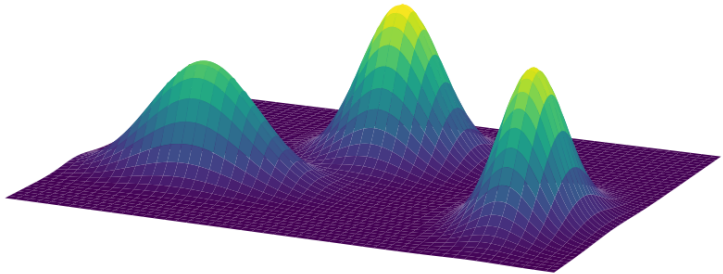
(hierarchical)
mixtures



(deep)
circuits

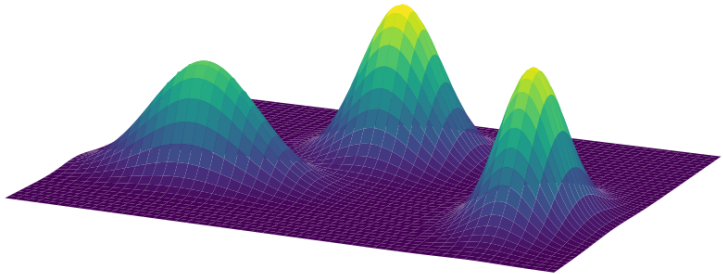


(hierarchical)
tensor factorizations



who knows mixture models?

image taken from Hao Tang's course on ASR

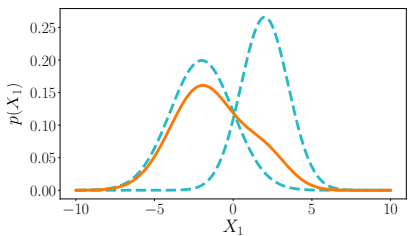


who loves mixture models?

image taken from Hao Tang's course on ASR

GMMs

as computational graphs

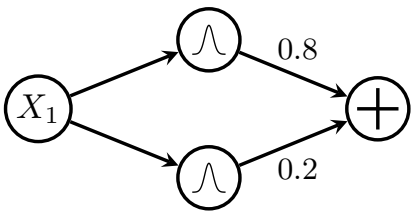


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

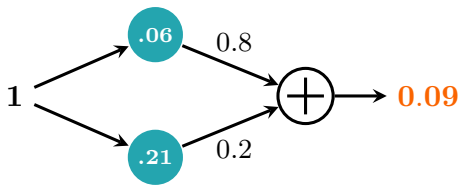


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

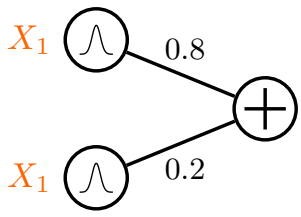


$$p(X = 1) = 0.2 \cdot p_1(X_1 = 1) + 0.8 \cdot p_2(X_1 = 1)$$

$\Rightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



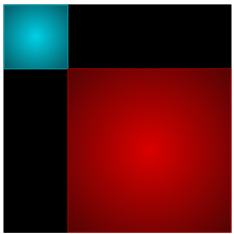
A simplified notation:

⇒ ***scopes*** *attached to inputs*

⇒ *edge directions omitted*

GMMs

as computational graphs

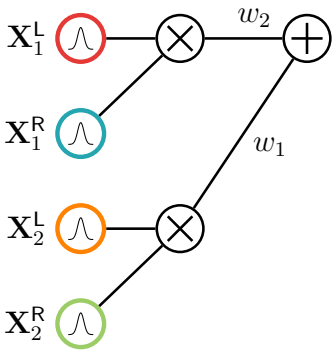


$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}_1^{\mathsf{L}}) \cdot p_1(\mathbf{X}_1^{\mathsf{R}}) + w_2 \cdot p_2(\mathbf{X}_2^{\mathsf{L}}) \cdot p_2(\mathbf{X}_2^{\mathsf{R}})$$

$\Longrightarrow$ *local factorizations...*

GMMs

as computational graphs



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

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

probabilistic circuits (PCs)

a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

$\Rightarrow$ *e.g., a multivariate Gaussian or small neural network*

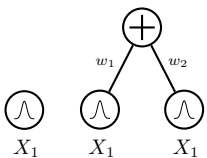
X_1

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*



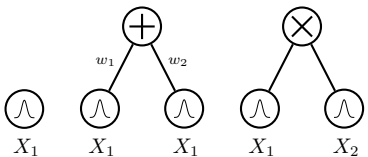
probabilistic circuits (PCs)

a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

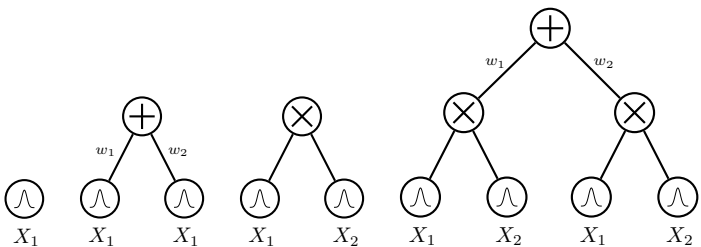
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



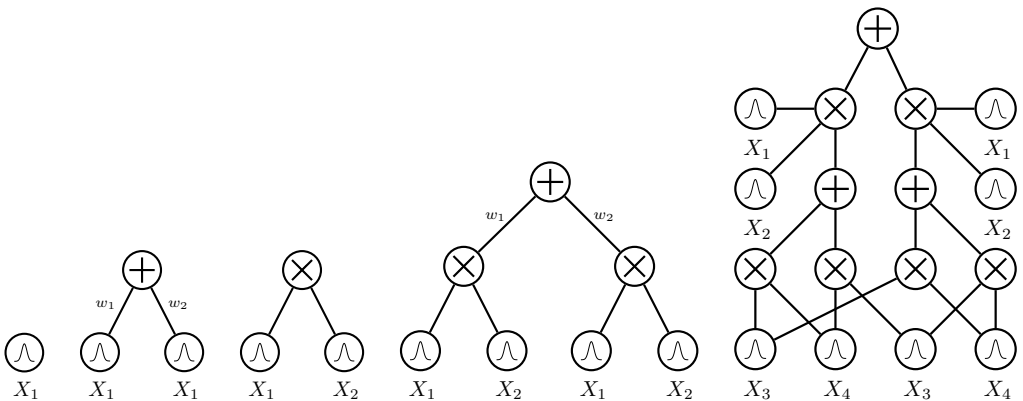
probabilistic circuits (PCs)

a grammar for tractable computational graphs



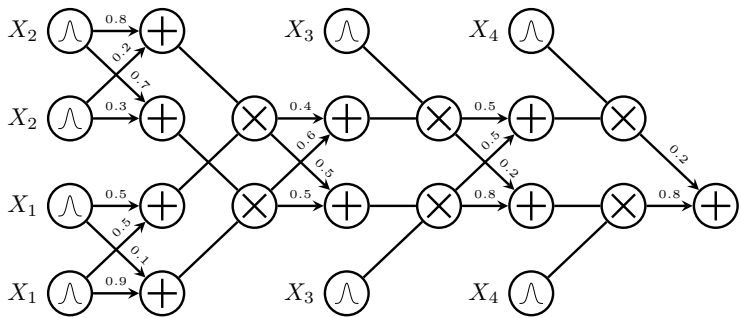
probabilistic circuits (PCs)

a grammar for tractable computational graphs



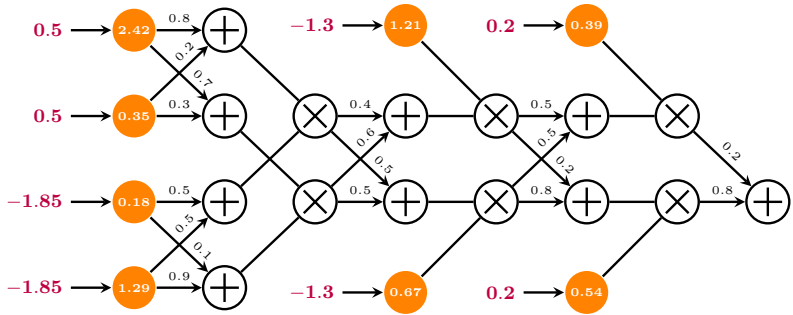
probabilistic queries = **feedforward** evaluation

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



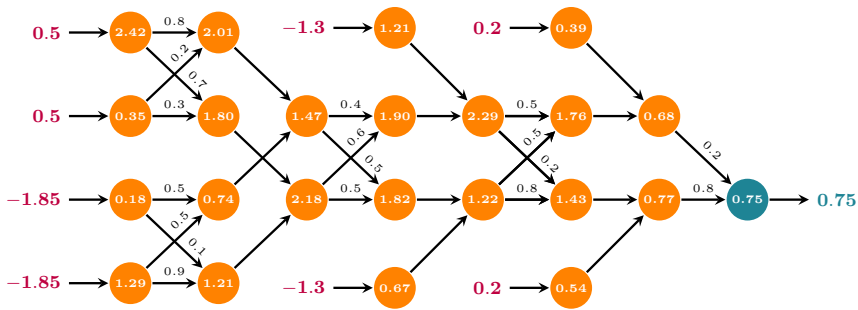
probabilistic queries = **feedforward** evaluation

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



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*

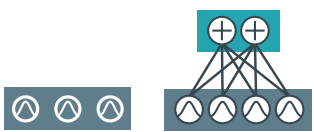
probabilistic circuits (PCs)

a tensorized definition

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

II. *A linear projection of a layer is a circuit layer*

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



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*

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

probabilistic circuits (PCs)

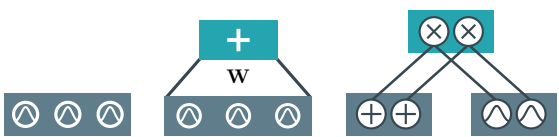
a tensorized definition

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*

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



probabilistic circuits (PCs)

