subtractive mixture models
representation, learning & inference

antonio vergari (he/him)

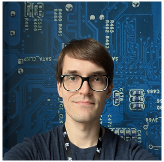
@nolovedeeplearning

15th July 2025 - **ELLIS Cambridge ML Summer School**

thanks to...

Lorenzo Loconte
U of Edinburgh

Lena Zellinger
U of Edinburgh



Aleksanteri Sladek
Aalto 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

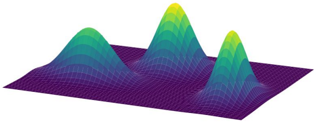
***a**bout*
***pr**obabilities*
***i**ntegrals &*
***l**ogic*

april

***a**pril is **p**robably a **r**ecursive **i**dentifier of a **l**ab*

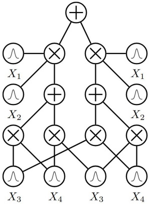
today's topic...

swiss-army knife of prob ML

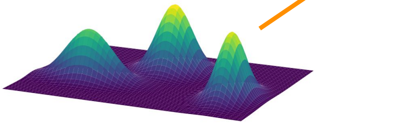


(hierarchical) mixtures

generalizing them as computational graphs

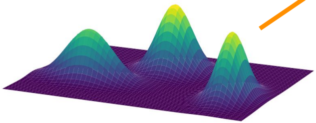


(deep) circuits

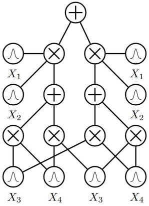


(hierarchical) mixtures

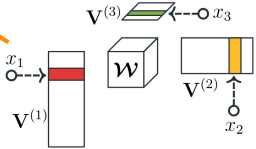
a single formalism for many models



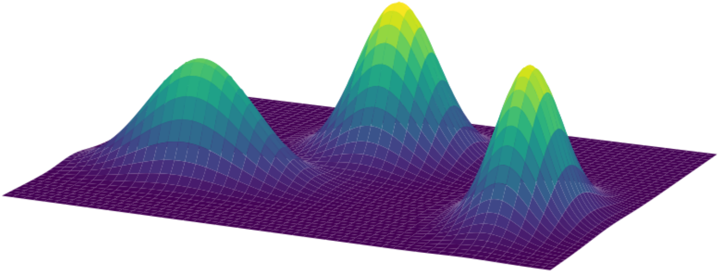
(hierarchical)
mixtures



(deep)
circuits

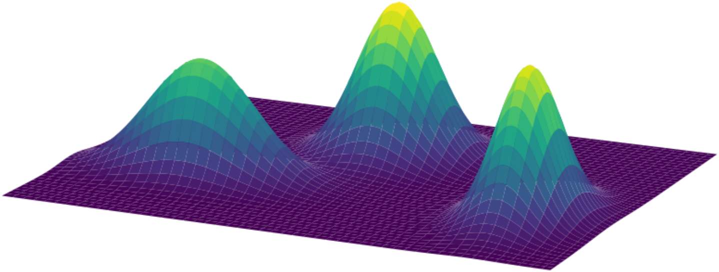


(hierarchical)
tensor factorizations



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

Hierarchical Gaussian Mixture Model Splatting for Efficient and Part Controllable 3D Generation

Qitong Yang, Mingtao Feng, Zijie Wu, Weisheng Dong, Fangfang Wu, Yaonan Wang, Ajmal Mian; Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR), 2025, pp. 11104-11114

Inversion of nitrogen and phosphorus contents in cotton leaves based on the Gaussian mixture model and differences in hyperspectral features of UAV

Lei Peng, Hui-Nan Xin, Cai-Xia Lv, Na Li, Yong-Fu Li, Qing-Long Geng, Shu-Huang Chen, Ning Lai

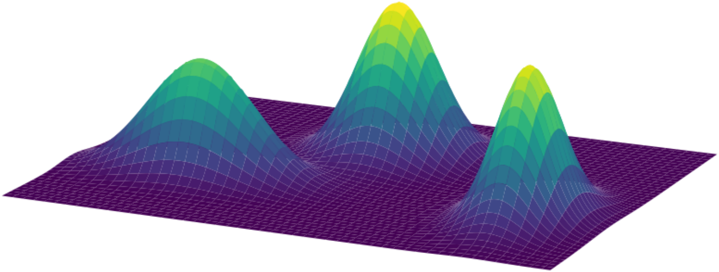
---

Gaussian Mixture Flow Matching Models

Hansheng Chen [1] Kai Zhang [2] Hao Tan [2] Zexiang Xu [3] Fujun Luan [2] Leonidas Guibas [1] Gordon Wetzstein [1] Sai Bi [2]

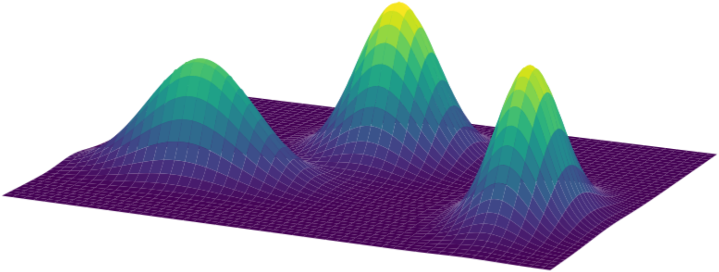
mixture models are everywhere

(still in 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



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

image taken from Hao Tang's course on ASR

$$\int \sum_i w_i p_i(\mathbf{x}) d\mathbf{x} = \sum_i w_i \int p_i(\mathbf{x}) d\mathbf{x}$$

mixture models can enable tractable inference

(if components are tractable, e.g., for marginals)

Hierarchical Decompositional Mixtures of Variational Autoencoders

Ping Liang Tan [1 2] Robert Peharz [1]

Mixtures of Laplace Approximations for Improved *Post-Hoc* Uncertainty in Deep Learning

Runa Eschenhagen[*,t] Erik Daxberger[c,m] Philipp Hennig[t,m] Agustinus Kristiadi[t]

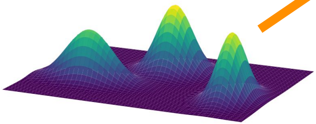
Efficient Mixture Learning in Black-Box Variational Inference

Alexandra Hotti [* 1 2 3] Oskar Kviman [* 1 2] Ricky Molén [1 2] Víctor Elvira [4] Jens Lagergren [1 2]

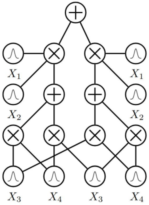
mixture models can enable tractable inference

(even in larger approximate inference pipelines)

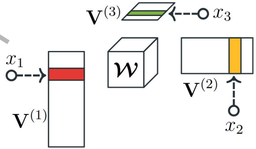
compile mixtures into circuits...



(hierarchical)
mixtures



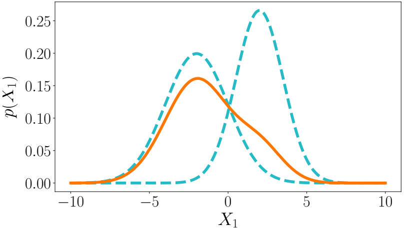
(deep)
circuits



(hierarchical)
tensor factorizations

GMMs

as computational graphs

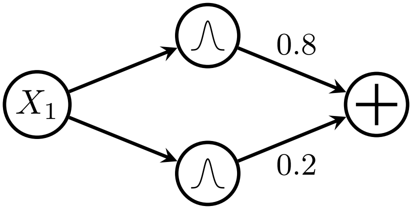


$$p(X_1) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

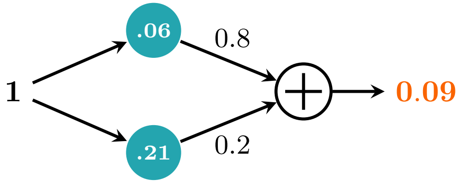


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

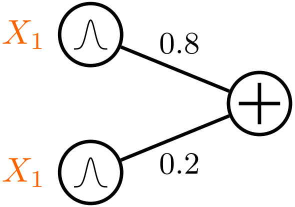


$$p(X_1 = 1) = 0.2 \cdot p_1(X_1 = 1) + 0.8 \cdot p_2(X_1 = 1)$$

$\Longrightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



A simplified notation:

⇒ ***scopes*** *attached to inputs*

⇒ *edge directions omitted*

wait...!

how do we learn them?

wait...!

how do we learn them?

$\Rightarrow$ ***by maximizing the (log-)likelihood***

which parameters?

how to reparameterize mixtures/circuits

Input distributions.

Sum unit parameters.

which parameters?

how to reparameterize mixtures/circuits

Input distributions. Each input can be a different parametric distribution

$\Longrightarrow$ *Bernoullis, Categoricals, Gaussians, **exponential families**, small NNs, ...*

Sum unit parameters.

which parameters?

how to reparameterize mixtures/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 ***negative log likelihood*** 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?

$\Rightarrow$ ***by maximizing the (log-)likelihood***

wait...!

how do we learn them?

