subtractive mixture models
representation, learning & inference

antonio vergari (he/him)

@tetraduzione

18th Feb 2025 - **Foundations of AI Seminar** *Warwick*

april

april-tools.github.io

april

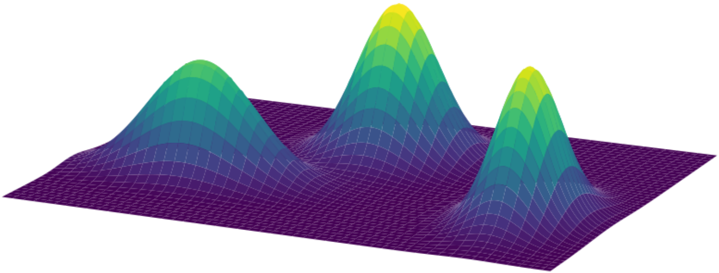
***a**bout*
***pr**obabilities*
***i**ntegrals &*
***l**ogic*

april

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

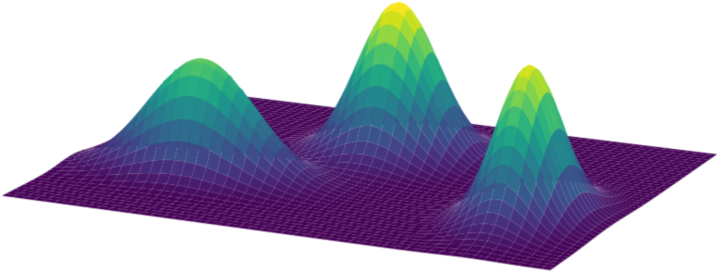
april

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



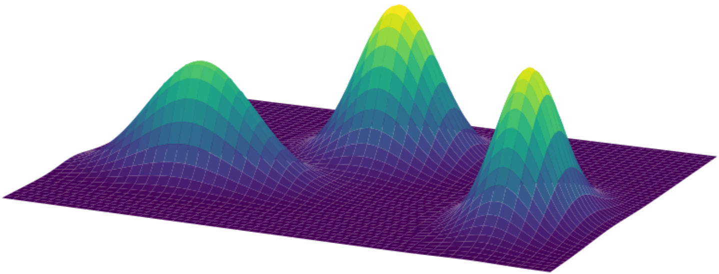
who knows mixture models?

image taken from Hao Tang’s course on ASR



who loves mixture models?

image taken from Hao Tang's course on ASR

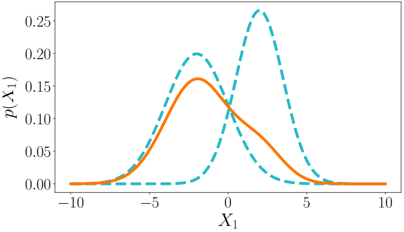


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

GMMs

as computational graphs

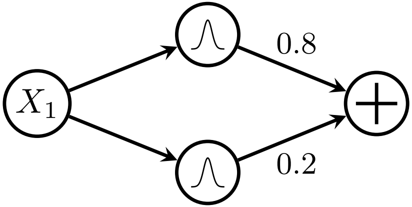


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

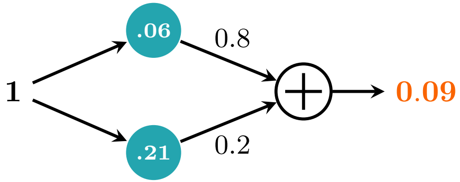


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

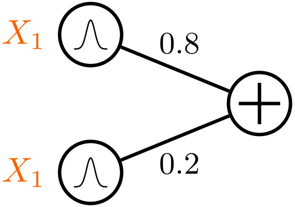


$$p(X = 1) = 0.2 \cdot p_1(X_1 = 1) + 0.8 \cdot p_2(X_1 = 1)$$

$\Longrightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



A simplified notation:

⇒ ***scopes*** *attached to inputs*

⇒ *edge directions omitted*

GMMs

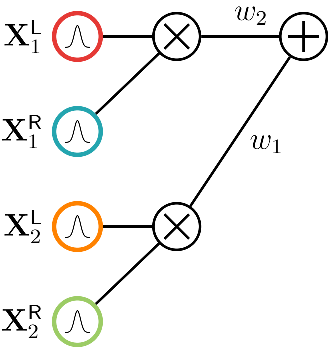
as computational graphs

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

$\Longrightarrow$ *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

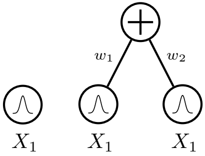
I. *A simple tractable function is a circuit*