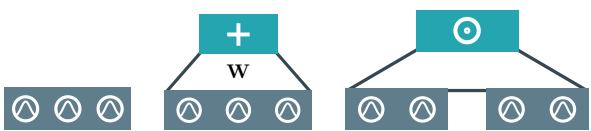
a tensorized definition

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*

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



probabilistic circuits (PCs)

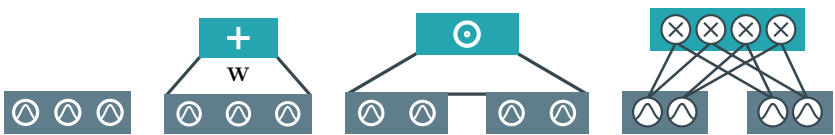
a tensorized definition

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*

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



probabilistic circuits (PCs)

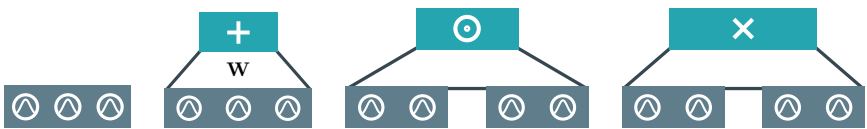
a tensorized definition

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

II. *A linear projection of a layer is a circuit layer*

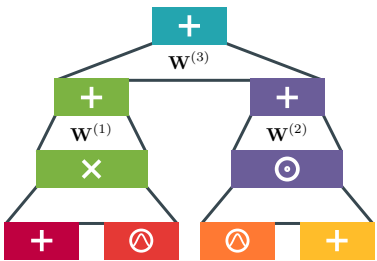
III. *The product of two layers is a circuit layer*

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



probabilistic circuits (PCs)

a tensorized definition



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

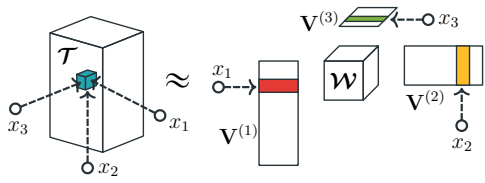
II. *A linear projection of a layer is a circuit layer*

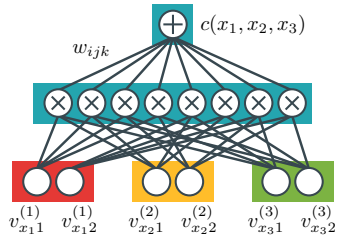
III. *The product of two layers is a circuit layer*

stack layers to build a deep circuit!

tensor factorizations

as circuits





Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", TMLR, 2025

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

$\Rightarrow$ *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

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

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., $\sum_i w_i = 1$

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

wait...!

how do we learn them?

wait...!

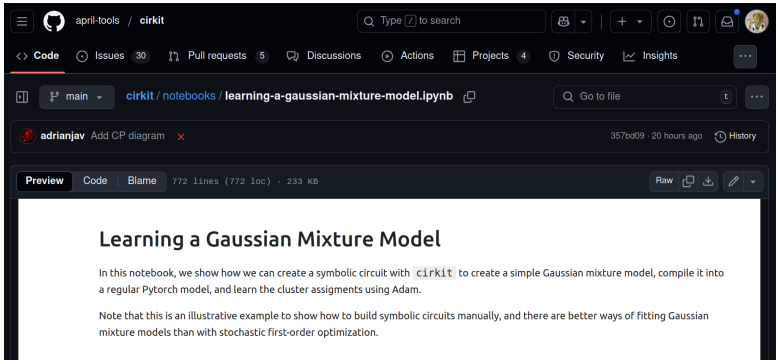
how do we learn them?

just SGD your way as usual!

$\Rightarrow$ ***or any other gradient-based optimizer***

learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`

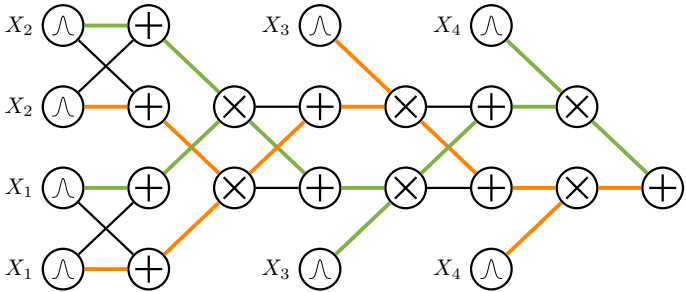


a notebook on learning GMMs as circuits

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

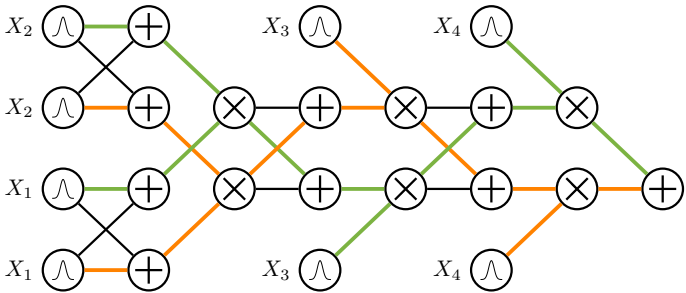
what about deep circuits?

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

$\Rightarrow$ *#HMMs #Trees #XGBoost, Tensor Networks, ...*

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic models

$\Rightarrow$ *#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

---

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

structural properties

property A

property B

property C

property D

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

---

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

structural properties

property A

property B

property C

property D

tractable computation of ***arbitrary integrals***

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

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

$\Rightarrow$ *tractable partition function*

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

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

structural properties

smoothness

decomposability

property C

property D

tractable computation of ***arbitrary integrals***

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

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

$\Rightarrow$ *tractable partition function*

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

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

structural properties

smoothness

decomposability

property C

property D

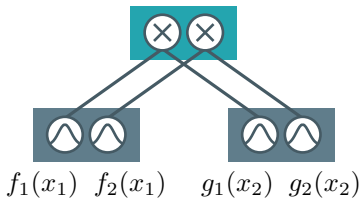
smoothness $\wedge$ decomposability $\Longrightarrow$ multilinearity

---

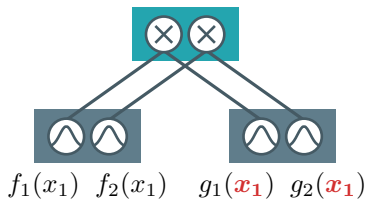
Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

multilinearity

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



✓ **multilinear**

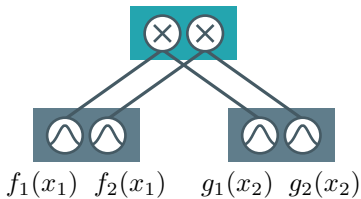


✗ **not multilinear**

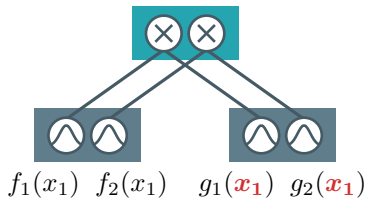
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

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



decomposable circuit

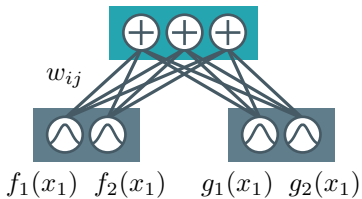


non-decomposable circuit

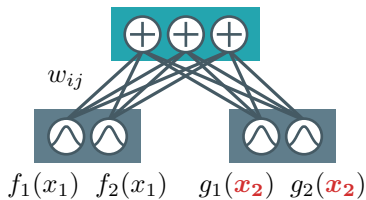
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

the inputs of sum units are defined over the same variables



✓ **multilinear**

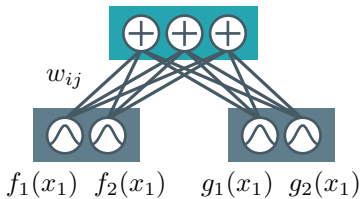


✗ **not multilinear**

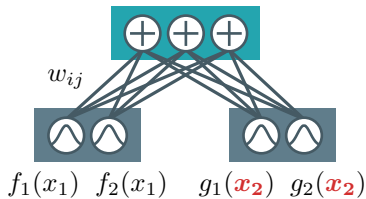
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

the inputs of sum units are defined over the same variables



smooth circuit

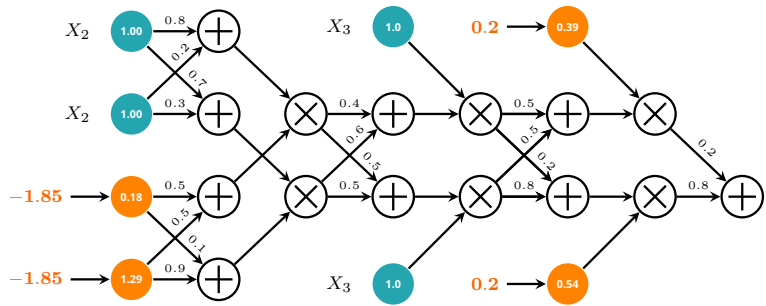


non-smooth circuit

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

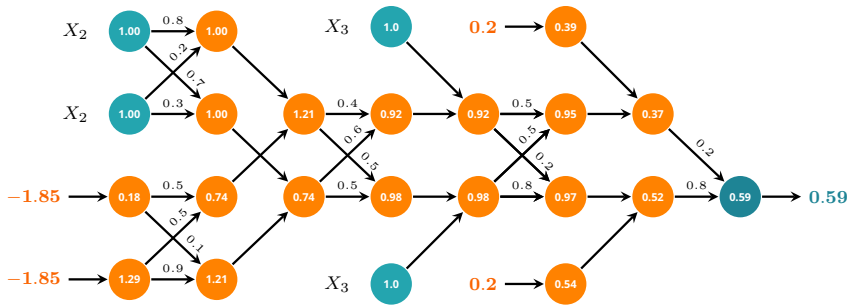
marginal queries = feedforward evaluation