⇒ by maximizing the (log-)likelihood

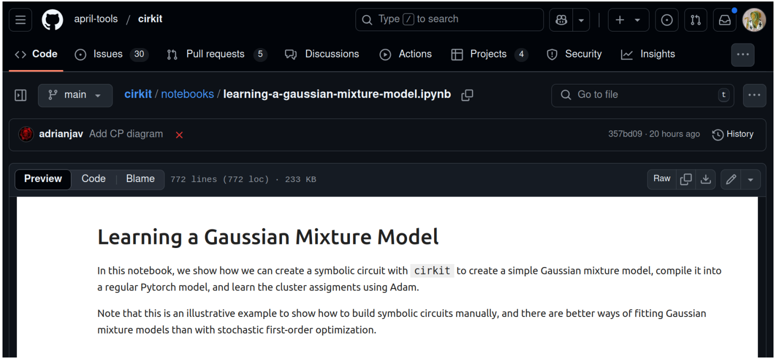
just SGD your way as usual!

⇒ or any other gradient-based optimizer

cirkit

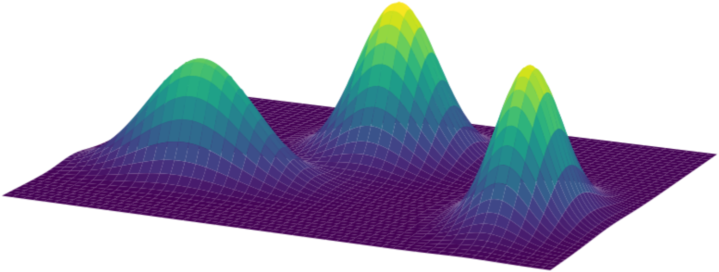
learning & reasoning with circuits in pytorch

`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

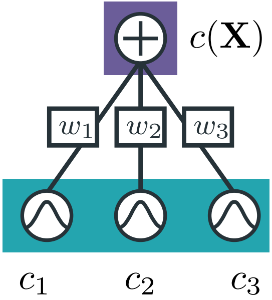


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

additive MMs

are so cool!



easily represented as shallow PCs

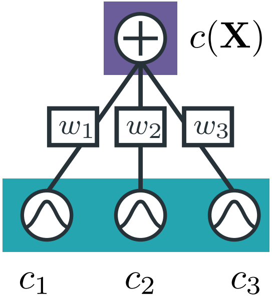
these are ***monotonic*** PCs

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

they are ***universal approximators***...

additive MMs

are so cool!



easily represented as shallow PCs

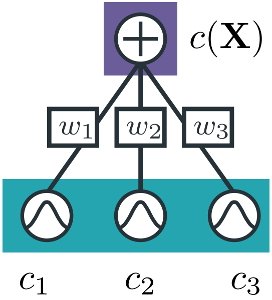
these are ***monotonic*** PCs

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

they are ***universal approximators***...

additive MMs

are so cool!



easily represented as shallow PCs

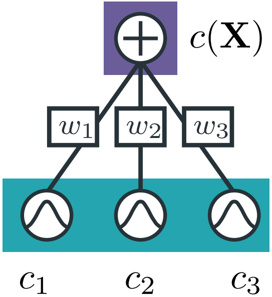
these are ***monotonic*** PCs

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

they are ***universal approximators***...

additive MMs

are so cool!



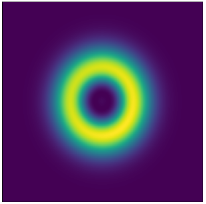
easily represented as shallow PCs

these are ***monotonic*** PCs

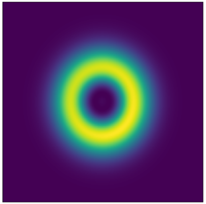
if marginals/conditionals are tractable for the components, they are tractable for the MM

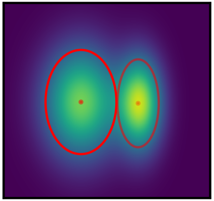
they are ***universal approximators***...

however...



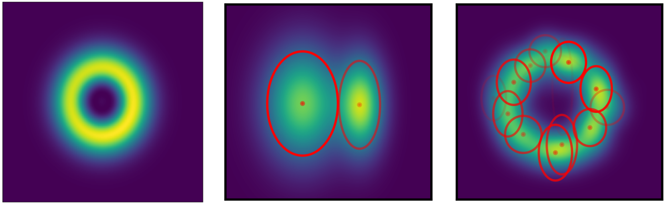
however...





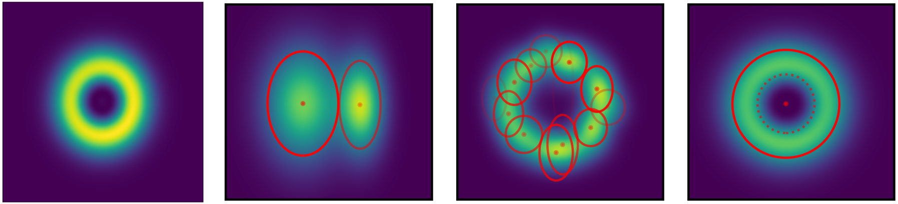
GMM ($K = 2$)

however...



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

however...



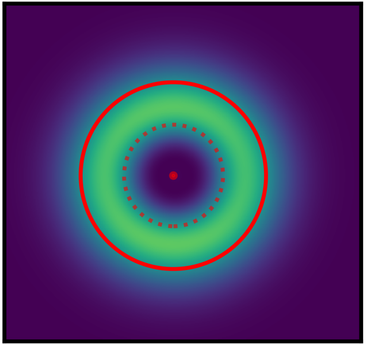
GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

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

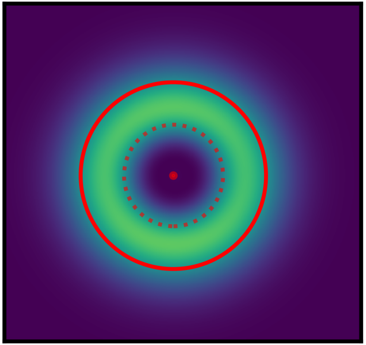
⇒ *or **non-monotonic** circuits,...*

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

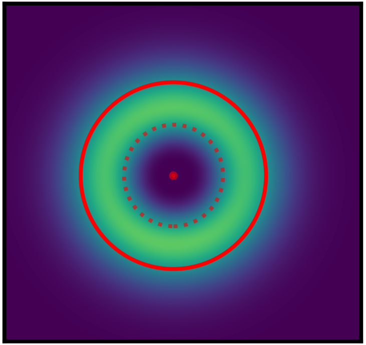
$\Rightarrow$ *or **non-monotonic** circuits,...*

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

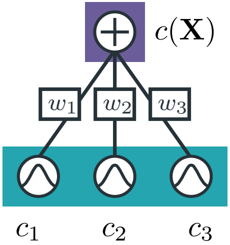
$\Rightarrow$ *or **non-monotonic** circuits,...*

issue: how to preserve non-negative outputs?

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

$\Rightarrow$ *constraints on variance, mean*

subtractive MMs as circuits

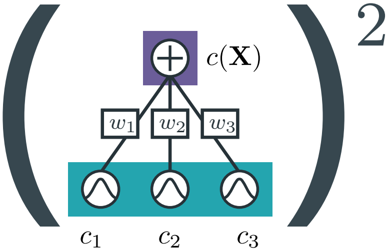


a **non-monotonic** smooth and (structured) decomposable circuit

$\Rightarrow$ *possibly with negative outputs*

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

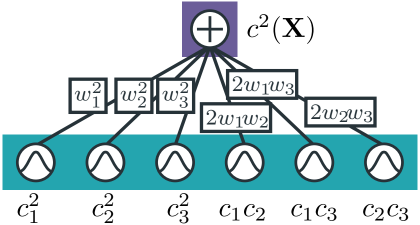
squaring shallow MMs



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

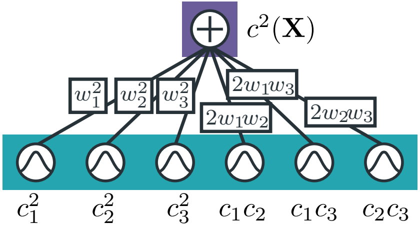
$\Longrightarrow$ *ensure non-negative output*

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

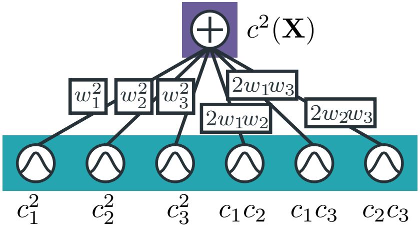


$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2$$
$$= \sum_{i=1}^{K} \sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})$$

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

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

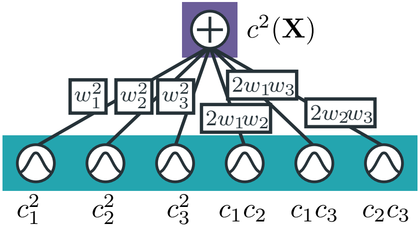
squaring shallow MMs



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

how to ***renormalize***?