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*



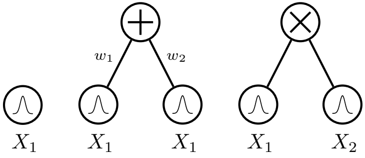
probabilistic circuits (PCs)

a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

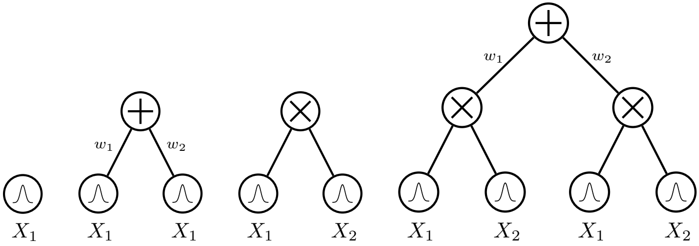
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



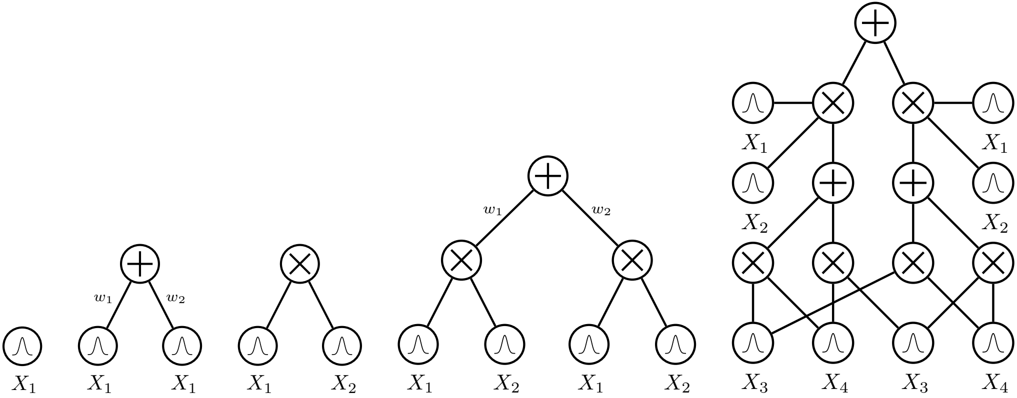
probabilistic circuits (PCs)

a grammar for tractable computational graphs

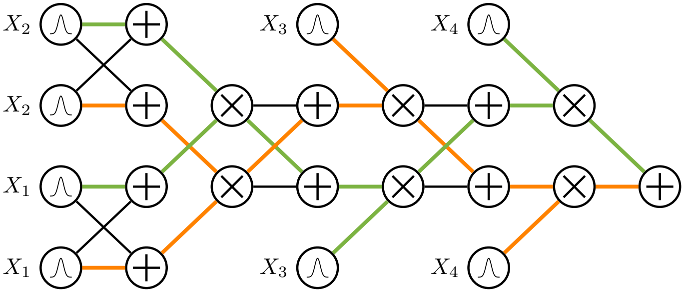


probabilistic circuits (PCs)

a grammar for tractable computational graphs

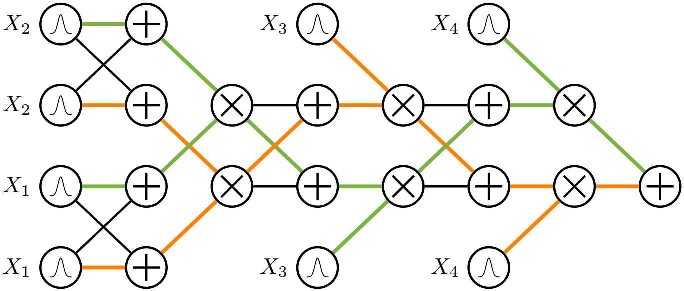


deep mixtures



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

deep mixtures



an exponential number of mixture components!

circuits

(and variants)

everywhere

Semantic Probabilistic Layers for Neuro-Symbolic Learning

Kareem Ahmed
CS Department
UCLA
ahmedk@cs.ucla.edu

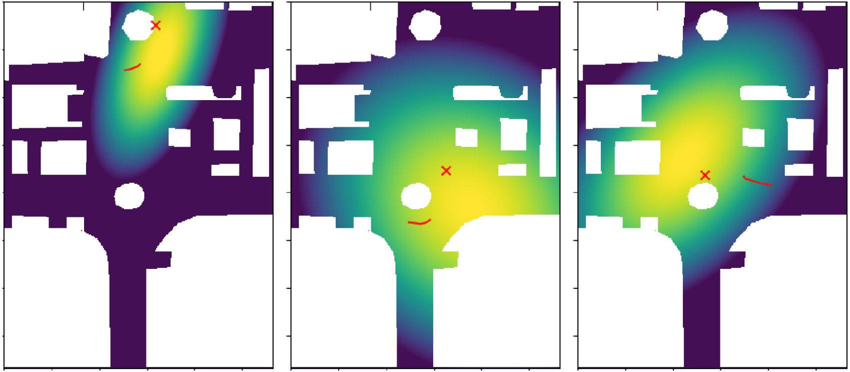
Stefano Teso
CIMeC and DISI
University of Trento
stefano.teso@unitn.it