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



marginal queries = feedforward evaluation

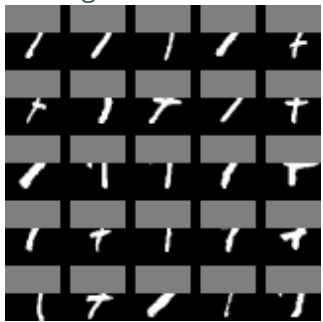
$$p(X_1 = -1.85, X_4 = 0.2)$$



tractable marginals on PCs

Original

Missing

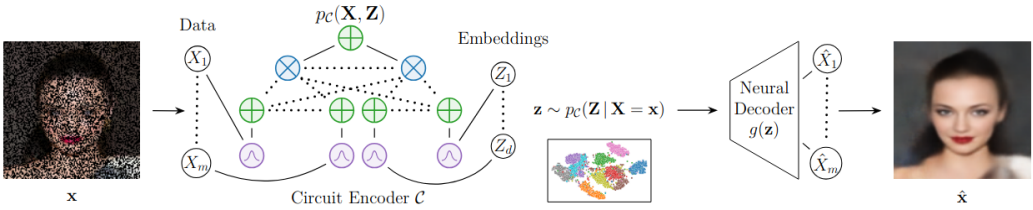


Conditional sample

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

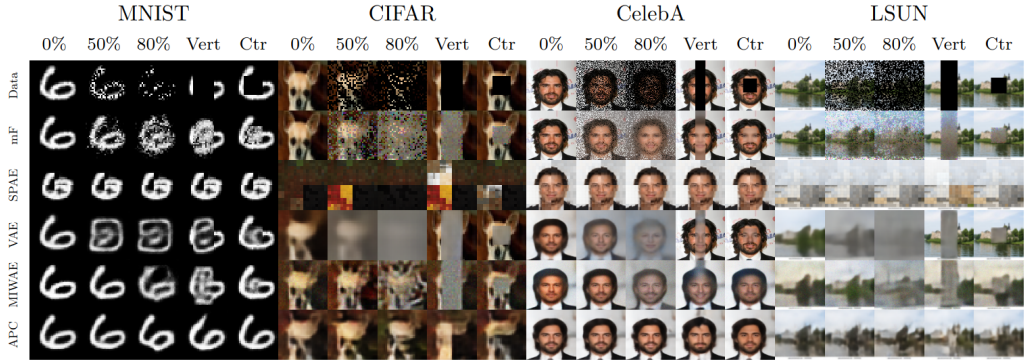
use tractable models
inside intractable pipelines
where it matters!

tractable + intractable



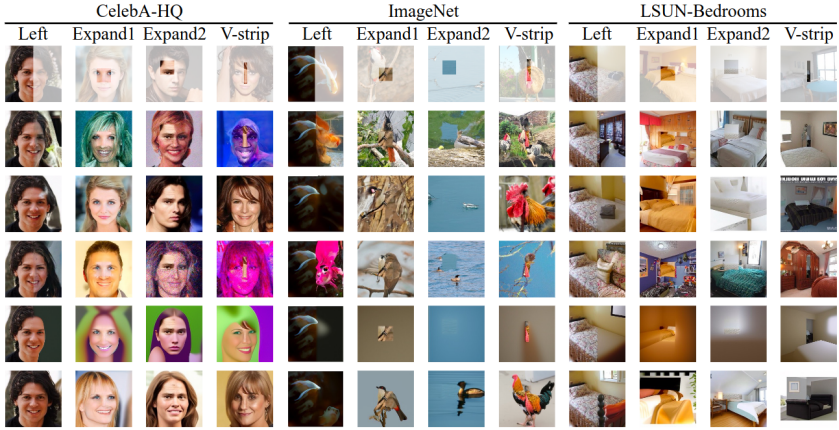
tractable conditioning over every missing mask

(under submission)



better than (V)AEs for missing values

(under submission)



Liu, Niepert, and Broeck, "Image Inpainting via Tractable Steering of Diffusion Models", The Twelfth International Conference on Learning Representations (ICLR), 2024

structural properties

smoothness

decomposability

compatibility

property D

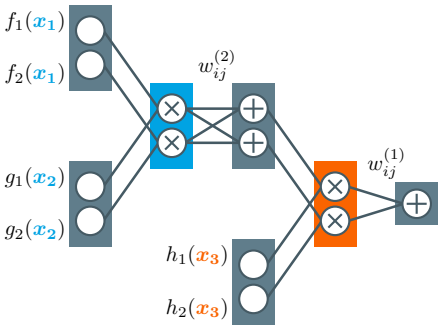
Integrals involving two or more functions:
e.g., expectations

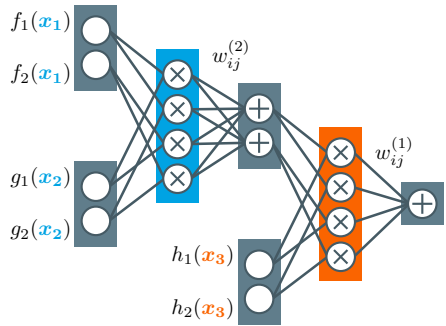
$$\mathbb{E}_{\mathbf{x}\sim p} f(\mathbf{x}) = \int p(\mathbf{x}) \; f(\mathbf{x}) \; \mathrm{d}\mathbf{x}$$

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

***Vergari** et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

compatibility

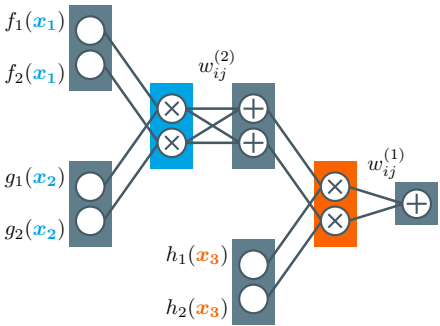


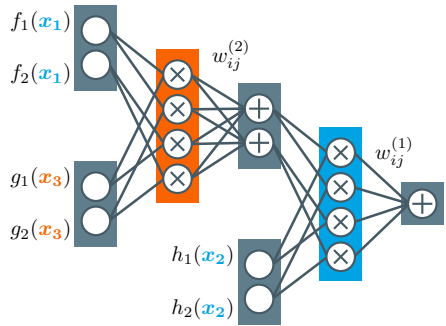


compatibile circuits

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

compatibility





non-compatibile circuits

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

structural properties

smoothness

decomposability

compatibility

property D

compatibility

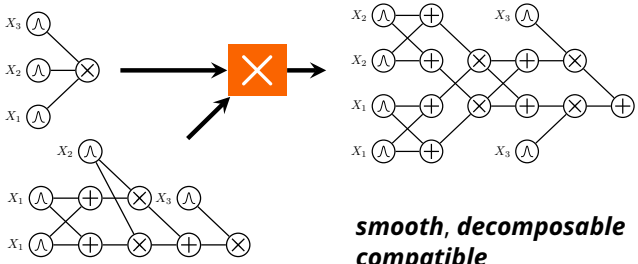
$\Downarrow$

smoothness $\wedge$ **decomposability**

compatiblity $\Rightarrow$ tractable expectations

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

tractable products



smooth, decomposable compatible

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

Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

which structural properties

for complex reasoning

q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

properties A+B

q2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$
(fairness)

property C

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

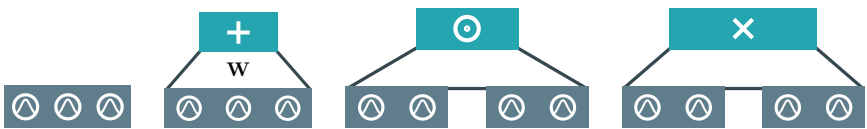
property A

```python
from cirkit.symbolic.circuit import Circuit
from cirkit.symbolic.functional import (
    integrate, multiply)

# Circuits expectation \int [p(x) f(x)]dx
def expectation(p: Circuit, f: Circuit) -> Circuit:
    i = multiply(p, f)
    return integrate(i)

# Squared loss \int [p(x)-q(x)]^2dx = E_p[p] + E_q[q] - 2E_p[q]
#             = \int p^2(x)dx + \int q^2(x)dx - 2\int p(x)q(x)dx
def squared_loss(p: Circuit, q: Circuit) -> Circuit:
    p2 = multiply(p, p)
    q2 = multiply(q, q)
    pq = multiply(p, q)
    return integrate(p2) + integrate(q2) - 2 * integrate(pq)
```

how to build deep circuits?

probabilistic circuits (PCs)

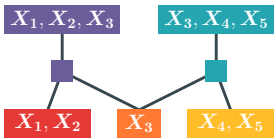
the layer-wise definition



```
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

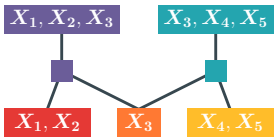
```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