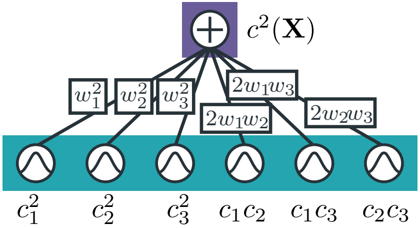
squaring shallow MMs



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

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 $\sum_i \sum_j w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x}) d\mathbf{x}$

$\Longrightarrow$ *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...!

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

wait...!

how do we learn them?

$\Rightarrow$ *by maximizing the (log-)likelihood*

which parameters?

how to reparameterize non-monotonic mixtures/circuits

Input functions.

Sum unit parameters.

which parameters?

how to reparameterize non-monotonic mixtures/circuits

Input functions. Each input can be a different parametric ***function***

$\implies$ *Bernoullis, Categoricals, Gaussians,* ***polynomials****, small NNs, ...*

Sum unit parameters.

which parameters?

how to reparameterize non-monotonic mixtures/circuits

Input functions. Each input can be a different parametric ***function***

Sum unit parameters. They can be negative, i.e., $w_i \in \mathbb{R}$ and we we need to renormalize the ***negative log likelihood*** loss after squaring

$$\min_{\theta} - \left(\sum_{i=1}^{N} 2 \log c_{\theta}(\mathbf{x}^{(i)}) - \log \int c_{\theta}^{2}(\mathbf{x}^{(i)})\, d\mathbf{X} \right)$$

wait...!

how do we learn them?

$\Rightarrow$ by maximizing the (log-)likelihood

wait...!

how do we **learn** them?

$\Rightarrow$ *by maximizing the (log-)likelihood*

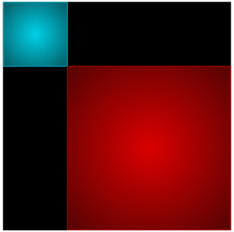
just SGD your way as usual!

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

what about deep mixtures/circuits?

GMMs

as computational graphs

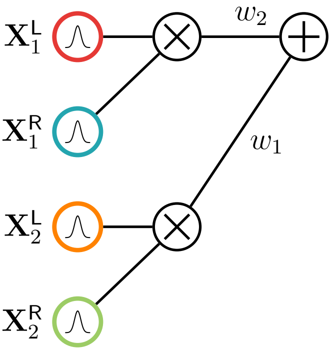


$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}') \cdot p_1(\mathbf{X}'') + w_2 \cdot p_2(\mathbf{X}''') \cdot p_2(\mathbf{X}'''')$$

$\Longrightarrow$ *local factorizations...*

GMMs

as computational graphs



$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}') \cdot p_1(\mathbf{X}'') + w_2 \cdot p_2(\mathbf{X}''') \cdot p_2(\mathbf{X}'''')$$

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

probabilistic circuits (PCs)

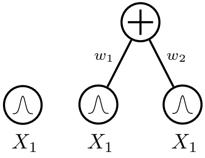
a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

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

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*

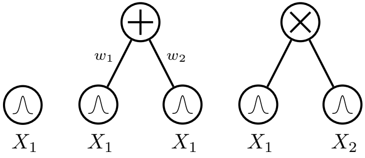
probabilistic circuits (PCs)

a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

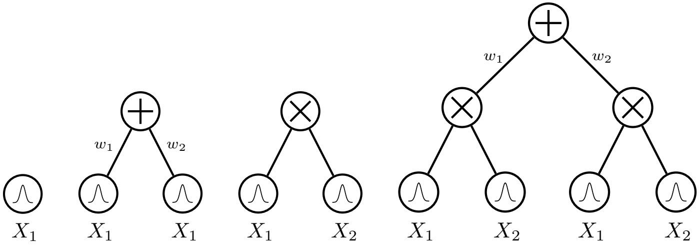
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



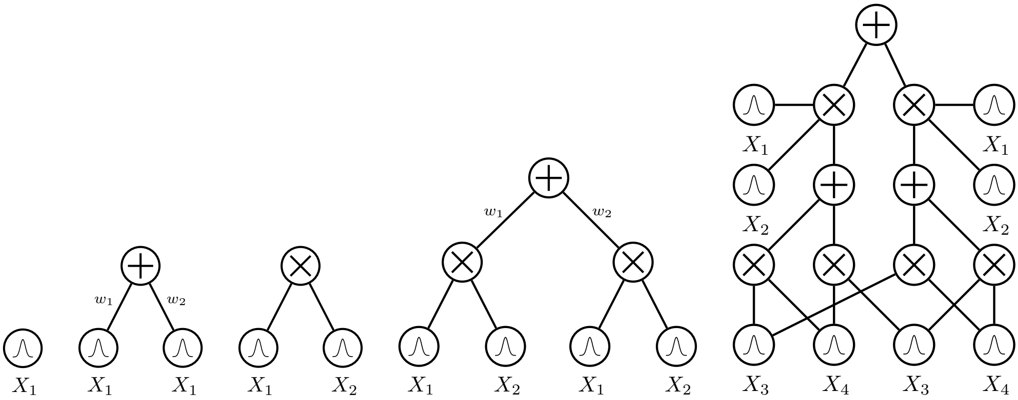
probabilistic circuits (PCs)

a grammar for tractable computational graphs



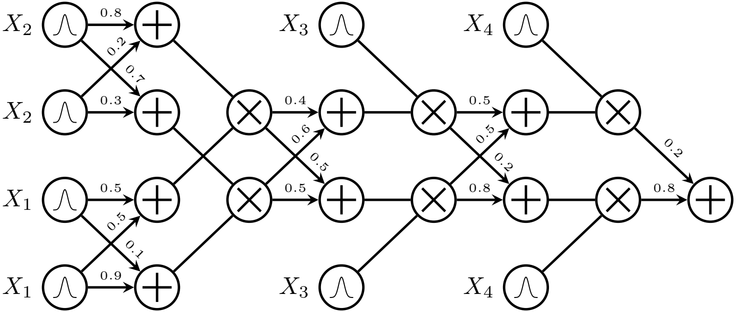
probabilistic circuits (PCs)

a grammar for tractable computational graphs



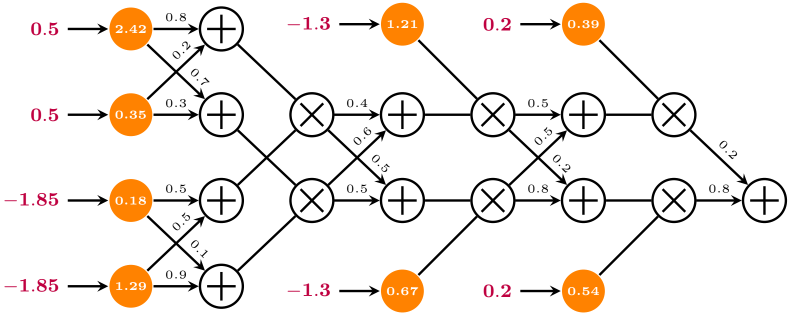
probabilistic queries = **feedforward** evaluation

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



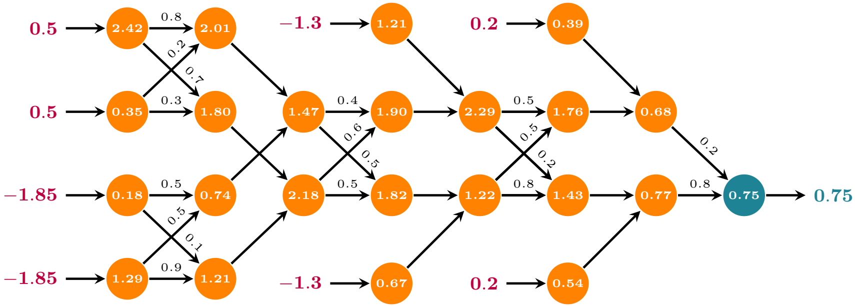
probabilistic queries = feedforward evaluation

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



probabilistic queries = feedforward evaluation

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



probabilistic circuits (PCs)

a tensorized definition

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

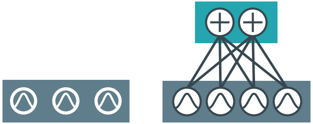
probabilistic circuits (PCs)

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)

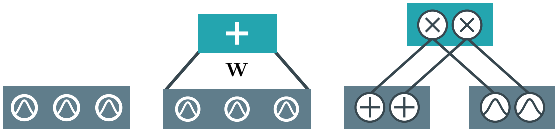
a tensorized definition

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)

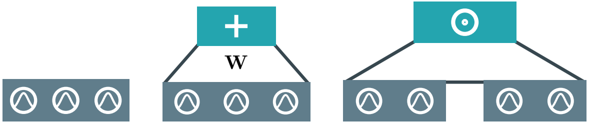
a tensorized definition

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)

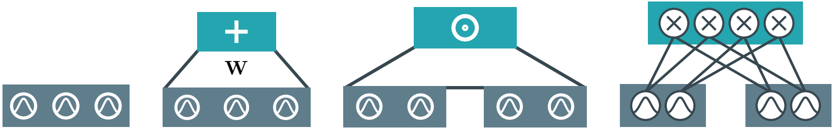
a tensorized definition

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*

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



probabilistic circuits (PCs)