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

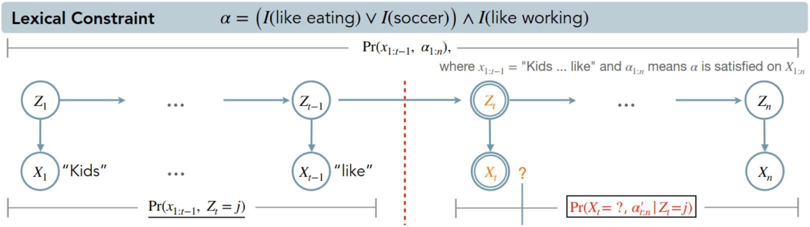
***enforce constraints in neural networks* at NeurIPS 2022**



extending it to SMT constraints

Tractable Control for Autoregressive Language Generation

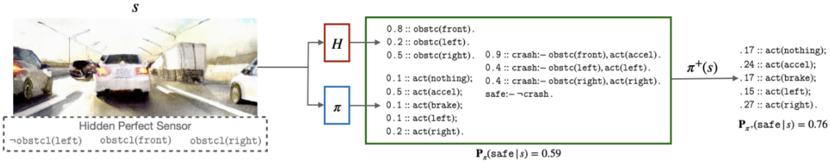
Honghua Zhang [*1] Meihua Dang [*1] Nanyun Peng [1] Guy Van den Broeck [1]



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

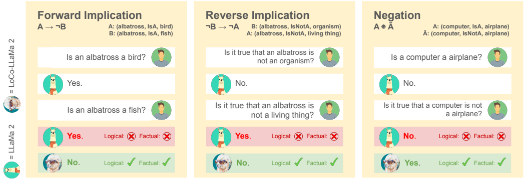
Nicola Di Mauro
University of Bari, Italy
nicola.dimauro@uniba.it

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

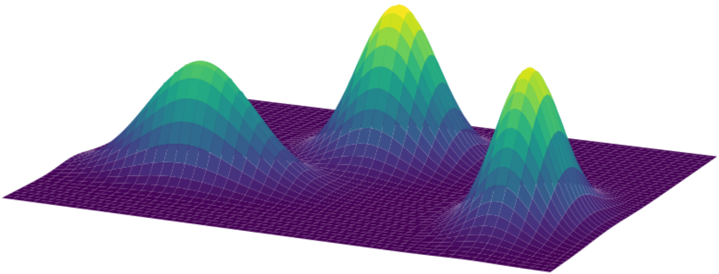


improving logical (self-)consistency in LLMs at ICLR 2025

cirkit

learning & reasoning with circuits in pytorch

github.com/april-tools/cirkit

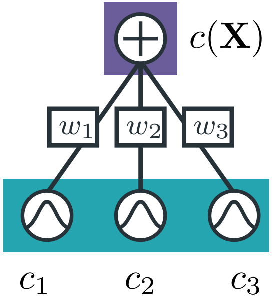


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

additive MMs

are so cool!



easily represented as shallow PCs

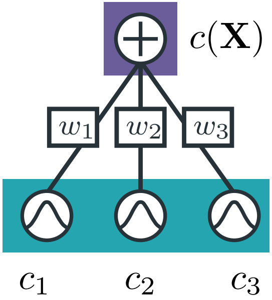
these are **monotonic** PCs

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

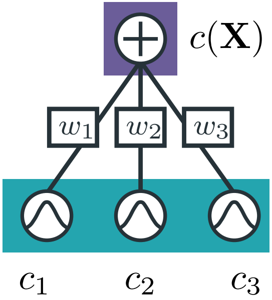
these are **monotonic** PCs

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

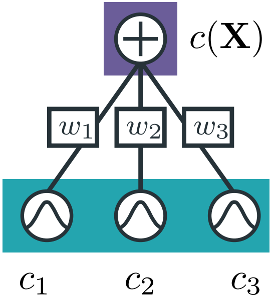
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



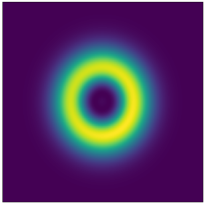
easily represented as shallow PCs

these are **monotonic** PCs

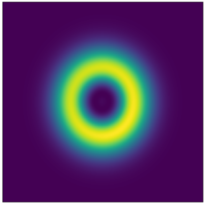
if marginals/conditionals are tractable for the components, they are tractable for the MM

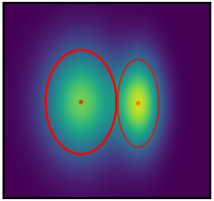
universal approximators...

however...