learning recipe

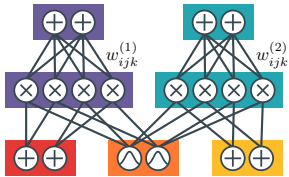


1) Build a ***region graph***

learning recipe



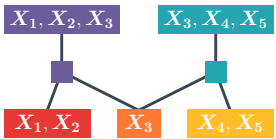
1) Build a ***region graph***

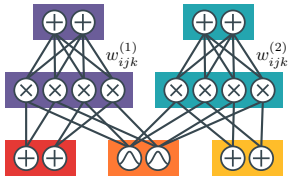


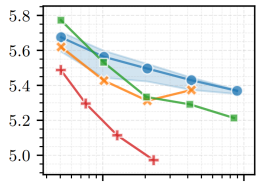
2) Overparameterize

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

learning recipe







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

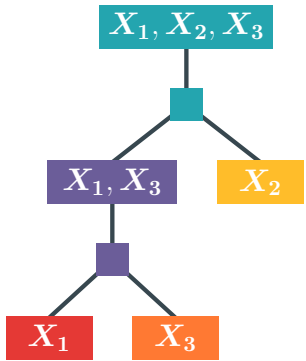
2) Overparameterize

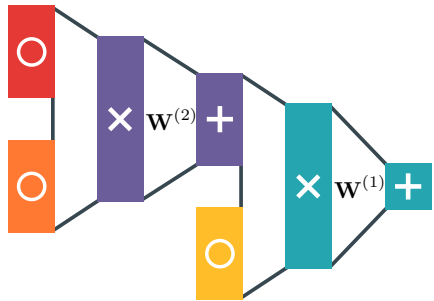
3) Learn parameters

use any optimizer in pytorch

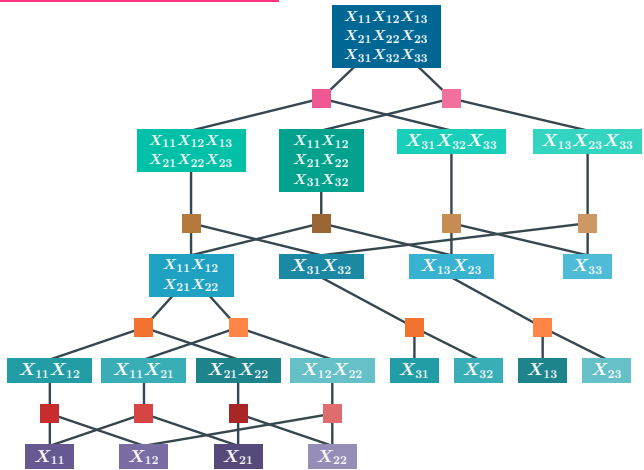
region graphs

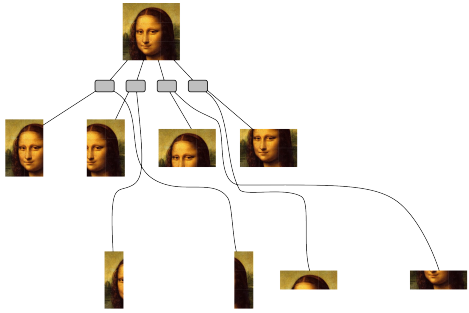
a template for smooth&decomposable PCs

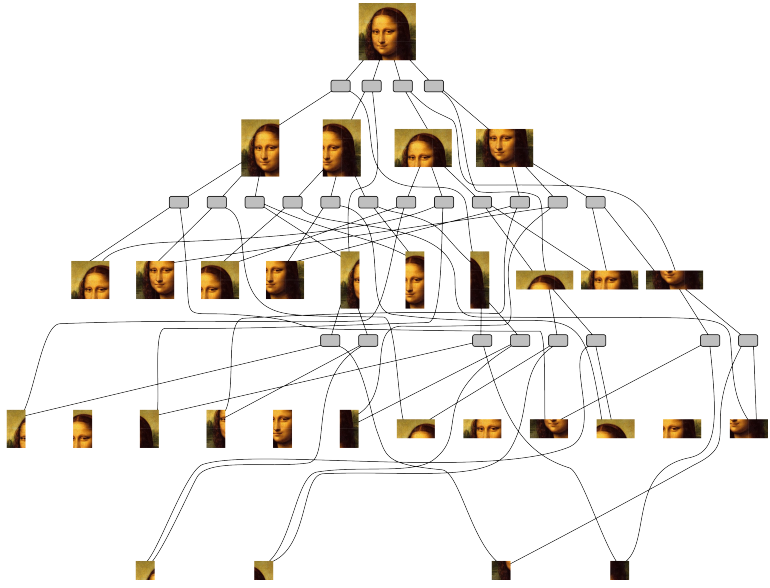


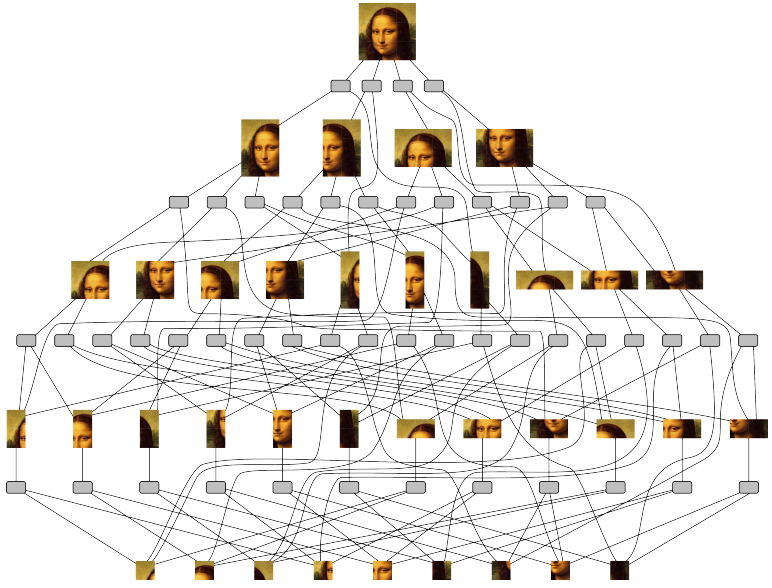


which region graph?





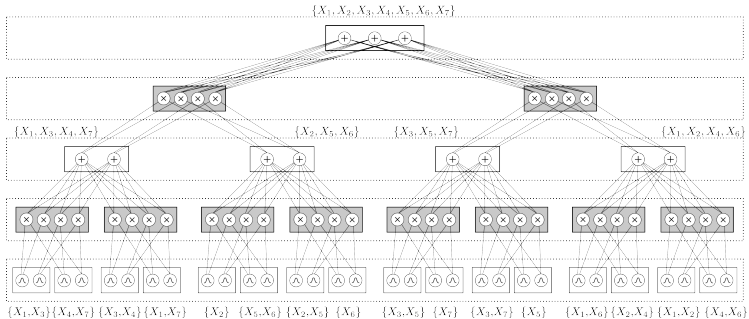




random regions graphs

The "no-learning" option

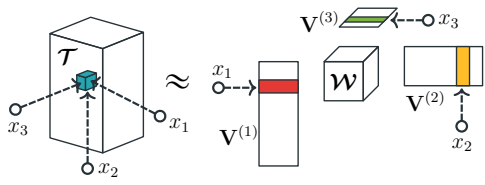
Generating a random region graph, by recursively splitting $\mathbf{X}$ into two random parts:

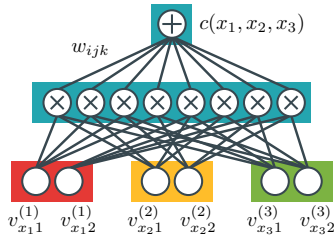


```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

circuits layers

as tensor factorizations

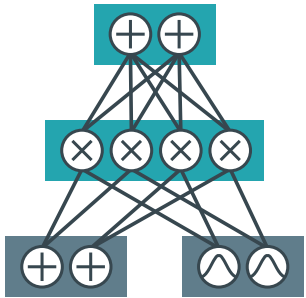


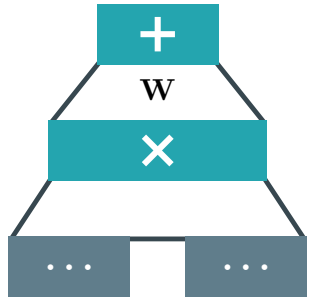


loconte2024relationship, loconte2024relationship, loconte2024relationship, loconte2024relationship

more layers

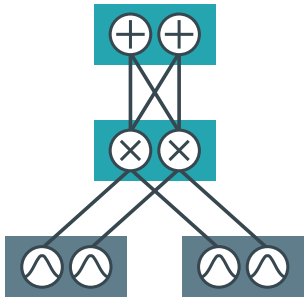
Tucker decomposition

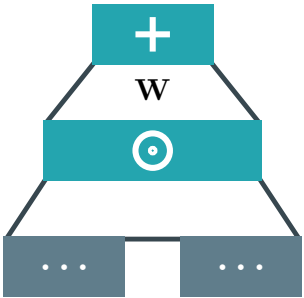




more layers

Candecomp Parafac (CP) decomposition

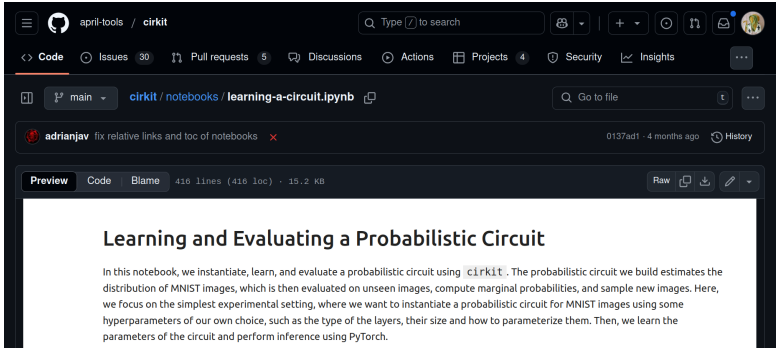




more layers

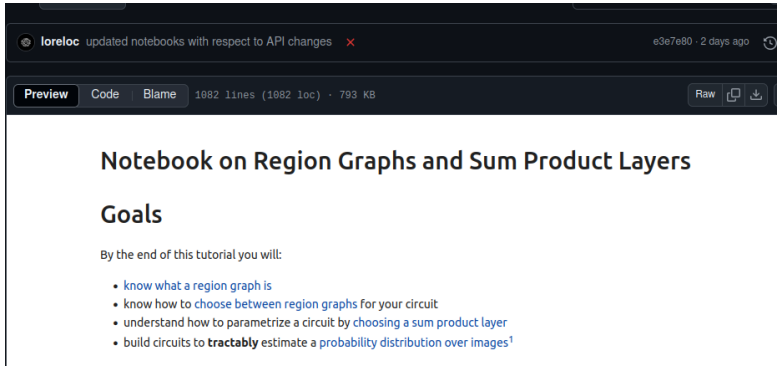
CP layer

Tucker layer



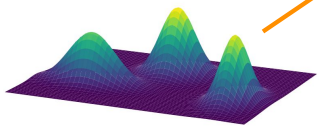
a notebook on learning a deep circuit on MNIST

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

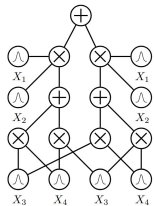


mix& match your RGs and layers

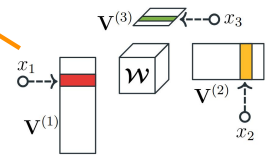
https://github.com/april-tools/cirkit/blob/main/notebooks/region-graphs-and-parametrisation.ipynb



(hierarchical)
mixtures

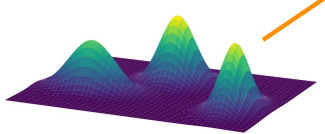


(deep)
circuits

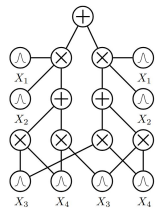


(hierarchical)
tensor factorizations

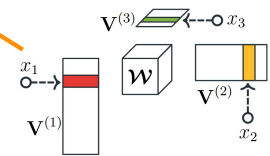
can we represent also non-probabilistic models?...



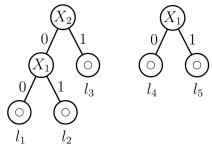
(hierarchical)
mixtures



(deep)
circuits



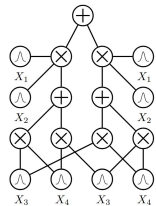
(hierarchical)
tensor factorizations



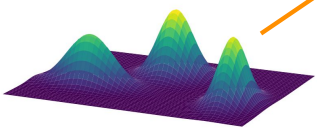
decision trees

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

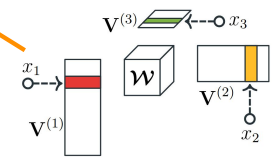
logical formulas & constraints



(deep) circuits



(hierarchical) mixtures



(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?

Ground Truth

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

nesy structured output prediction (SOP) tasks

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

When?

Ground Truth

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

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

nesy structured output prediction (SOP) tasks

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

When?

Ground Truth

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

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

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

nesy structured output prediction (SOP) tasks

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

When?

Ground Truth

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

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

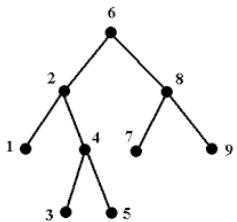
s.t. $\mathbf{y} \models \mathsf{K}$ *// e.g., that form a valid path*

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

nesy structured output prediction (SOP) tasks

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

When?



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

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

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

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

hierarchical multi-label classification

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

When?

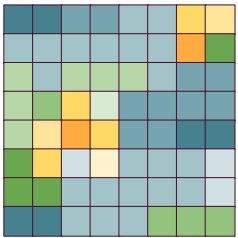
given $\mathbf{x}$ *// e.g. a user preference over $K - N$ sushi types*

find $\mathbf{y}^* = \text{argmax}_{\mathbf{y}}\, p_\theta(\mathbf{y} \mid \mathbf{x})$ *// e.g. prefs over N more types*

s.t. $\mathbf{y} \models \mathsf{K}$ *// e.g., output valid rankings*

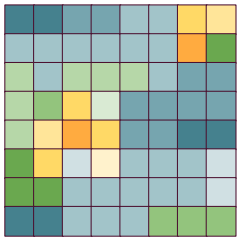
user preference learning

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

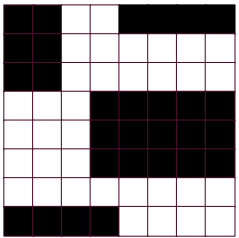


$q(\mathbf{x})$

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

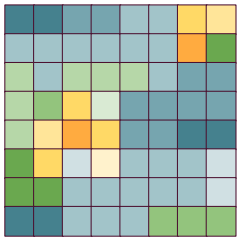


$q(\mathbf{x})$

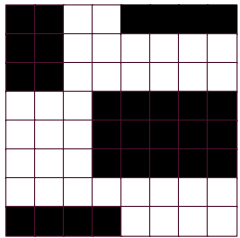


$c(\mathbf{x})$

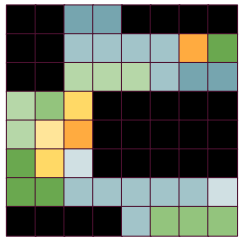
...and cut its support by a constraint $c(\mathbf{x})$



$q(\mathbf{x})$

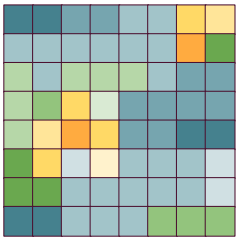


$c(\mathbf{x})$

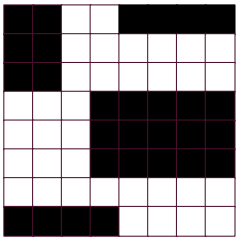


$q(\mathbf{x}) \cdot c(\mathbf{x})$

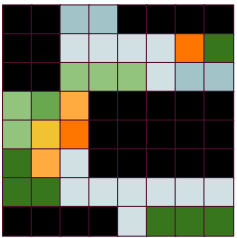
by multiplying them $q(\mathbf{x})c(\mathbf{x})$...



$q(\mathbf{x})$

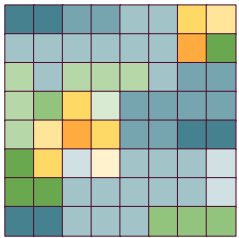


$c(\mathbf{x})$

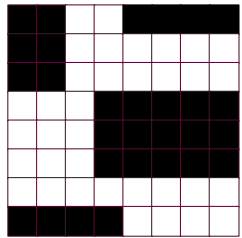


$$\frac{q(\mathbf{x}) \cdot c(\mathbf{x})}{\sum_{\mathbf{x}} q(\mathbf{x}) \cdot c(\mathbf{x})}$$

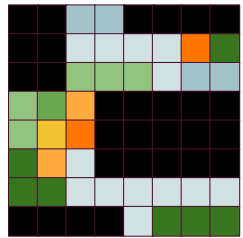
and then renormalizing them!



$q(\mathbf{x})$



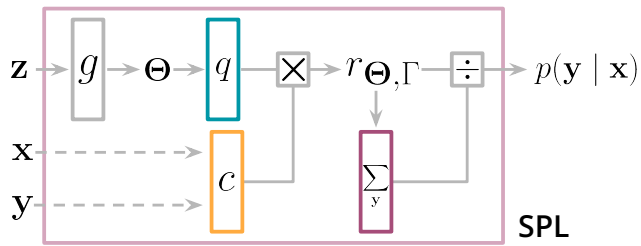
$c(\mathbf{x})$



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

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

SPL



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

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

SPL

Ground Truth

ResNet-18

Semantic Loss

circuits

predictions guarantee a logical constraint 100% of the time!

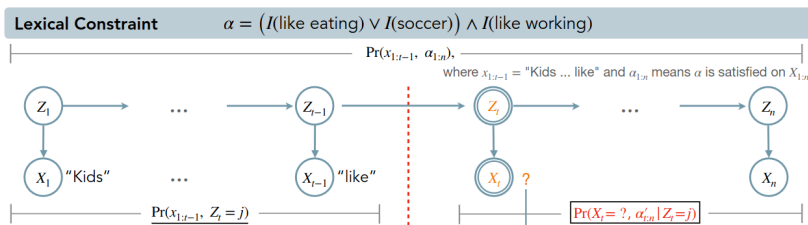
SPL

(and variants)

everywhere

Tractable Control for Autoregressive Language Generation

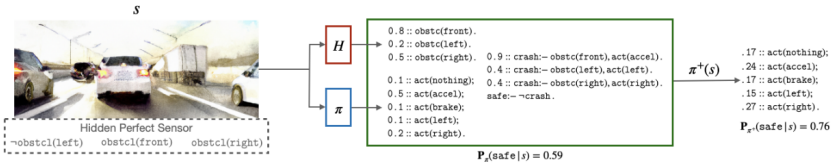
Honghua Zhang [*1] Meihua Dang [*1] Nanyun Peng [1] Guy Van den Broeck [1]