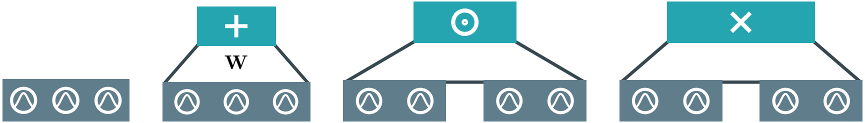
a tensorized definition

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

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

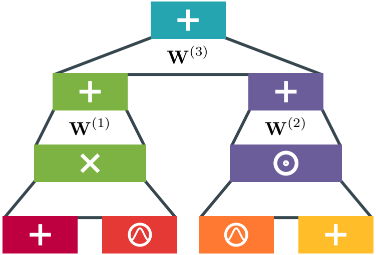
III. *The product of two layers is a circuit layer*

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



probabilistic circuits (PCs)

a tensorized definition



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

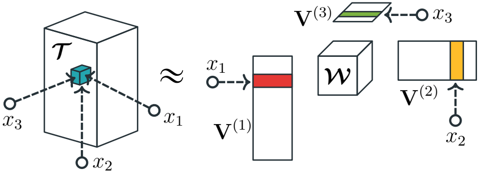
II. *A linear projection of a layer is a circuit layer*

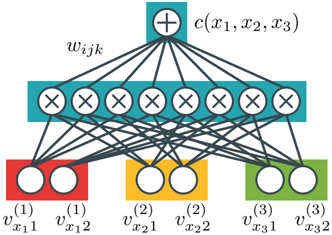
III. *The product of two layers is a circuit layer*

stack layers to build a deep circuit!

tensor factorizations

as circuits



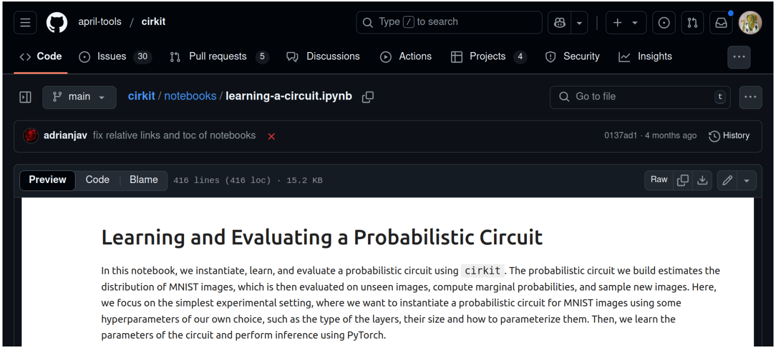


Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", TMLR, 2025

cirkit

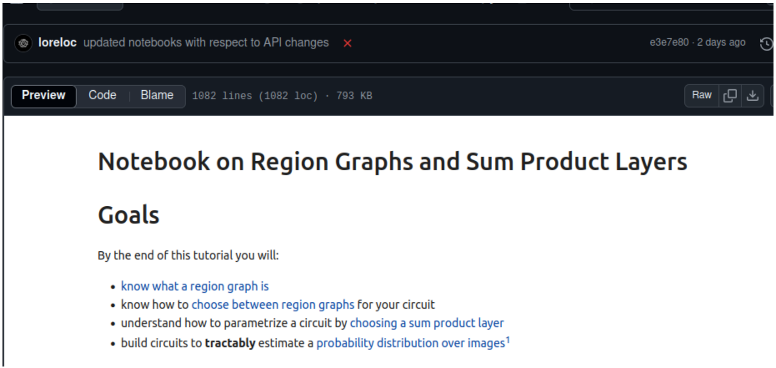
learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`



a notebook on learning a deep circuit on MNIST

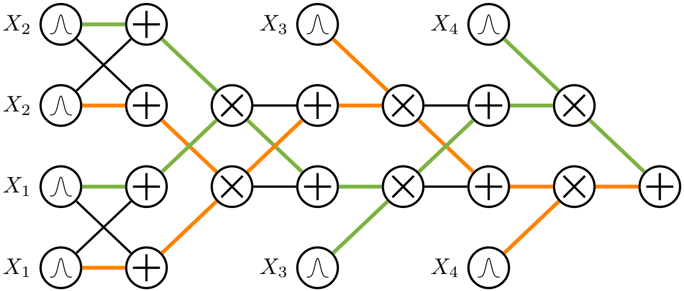
https://github.com/april-tools/cirkit/blob/main/notebooks/learning-a-circuit.ipynb



mix& match your structure and layers

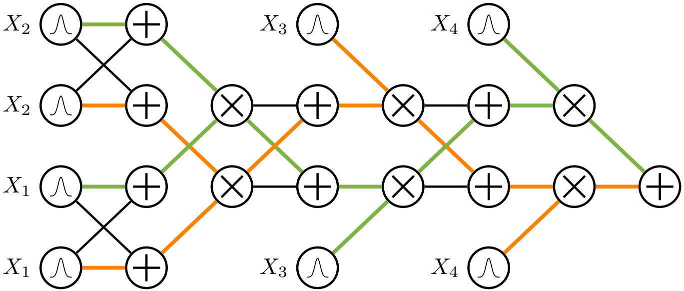
https://github.com/april-tools/cirkit/blob/main/notebooks/region-graphs-and-parametrisation.ipynb

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

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

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic models

$\Longrightarrow$ *#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***

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

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

$\Longrightarrow$ *tractable partition function*

$\Longrightarrow$ *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***

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

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

$\Longrightarrow$ *tractable partition function*

$\Longrightarrow$ *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

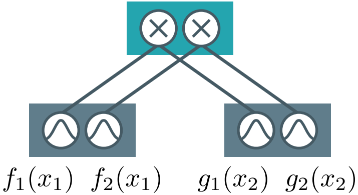
smoothness $\wedge$ decomposability $\implies$ multilinearity

---

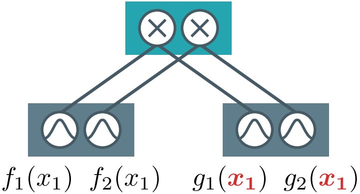
Vergari *et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", NeurIPS, 2021*

multilinearity

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



✓ **multilinear**

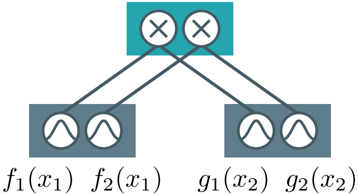


✗ **not multilinear**

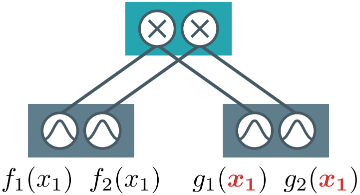
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

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



decomposable circuit

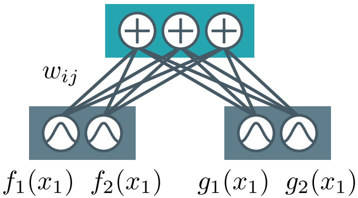


non-decomposable circuit

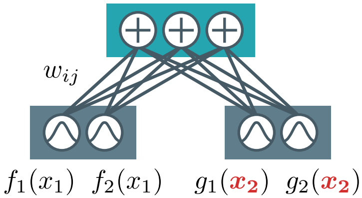
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

the inputs of sum units are defined over the same variables



✓ **multilinear**

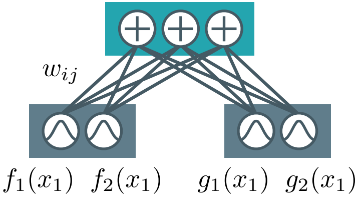


✗ **not multilinear**

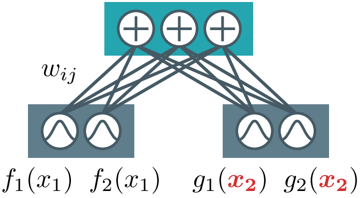
Darwiche and Marquis, "A knowledge compilation map", , 2002

multilinearity

the inputs of sum units are defined over the same variables



smooth circuit

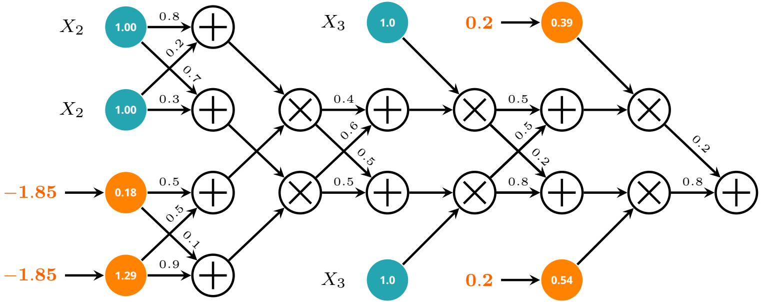


non-smooth circuit

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

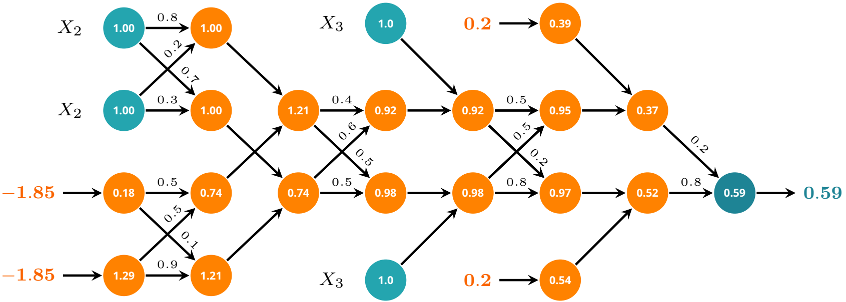
marginal queries = feedforward evaluation