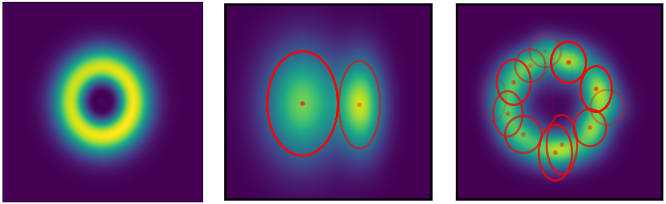
however...





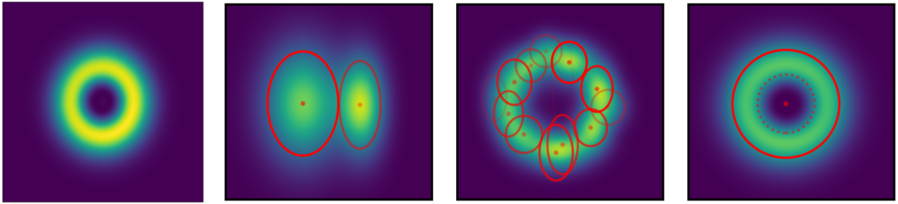
GMM ($K = 2$)

however...



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

however...



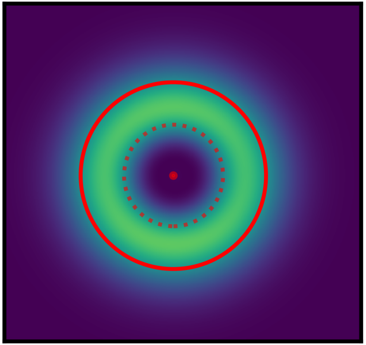
GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

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

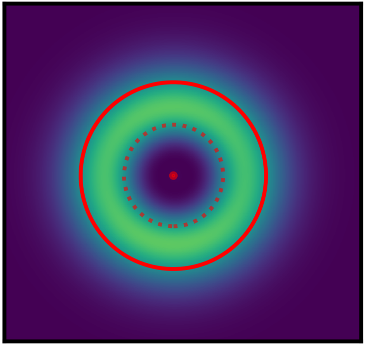
⇒ *or **non-monotonic** circuits,...*

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

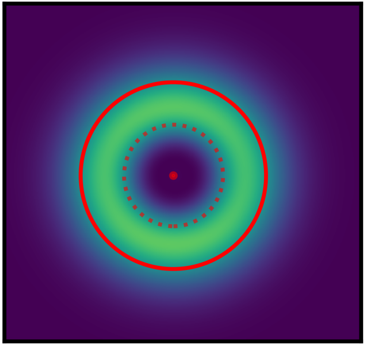
$\Rightarrow$ *or **non-monotonic** circuits,...*

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*

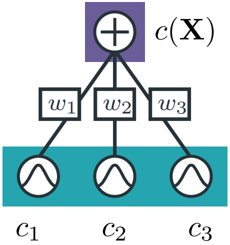
tl;dr

"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 squared circuits"

subtractive MMs as circuits

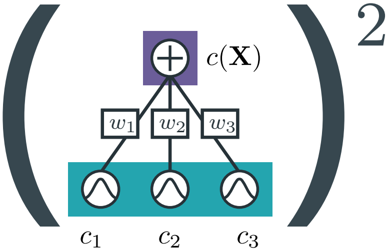


a **non-monotonic** smooth and (structured) decomposable circuit

⇒ *possibly with negative outputs*

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

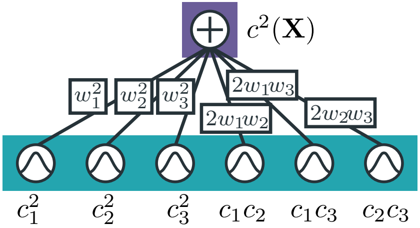
squaring shallow MMs



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

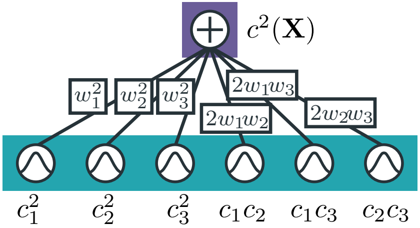
$\Longrightarrow$ *ensure non-negative output*

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



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

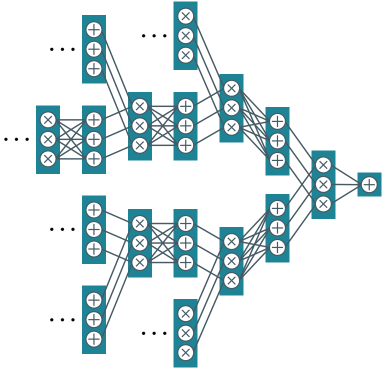
still a smooth and (str) decomposable PC with $\mathcal{O}(K^2)$ components!

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

wait...

"do negative parameters really boost expressiveness? and...always?"