constrained text generation with LLMs (ICML 2023)

Safe Reinforcement Learning via Probabilistic Logic Shields

Wen-Chi Yang[1] , **Giuseppe Marra**[1] , **Gavin Rens** and **Luc De Raedt**[1,2]



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

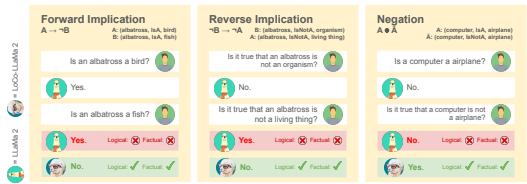
Nicola Di Mauro
University of Bari, Italy
nicola.dimauro@uniba.it

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

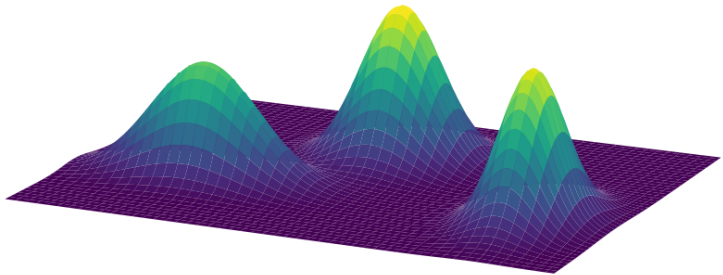
Antonio Vergari
University of Edinburgh, UK
avergari@ed.ac.uk

enforce constraints in knowledge graph embeddings
oral at NeurIPS 2023

Logically Consistent Language Models via Neuro-Symbolic Integration



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

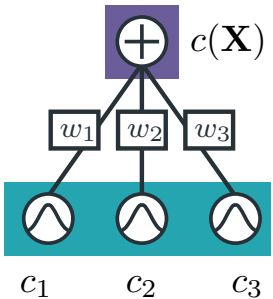


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

additive MMs

are so cool!



easily represented as shallow PCs

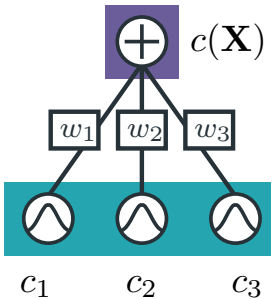
these are **monotonic** PCs

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

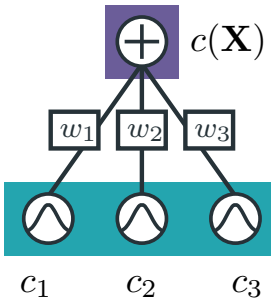
these are **monotonic** PCs

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

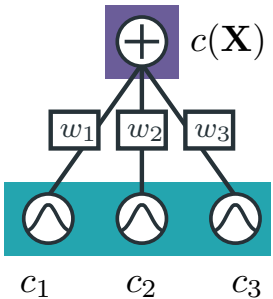
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



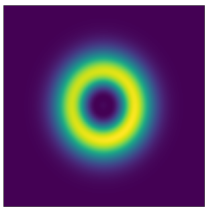
easily represented as shallow PCs

these are **monotonic** PCs

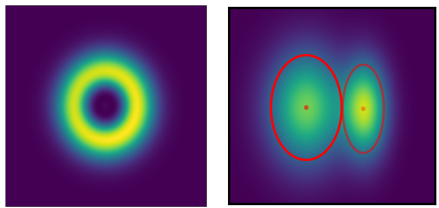
if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

however...

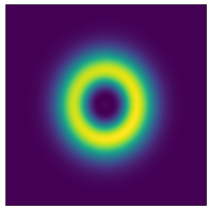


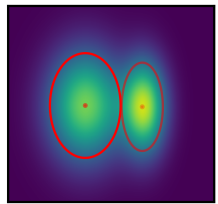
however...



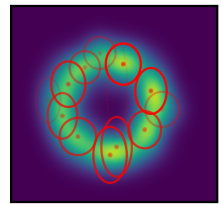
GMM ($K = 2$)

however...



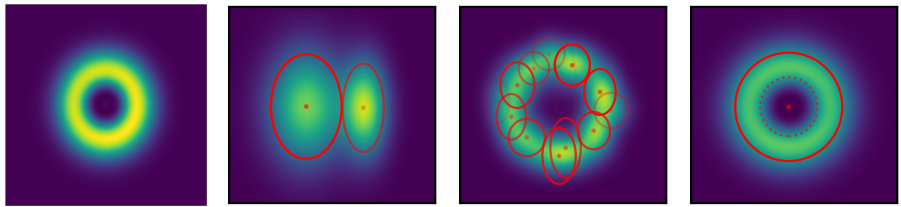


GMM ($K = 2$)



GMM ($K = 16$)

however...



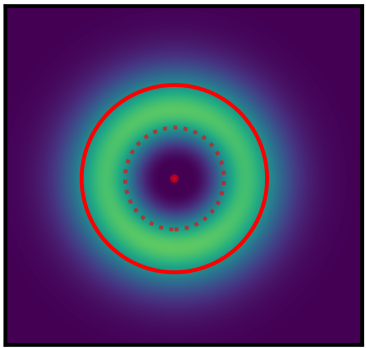
GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

spoiler

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

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

subtractive MMs



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

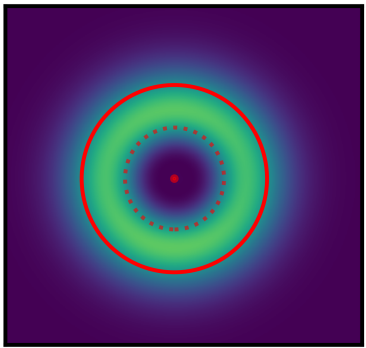
$\Rightarrow$ *or **non-monotonic** circuits,...*

issue: how to preserve non-negative outputs?

well understood for simple parametric forms
e.g., Weibulls, Gaussians

$\Rightarrow$ *constraints on variance, mean*

subtractive MMs



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

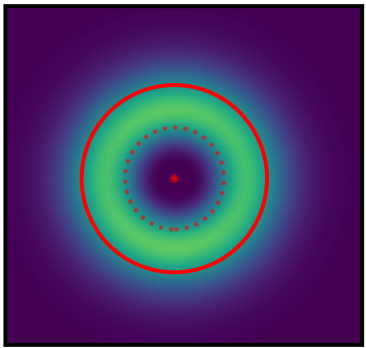
$\Rightarrow$ *or **non-monotonic** circuits,...*

issue: how to preserve non-negative outputs?

well understood for simple parametric forms
e.g., Weibulls, Gaussians

$\Rightarrow$ *constraints on variance, mean*

subtractive MMs



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

$\Rightarrow$ *or* ***non-monotonic*** *circuits,...*

issue: how to preserve non-negative outputs?

well understood for simple parametric forms
e.g., Weibulls, Gaussians

$\Rightarrow$ *constraints on variance, mean*

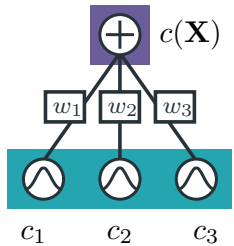
tl;dr

"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

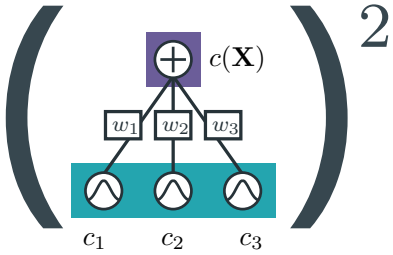


a **non-monotonic** smooth and (structured) decomposable circuit

$\Rightarrow$ *possibly with negative outputs*

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

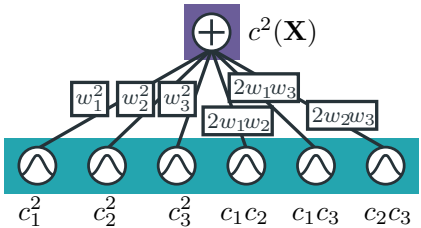
squaring shallow MMs



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

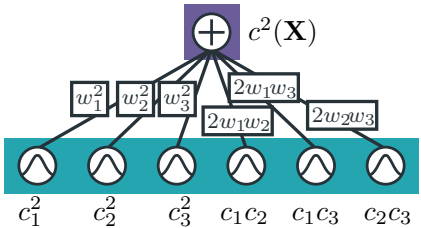
$\Rightarrow$ *ensure non-negative output*

squaring shallow MMs



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

squaring shallow MMs

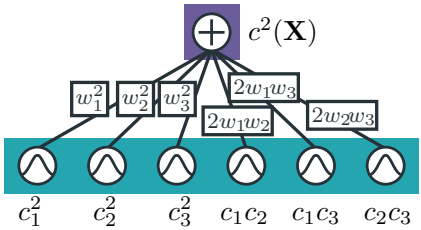


$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2$$
$$= \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})$$

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

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

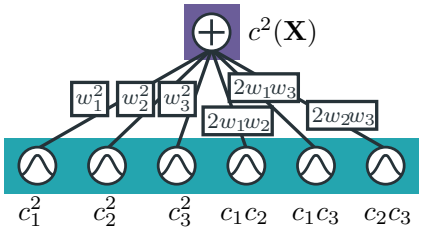
squaring shallow MMs



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

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



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

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

$\Rightarrow$ *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

Chirag Modi
Flatiron Institute
cmodi@flatironinstitute.org

Charles C. Margossian
Flatiron Institute
cmargossian@flatironinstitute.org

Robert M. Gower
Flatiron Institute
rgower@flatironinstitute.org

David M. Blei
Columbia University
david.blei@columbia.edu

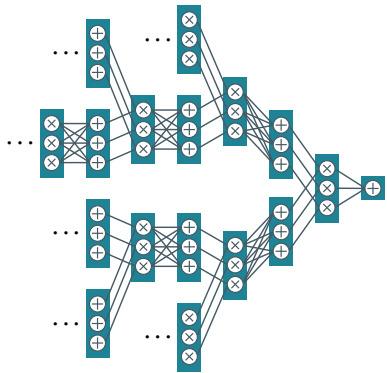
Lawrence K. Saul
Flatiron Institute
lsaul@flatironinstitute.org

orthonormal squared mixtures for VI

wait...

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

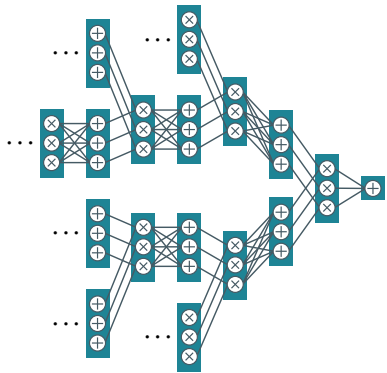
theorem

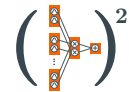


$\exists\, p$ **requiring exponentially large monotonic circuits...**

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

theorem

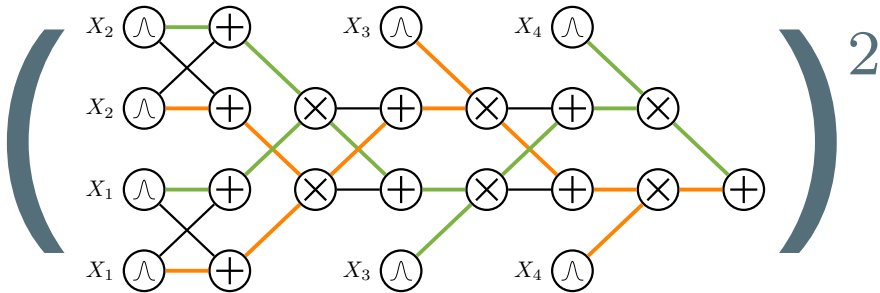




...but compact

squared non-monotonic circuits

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024



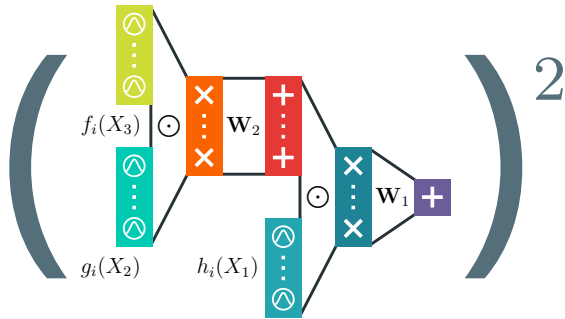
how to efficiently square (and *renormalize*) a deep PC?

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

compositional inference I

cirkit