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



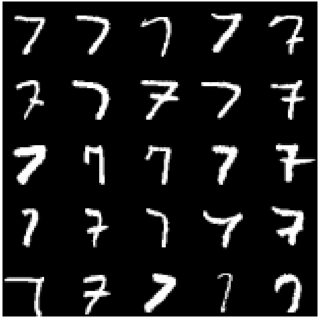
marginal queries = feedforward evaluation

$$p(X_1 = -\mathbf{1.85}, X_4 = \mathbf{0.2})$$

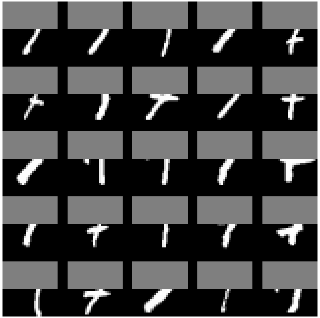


tractable marginals on PCs

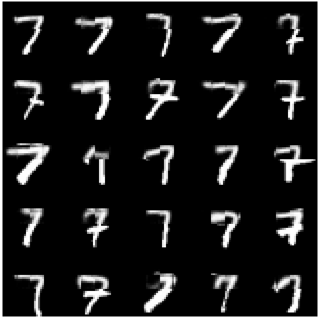
Original



Missing



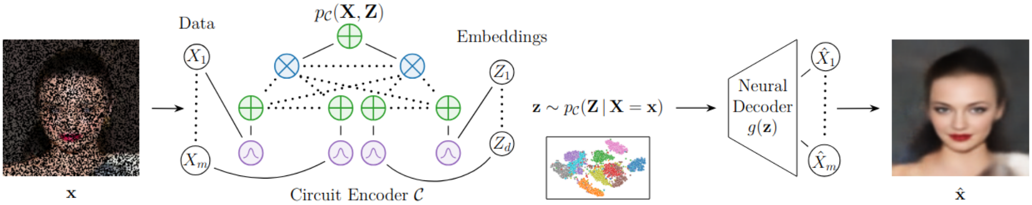
Conditional sample



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

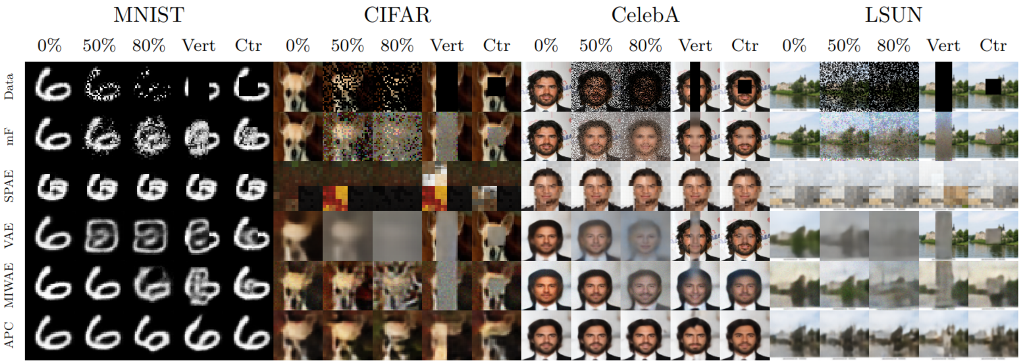
use tractable models
inside intractable pipelines
where it matters!

tractable + intractable



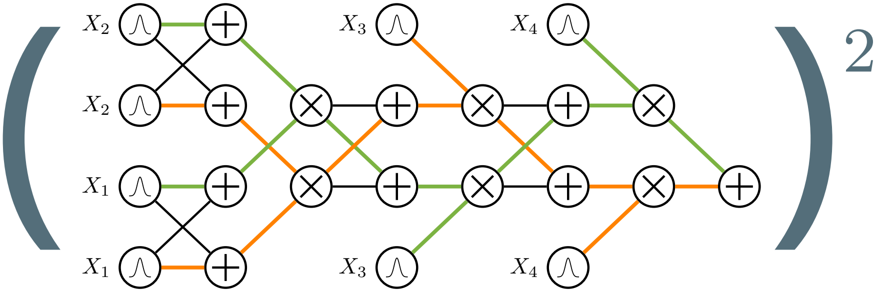
tractable conditioning over every missing mask

(under submission)



better than (V)AEs for missing values

(under submission)



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

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

compositional inference

```
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)
```

structural properties

smoothness

decomposability

property C

property D

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

structural properties

smoothness

decomposability

compatibility

property D

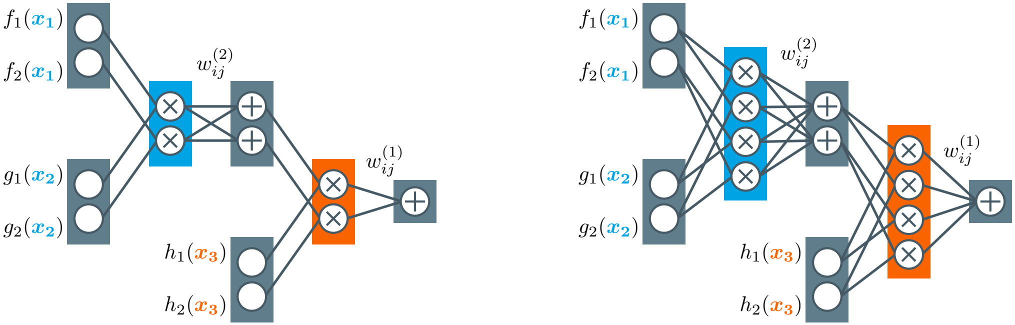
Integrals involving two or more functions:
e.g., expectations

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

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

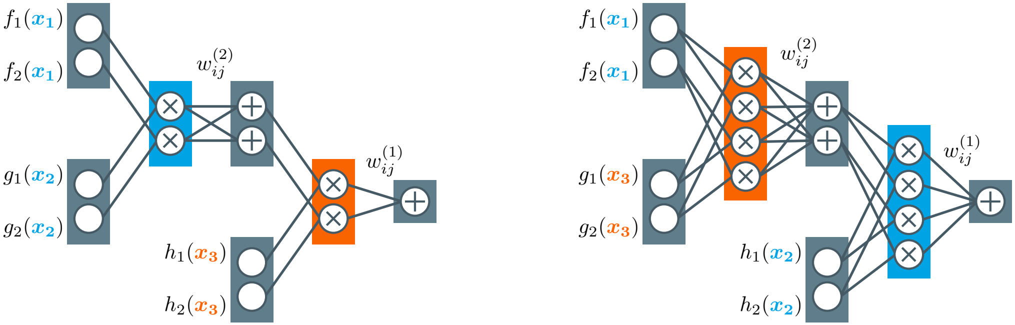
compatibility



compatibile circuits

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

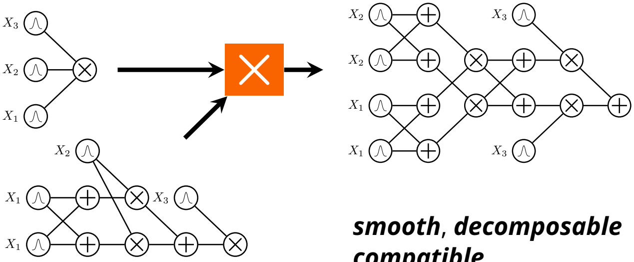
compatibility



non-compatibile circuits

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

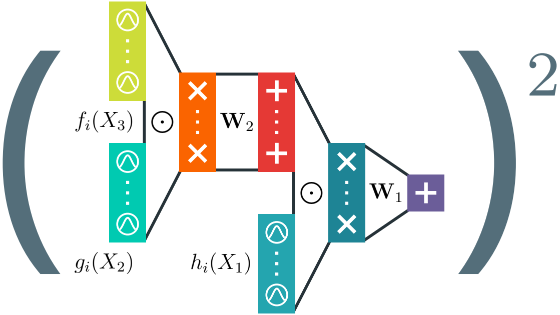
tractable products



smooth*, *decomposable
compatible

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

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

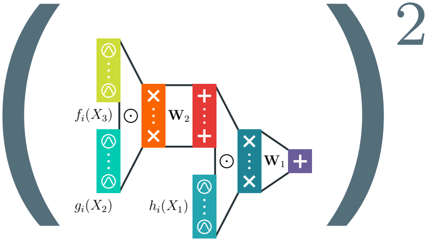


how to efficiently square (and *renormalize*) a deep PC?