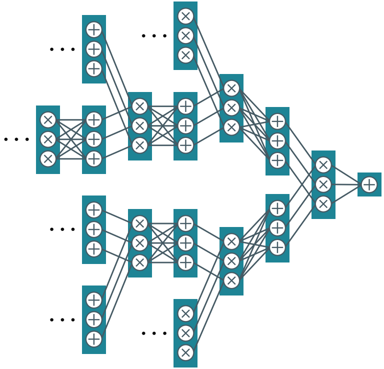
theorem

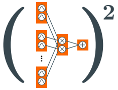


$\exists\, p$ **requiring exponentially large monotonic circuits...**

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

theorem

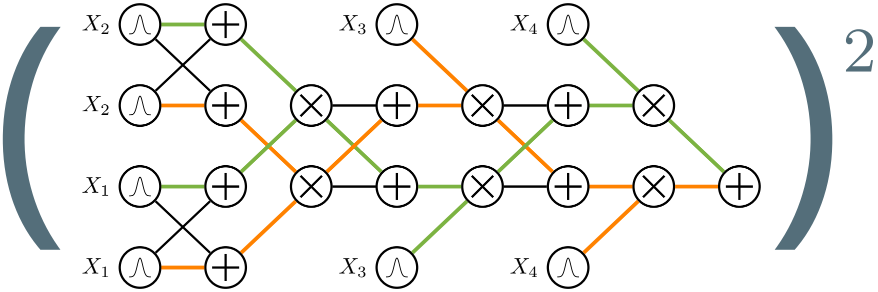




...but compact

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

```python
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)
```

probabilistic circuits (PCs)

the unit-wise definition

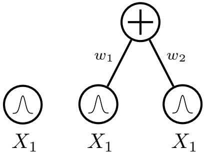
I. *A simple tractable function is a circuit*

probabilistic circuits (PCs)

the unit-wise definition

I. *A simple tractable function is a circuit*

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



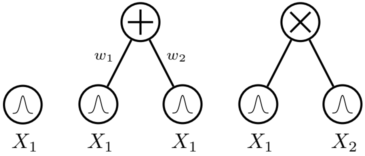
probabilistic circuits (PCs)

the unit-wise definition

I. *A simple tractable function is a circuit*

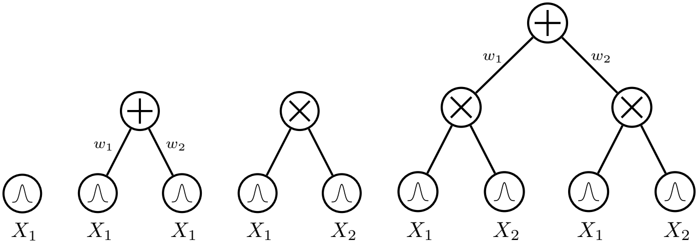
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



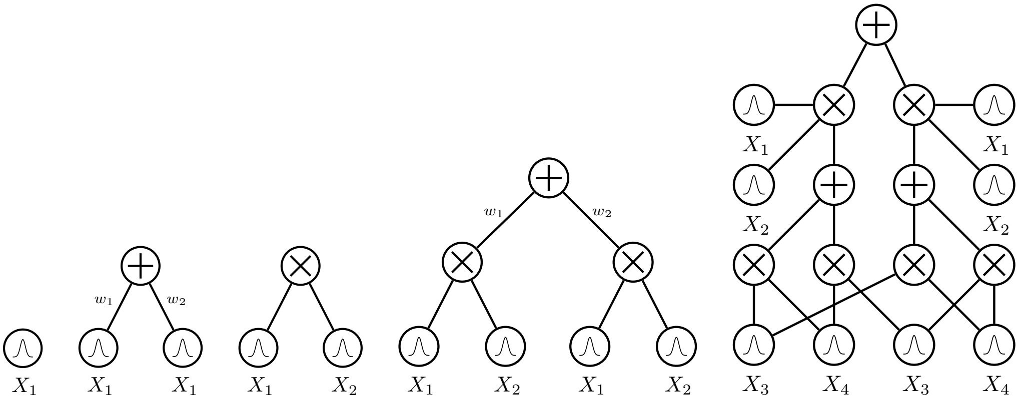
probabilistic circuits (PCs)

the unit-wise definition



probabilistic circuits (PCs)

the unit-wise definition



probabilistic circuits (PCs)

a tensorized definition

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

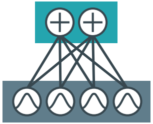
probabilistic circuits (PCs)

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)

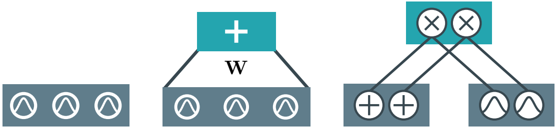
a tensorized definition

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)