```
from cirkit.symbolic.functional import integrate, multiply

#
# create a deep circuit
c = build_symbolic_circuit('quad-tree-4')

#
# compute the partition function of c^2
def renormalize(c):
    c2 = multiply(c, c)
    return integrate(c2)
```

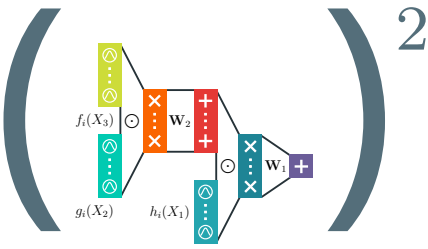


how to efficiently square (and *renormalize*) a deep PC?

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

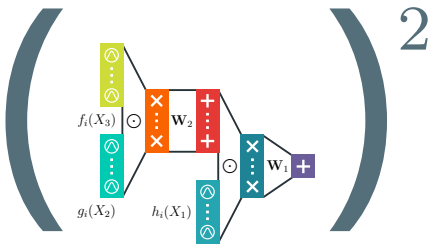
squaring deep PCs

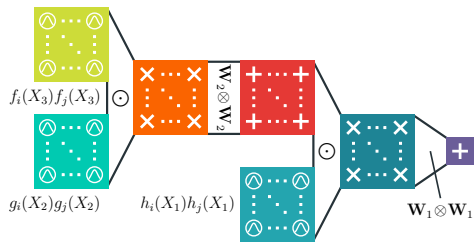
the tensorized way



squaring deep PCs

the tensorized way

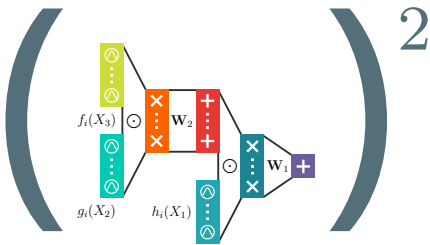


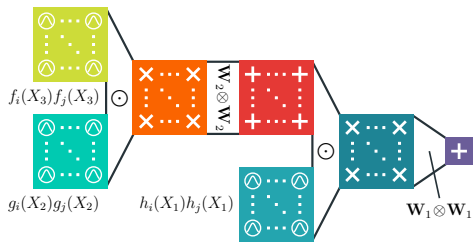


squaring a circuit = squaring layers

squaring deep PCs

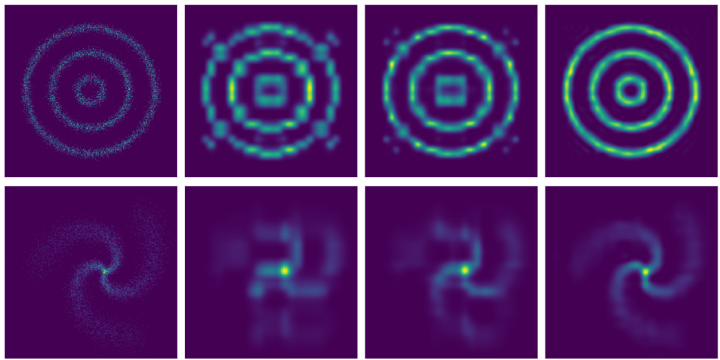
the tensorized way





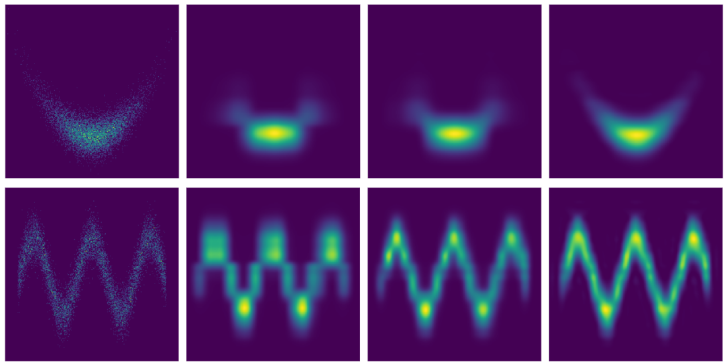
exactly compute $\int \boldsymbol{c}(\mathrm{x})\boldsymbol{c}(\mathrm{x})d\mathrm{X}$ in time $O(\boldsymbol{L}\boldsymbol{K}^2)$

more expressive?



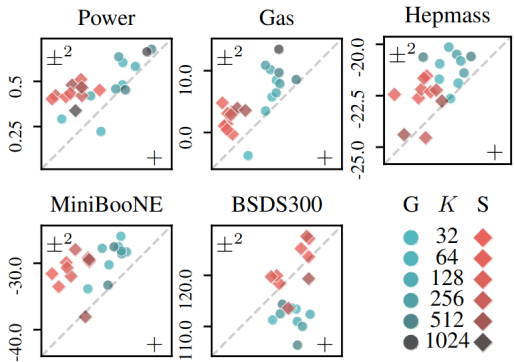
data monoPC monoPC^2 non $-$ monoPC^2

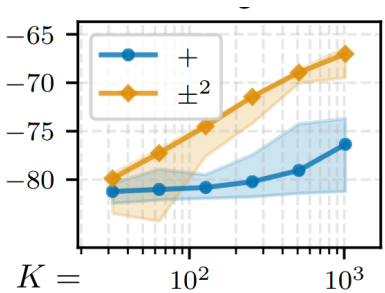
more expressive?



how more expressive?

real-world data

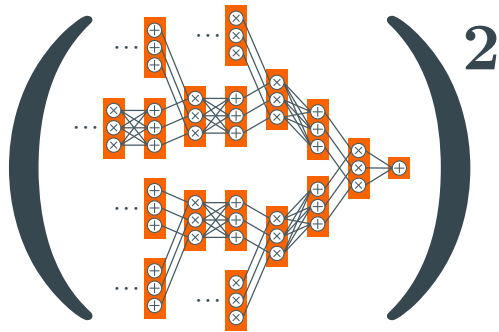




theorem

$\exists\, p$ **requiring exponentially large**

squared non-mono circuits...

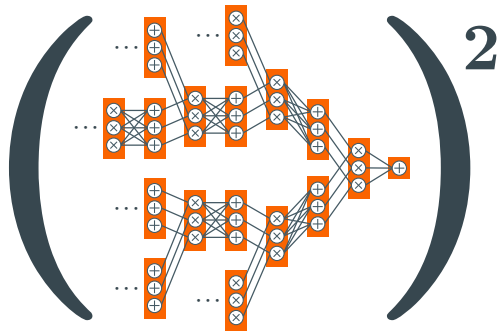


Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

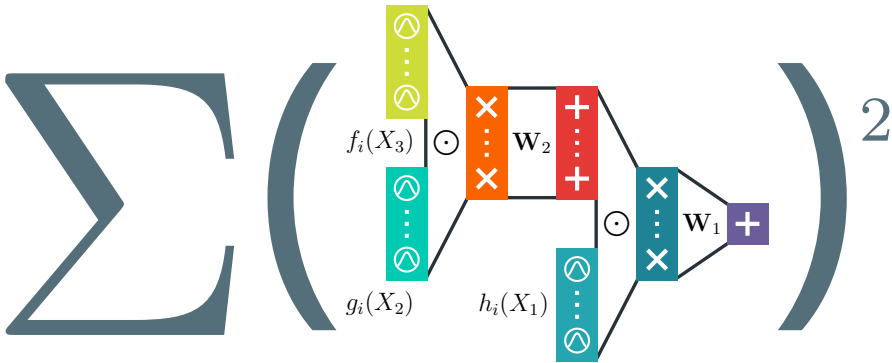
theorem

...but compact

monotonic circuits...!

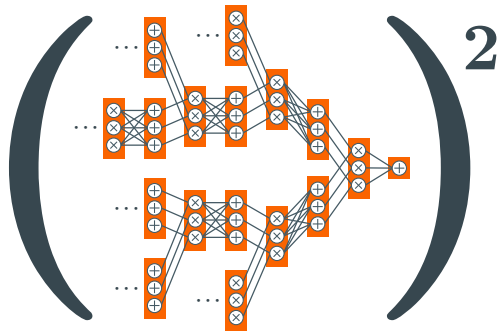


Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



what if we use more that one square?

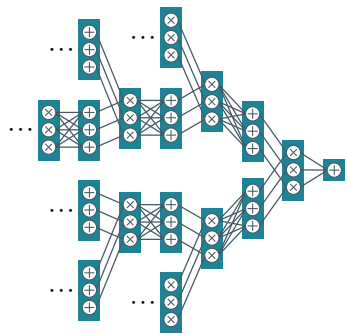
theorem



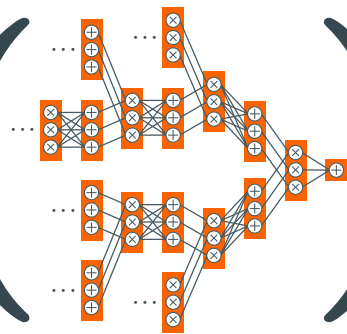
$\exists\, p$ **requiring exponentially large squared non-mono circuits...**

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem



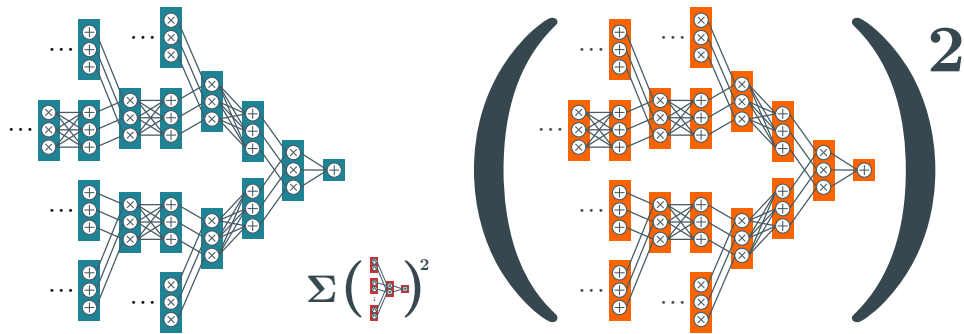
$$\left(\quad \right)^2$$



...exponentially large monotonic circuits...

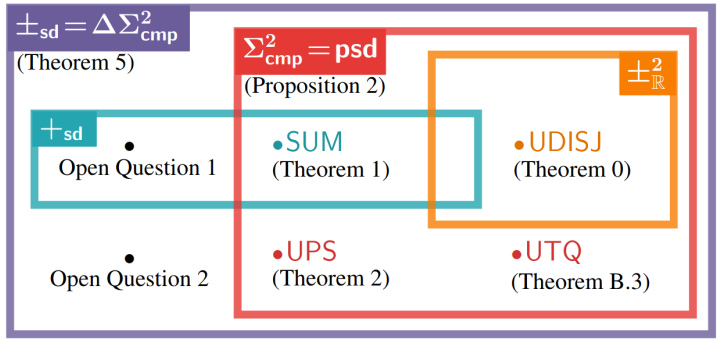
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem



...but compact SOS circuits...!

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



a hierarchy of subtractive mixtures

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

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

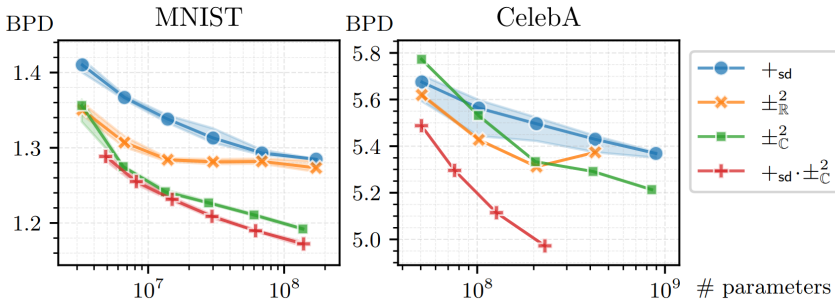
$$c^2(\mathbf{x}) = c(\mathbf{x})^\dagger c(\mathbf{x}), \qquad c(\mathbf{x}) \in \mathbb{C}$$

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

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

complex circuits are SOS (and scale better!)

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



complex circuits are SOS (and scale better!)

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

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"

$\Rightarrow$ ***but how to implement them?***

compositional inference I

cirkit