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

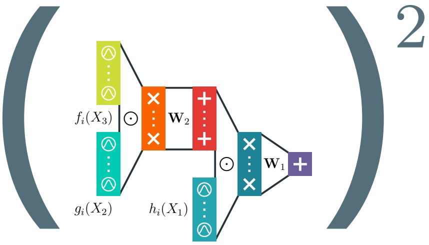
squaring deep PCs

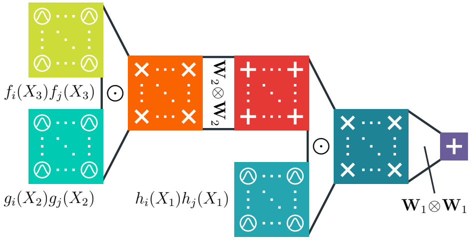
the tensorized way



squaring deep PCs

the tensorized way

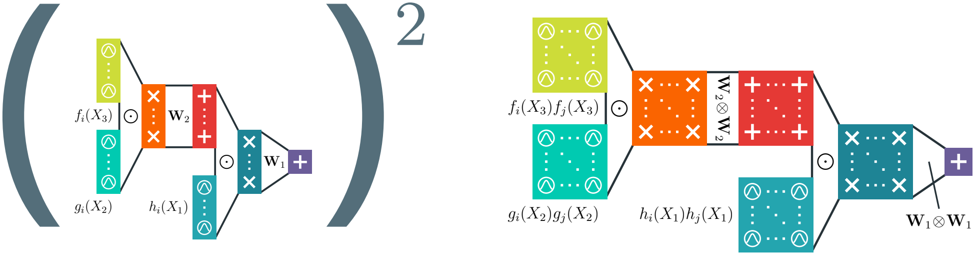




squaring a circuit = squaring layers

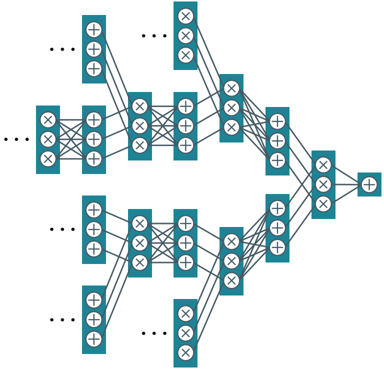
squaring deep PCs

the tensorized way



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

theorem I

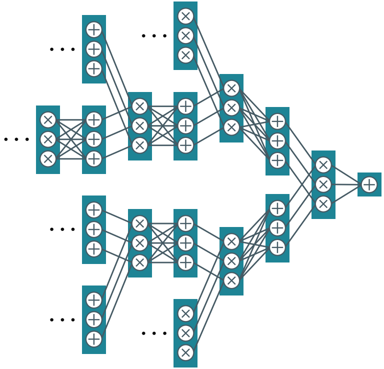


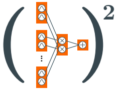
$\exists\, p'$ **requiring exponentially large**

monotonic circuits...

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

theorem I



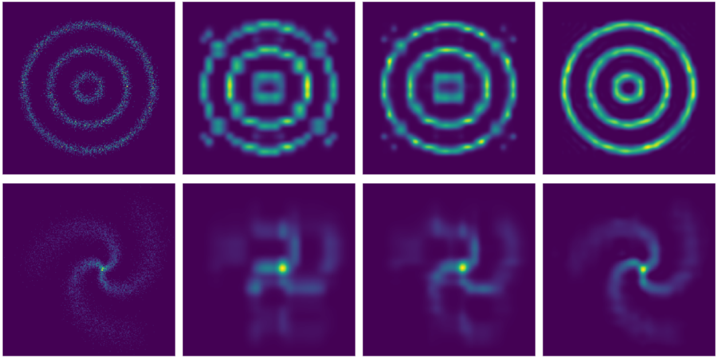


...but compact

squared non-monotonic circuits

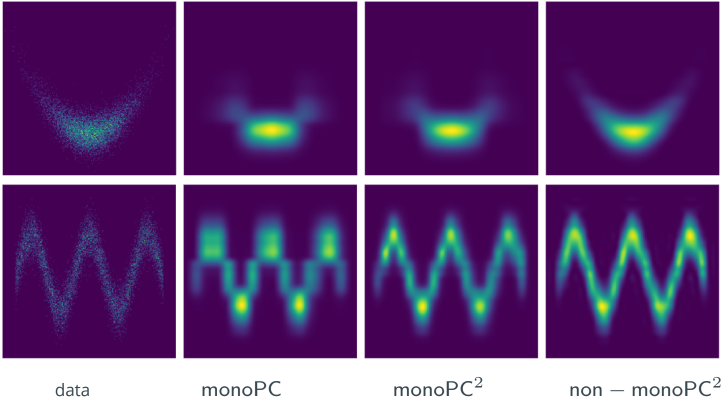
Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", ICLR, 2024

more expressive?



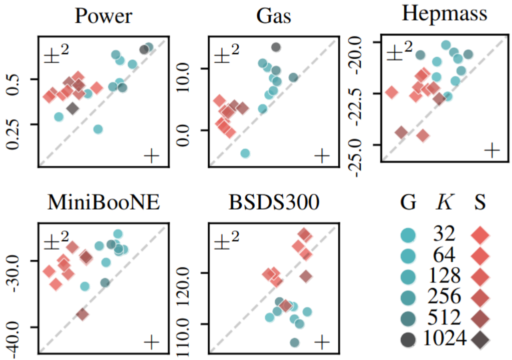
data | monoPC | monoPC^2 | $\text{non}-\text{monoPC}^2$

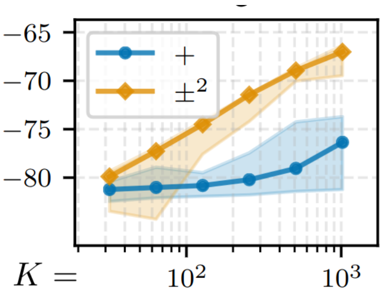
more expressive?



how more expressive?

real-world data

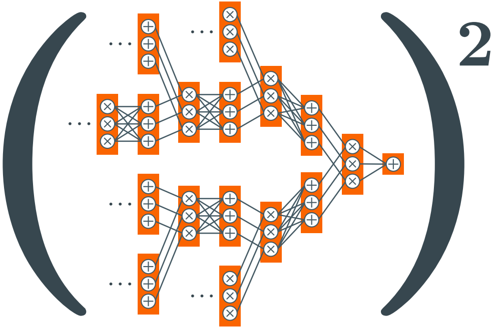




theorem II

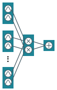
$\exists\ p''$ **requiring exponentially large**

squared non-mono circuits...



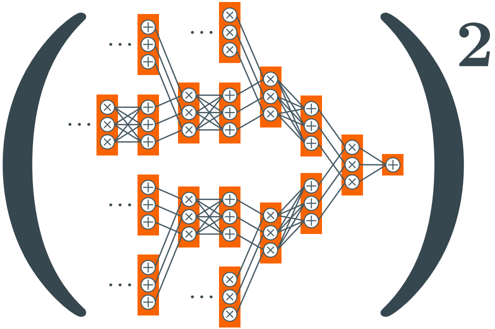
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem II

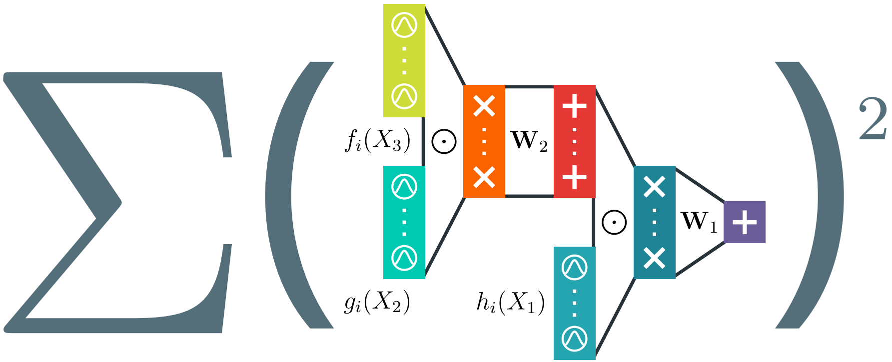


...but compact

monotonic circuits...!

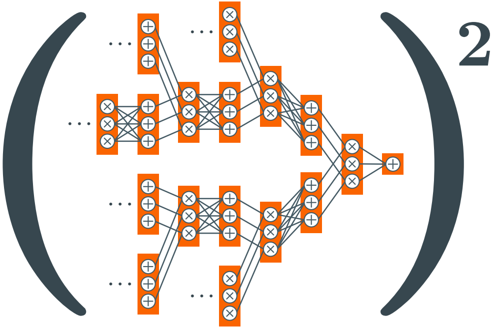


Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



what if we use more that one square?