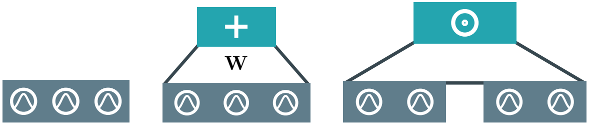
a tensorized definition

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)

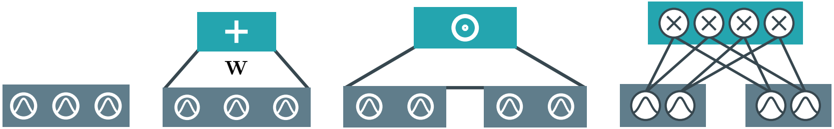
a tensorized definition

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)

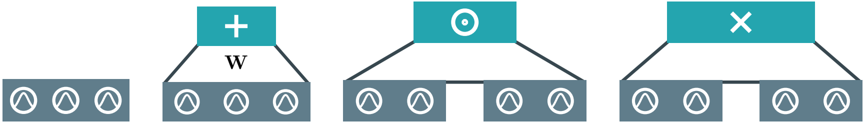
a tensorized definition

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

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

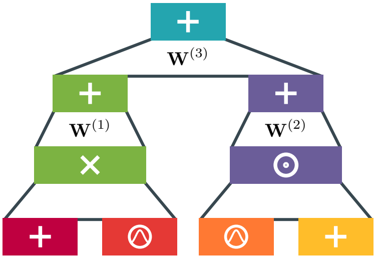
III. *The product of two layers is a circuit layer*

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



probabilistic circuits (PCs)

a tensorized definition



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

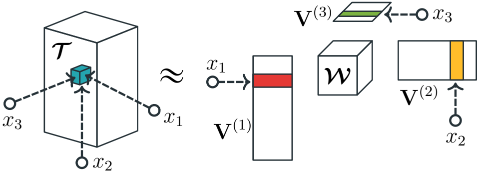
II. *A linear projection of a layer is a circuit layer*

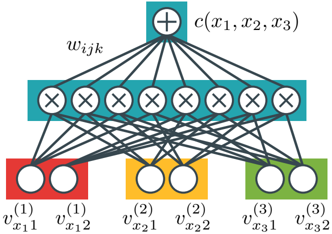
III. *The product of two layers is a circuit layer*

stack layers to build a deep circuit!

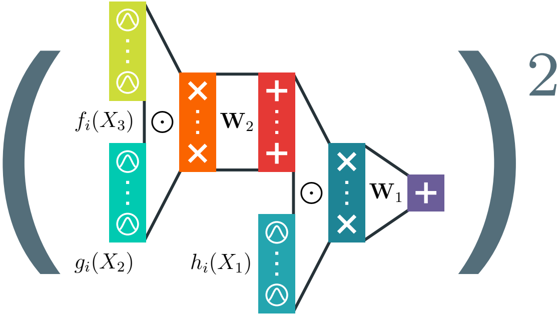
circuits layers

as tensor factorizations





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

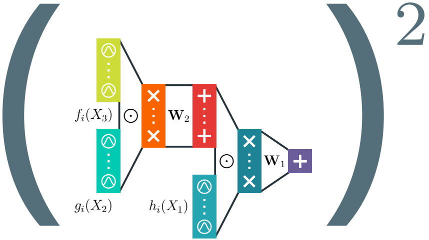


how to efficiently square (and *renormalize*) a deep PC?