```python
from cirkit.symbolic.functional import integrate, multiply,
    conjugate

# create a deep circuit with complex parameters
c = build_symbolic_complex_circuit('quad-tree-4')

# compute the partition function of c^2
def renormalize(c):
    c1 = conjugate(c)
    c2 = multiply(c, c1)
    return integrate(c2)
```

A Quantum Information Theoretic Approach to Tractable Probabilistic Models

Pedro Zuidberg Dos Martires[1]

[1]Örebro University, Sweden

negative weights without squaring

approximate inference

e.g., via sampling

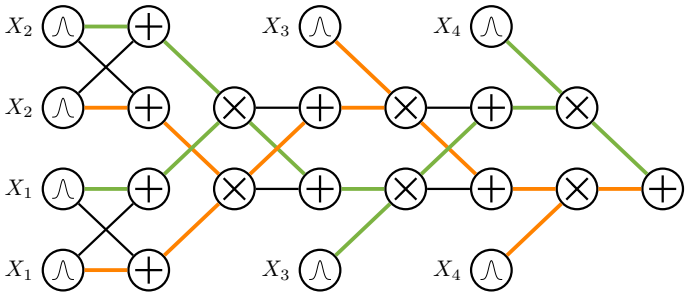
Can we use a subtractive mixture model to approximate expectations?

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

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

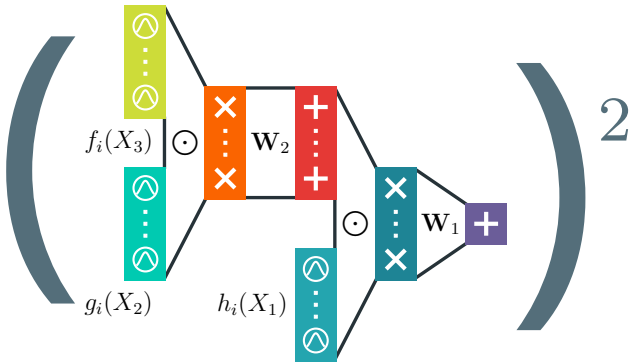
Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", TMLR, 2025

wait...!

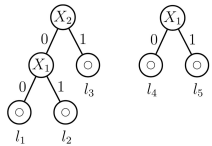


how to sample from a monotonic deep PC?

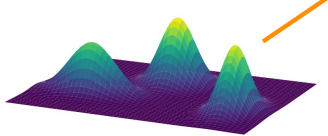
wait...!



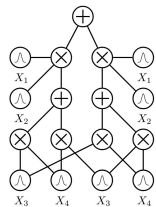
how to sample from a non-monotonic deep PC?



decision trees



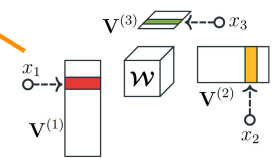
(hierarchical) mixtures



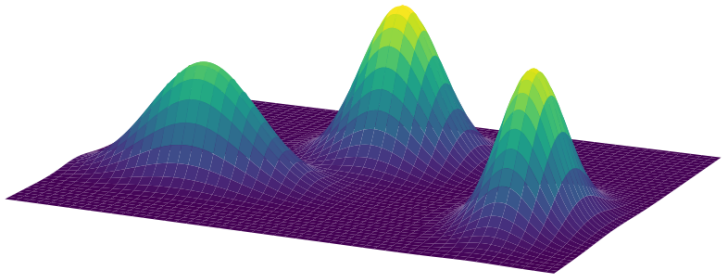
(deep) circuits

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

logical formulas & constraints

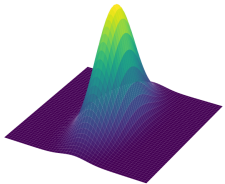


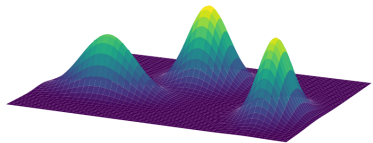
(hierarchical) tensor factorizations



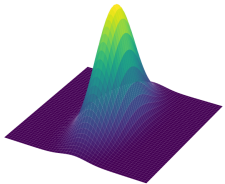
oh mixtures, you're so fine you blow my mind!

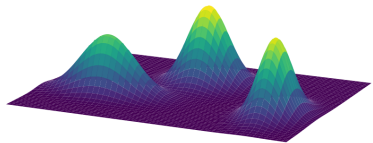
image taken from Hao Tang's course on ASR





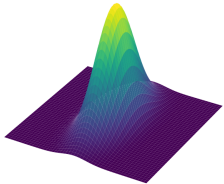
$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

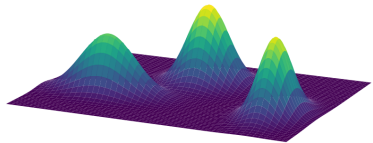


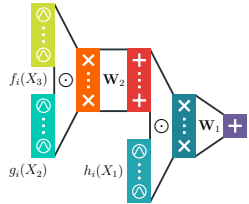


$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

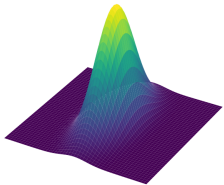
*"if someone publishes a paper on **model A**, there will be a paper about **mixtures of A** soon, with high probability"* **A. Vergari**

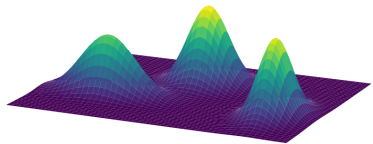


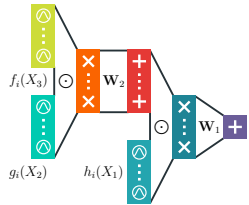




$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$



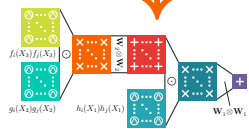




$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

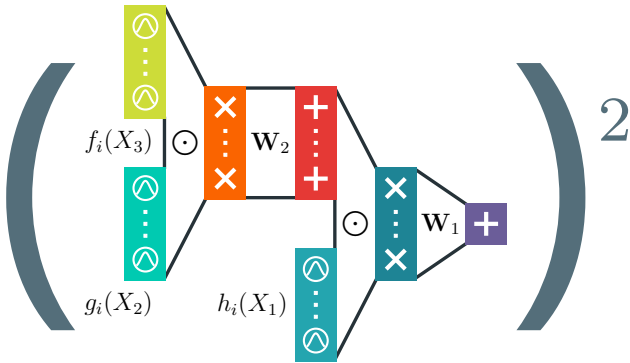
$$\downarrow \qquad\qquad\qquad\qquad\qquad\qquad \downarrow$$

$$\left(\sum_{i=1}^{K} w_i p_i(\mathbf{X})\right)^2 \quad w_i \in \mathbb{R} \rightarrow$$



learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`



questions?

structural properties

smoothness

decomposability

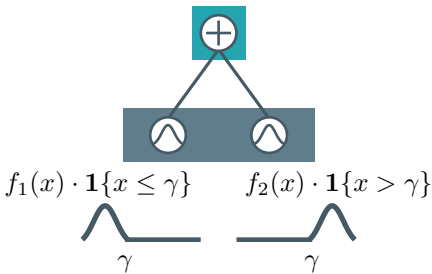
compatibility

determinism

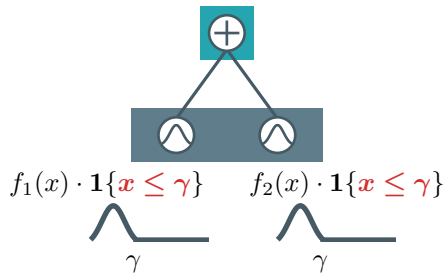
***Vergari** et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

determinism

the inputs of sum units are defined over disjoint supports



deterministic circuit



non-deterministic circuit

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