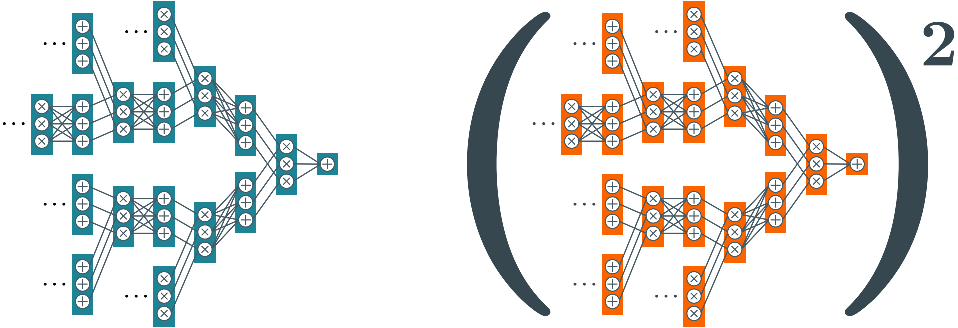
theorem III



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

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

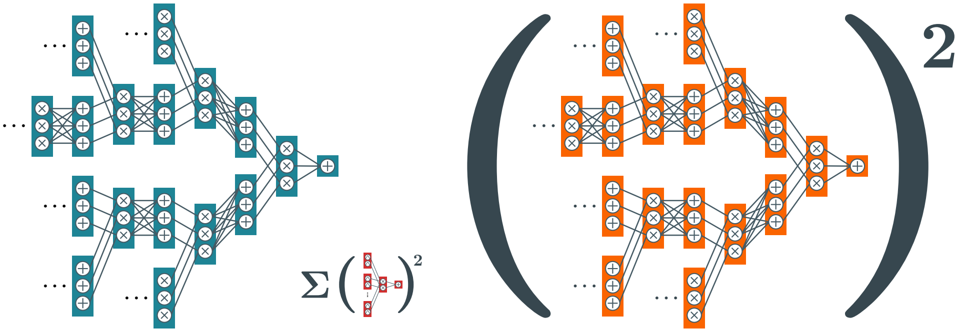
theorem III



...exponentially large monotonic circuits...

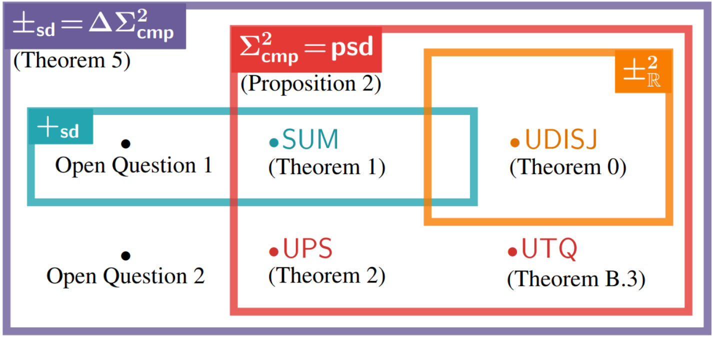
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem III



...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:

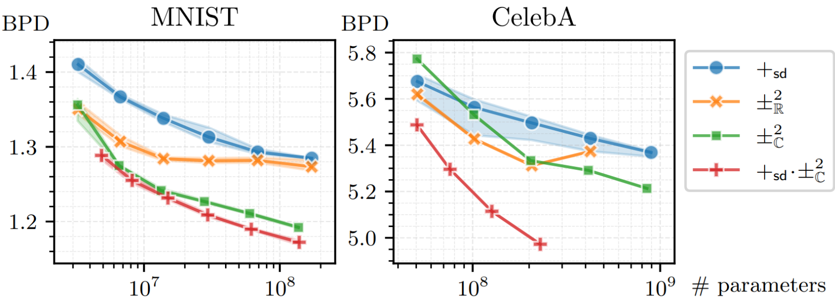
$$c^2(\mathbf{x}) = c(\mathbf{x})^\dagger c(\mathbf{x}), \qquad c(\mathbf{x}) \in \mathbb{C}$$