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

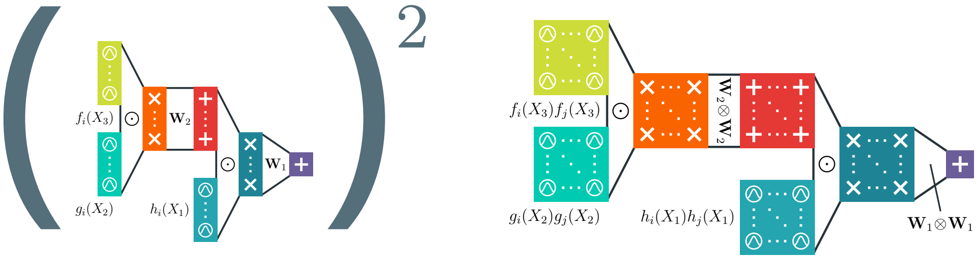
squaring deep PCs

the tensorized way



squaring deep PCs

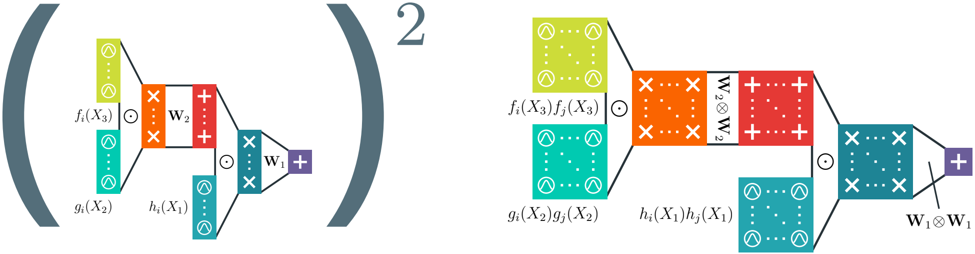
the tensorized way



squaring a circuit = squaring layers

squaring deep PCs

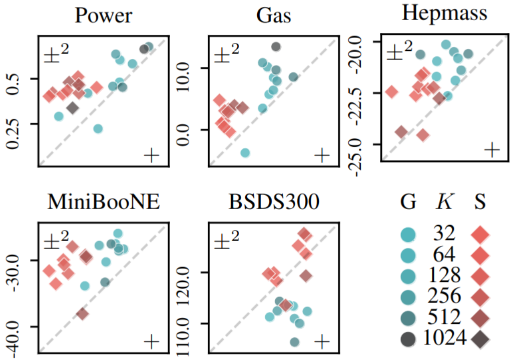
the tensorized way

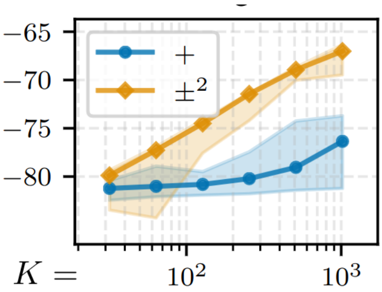


exactly compute $\int c(\mathrm{x})c(\mathrm{x})dX$ **in time** $O(LK^2)$

how more expressive?

for the ML crowd

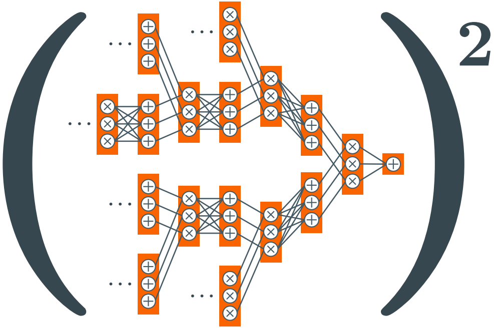




theorem

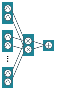
$\exists\, p$ **requiring exponentially large**

squared non-mono circuits...



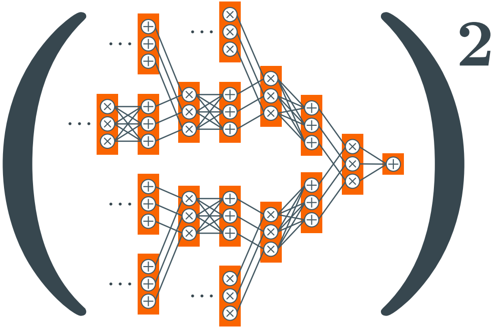
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem

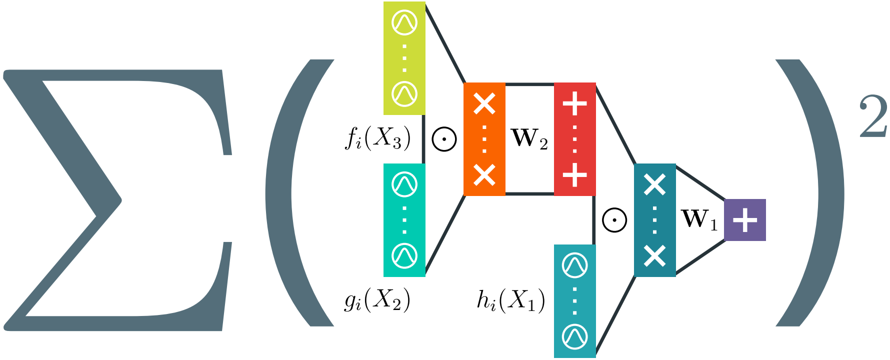


...but compact

monotonic circuits...!

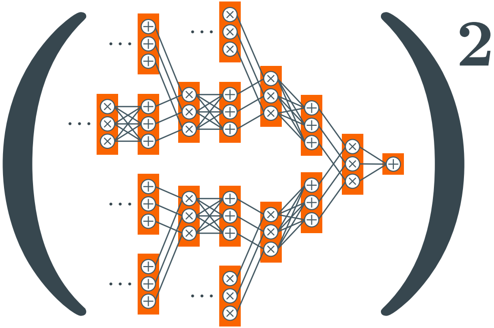


Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



what if we use more that one square?