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?***

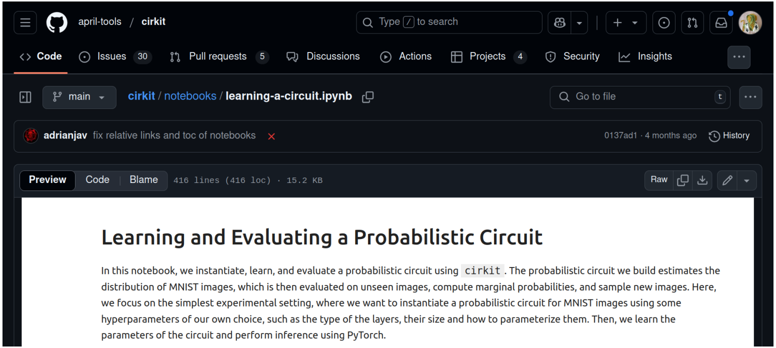
compositional inference I

cirkit

```
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 notebook on learning SOS subtractive mixtures

`https://github.com/april-tools/cirkit/blob/main/notebooks/sum-of-squares-circuits.ipynb`

approximate inference

e.g., via sampling

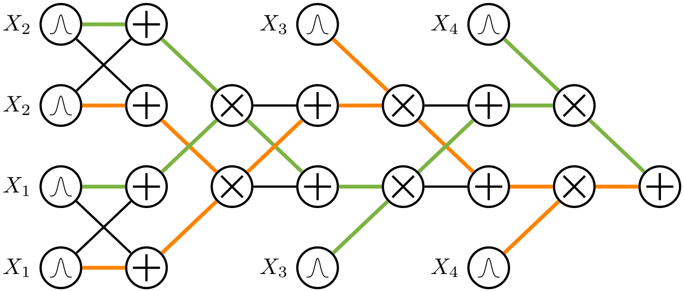
Can we use a subtractive mixture model to approximate expectations?

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

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

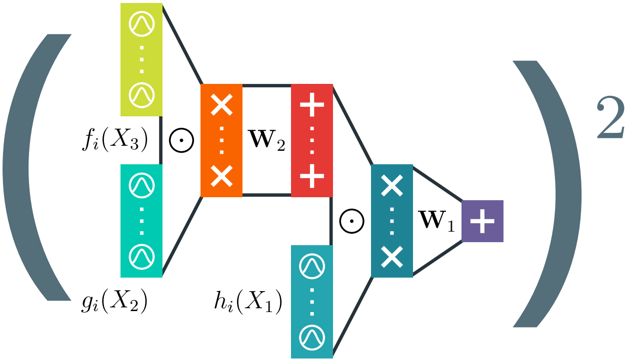
Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", TMLR, 2025

wait...!



how to sample from a monotonic deep PC?

wait...!



how to sample from a non-monotonic deep PC?

approximate inference

e.g., via sampling

Can we use a subtractive mixture model to approximate expectations?

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

$\Longrightarrow$ *but how to sample from a* ***non-monotonic*** *q?*

use ***autoregressive inverse transform sampling***:

$$x_1 \sim q(x_1), \qquad x_i \sim q(x_i | \mathbf{x}_{<i}) \qquad \text{for} \quad i \in \{2, \ldots, d\}$$

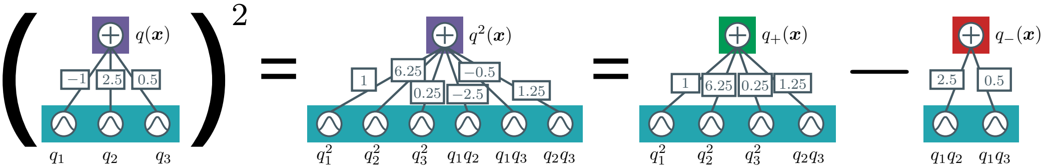
$\Longrightarrow$ *can be slow for large dimensions, requires* ***inverting the CDF***

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

approximate inference

difference of expectation estimator

Idea: represent q as a difference of two additive mixtures



Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025

approximate inference

difference of expectation estimator

Idea: represent q as a difference of two additive mixtures

$$q(\mathbf{x}) = Z_+ \cdot q_+(\mathbf{x}) - Z_- \cdot q_-(\mathbf{x})$$

$\Longrightarrow$ *expectations will break down in two "parts"*

Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025

approximate inference

difference of expectation estimator

Idea: represent q as a difference of two additive mixtures

$$q(\mathbf{x}) = Z_+ \cdot q_+(\mathbf{x}) - Z_- \cdot q_-(\mathbf{x})$$

$\Longrightarrow$ *expectations will break down in two "parts"*

$$\frac{Z_+}{S_+} \sum_{s=1}^{S_+} f(\mathbf{x}_+^{(s)}) - \frac{Z_-}{S_-} \sum_{s=1}^{S_-} f(\mathbf{x}_-^{(s)}), \text{ where } \begin{array}{l} \mathbf{x}_+^{(s)} \sim q_+(\mathbf{x}_+) \\ \mathbf{x}_-^{(s)} \sim q_-(\mathbf{x}_-) \end{array}, \quad (1)$$

Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025

approximate inference

difference of expectation estimator

Method	d	$K=2$: $\log(\lvert\widehat{I}-I\rvert)$	$K=2$: Time (s)	$K=4$: $\log(\lvert\widehat{I}-I\rvert)$	$K=4$: Time (s)	$K=6$: $\log(\lvert\widehat{I}-I\rvert)$	$K=6$: Time (s)
		Number of components (K)					
		2		**4**		**6**	
ΔExS	16	-19.507 $\pm$ 1.025	0.293 $\pm$ 0.004	-19.062 $\pm$ 0.823	1.049 $\pm$ 0.077	-19.497 $\pm$ 1.974	2.302 $\pm$ 0.159
ARITS	16	-19.111 $\pm$ 1.103	7.525 $\pm$ 0.038	-19.299 $\pm$ 1.611	7.52 $\pm$ 0.023	-18.739 $\pm$ 1.024	7.746 $\pm$ 0.032
ΔExS	32	-48.411 $\pm$ 1.265	0.325 $\pm$ 0.012	-48.046 $\pm$ 0.972	1.027 $\pm$ 0.107	-48.34 $\pm$ 0.814	2.213 $\pm$ 0.177
ARITS	32	-47.897 $\pm$ 1.165	15.196 $\pm$ 0.059	-47.349 $\pm$ 0.839	15.535 $\pm$ 0.059	-47.3 $\pm$ 0.978	17.371 $\pm$ 0.06
ΔExS	64	-108.095 $\pm$ 1.094	0.38 $\pm$ 0.034	-107.56 $\pm$ 0.616	0.9 $\pm$ 0.14	-107.653 $\pm$ 0.945	1.512 $\pm$ 0.383
ARITS	64	-107.898 $\pm$ 1.129	30.459 $\pm$ 0.098	-107.33 $\pm$ 0.929	33.892 $\pm$ 0.119	-107.374 $\pm$ 1.138	52.02 $\pm$ 0.127

faster than autoregressive sampling

Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025

approximate inference

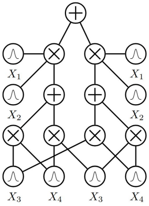
difference of expectation estimator

		Number of components (K)					
		2		**4**		**6**	
Method	d	$\log(\lvert\widehat{I} - I\rvert)$	Time (s)	$\log(\lvert\widehat{I} - I\rvert)$	Time (s)	$\log(\lvert\widehat{I} - I\rvert)$	Time (s)
ΔExS	16	-19.507 ± 1.025	0.293 ± 0.004	-19.062 ± 0.823	1.049 ± 0.077	-19.497 ± 1.974	2.302 ± 0.159
ARITS	16	-19.111 ± 1.103	7.525 ± 0.038	-19.299 ± 1.611	7.52 ± 0.023	-18.739 ± 1.024	7.746 ± 0.032
ΔExS	32	-48.411 ± 1.265	0.325 ± 0.012	-48.046 ± 0.972	1.027 ± 0.107	-48.34 ± 0.814	2.213 ± 0.177
ARITS	32	-47.897 ± 1.165	15.196 ± 0.059	-47.349 ± 0.839	15.535 ± 0.059	-47.3 ± 0.978	17.371 ± 0.06
ΔExS	64	-108.095 ± 1.094	0.38 ± 0.034	-107.56 ± 0.616	0.9 ± 0.14	-107.653 ± 0.945	1.512 ± 0.383
ARITS	64	-107.898 ± 1.129	30.459 ± 0.098	-107.33 ± 0.929	33.892 ± 0.119	-107.374 ± 1.138	52.02 ± 0.127

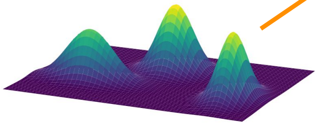
how to learn SMMs via VI...?

Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025

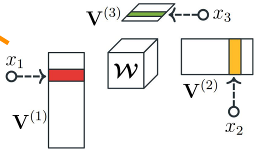
towards conclusions...



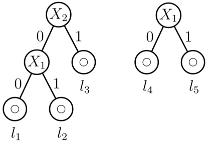
(deep)
circuits



(hierarchical)
mixtures



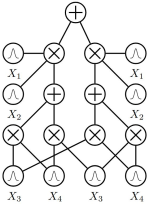
(hierarchical)
tensor factorizations



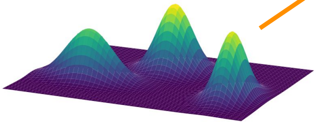
decision trees

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

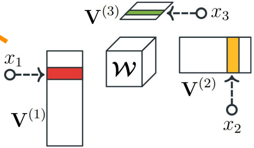
logical formulas & constraints



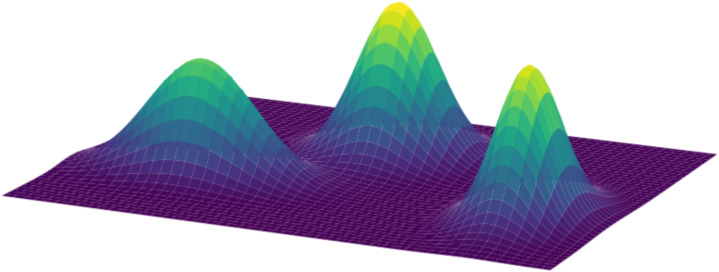
(deep) circuits



(hierarchical) mixtures

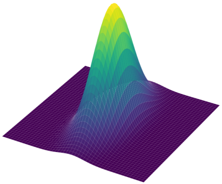


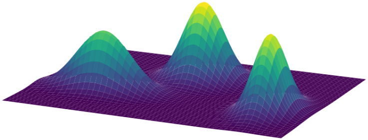
(hierarchical) tensor factorizations



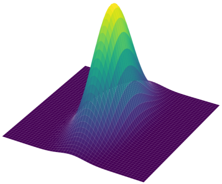
oh mixtures, you're so fine you blow my mind!

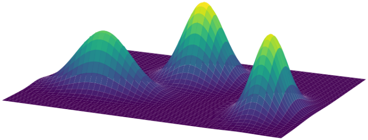
image taken from Hao Tang's course on ASR





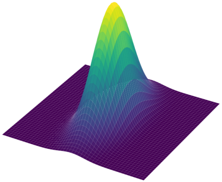
$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

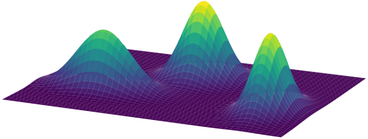


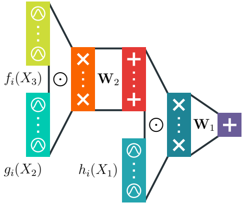


$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

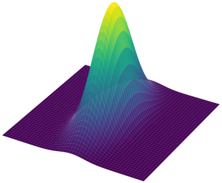
*“if someone publishes a paper on **model A**, there will be a paper about **mixtures of A** soon, with high probability”* **A. Vergari**

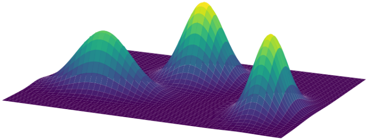


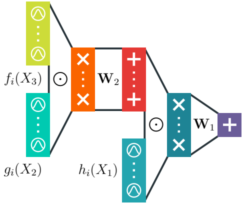




$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

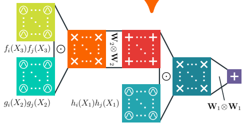






$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

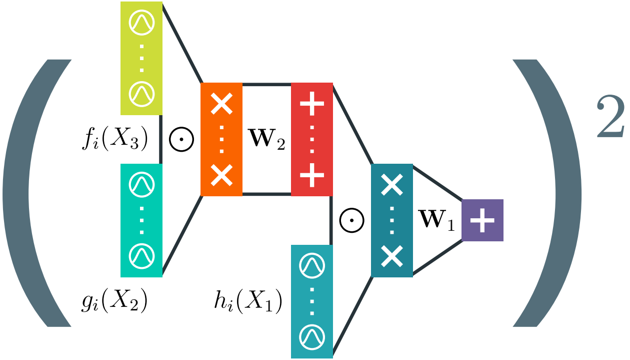
$$\sum_{j} \left(\sum_{i=1}^{K} w_{ij} p_{ij}(\mathbf{X}) \right)^2 \quad w_{ij} \in \mathbb{R}$$



cirkit

learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`



questions?

structural properties

smoothness

decomposability

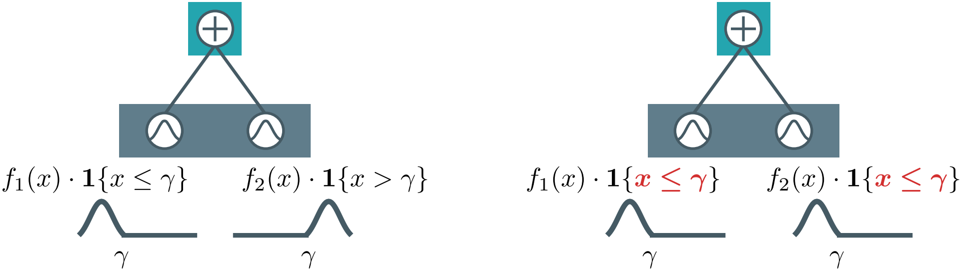
compatibility

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