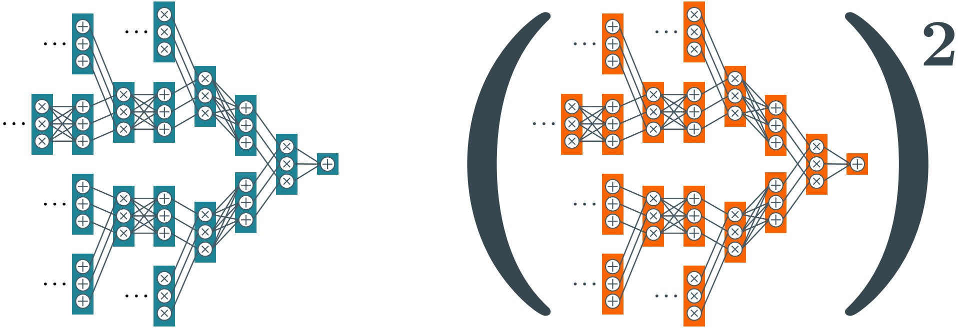
theorem



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

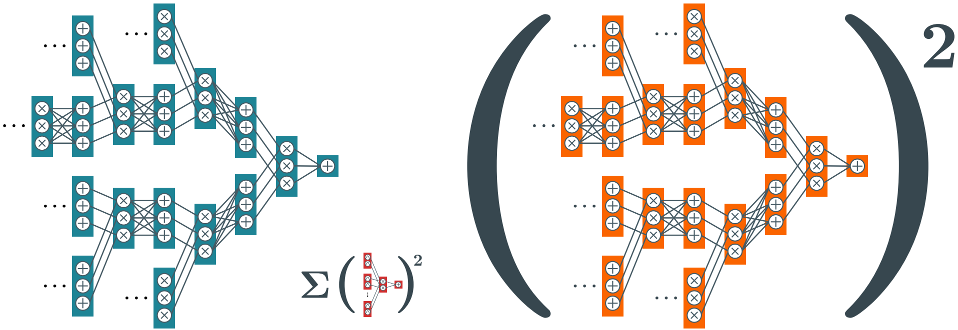
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem



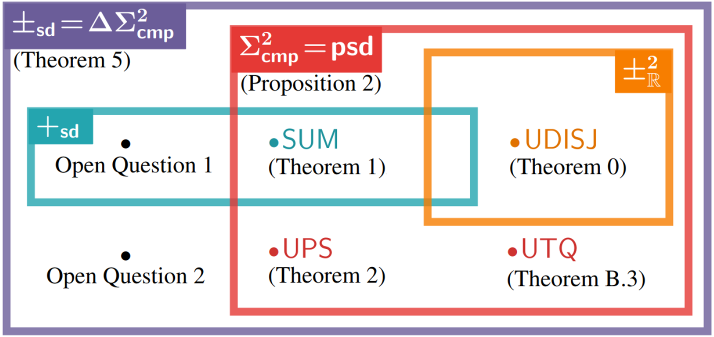
...exponentially large monotonic circuits...

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



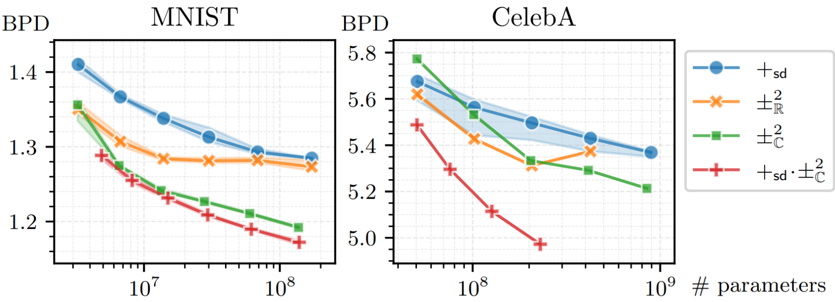
...but compact **SOS circuits...!**

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



complex circuits are SOS (and scale better!)

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

compositional inference I

```
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)
```

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

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?*

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

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

$\implies$ *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

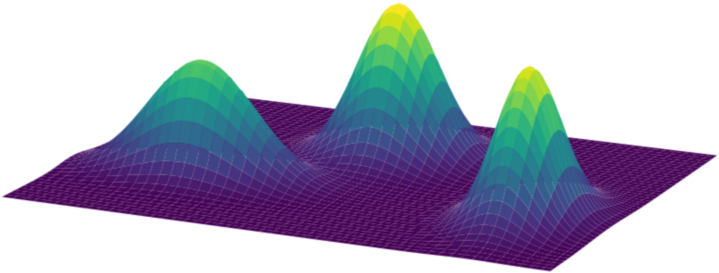
approximate inference

difference of expectation estimator

		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

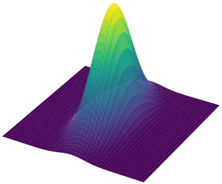
faster than autoregressive sampling

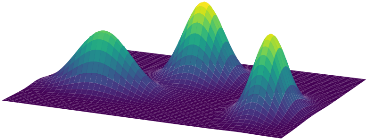
Zellinger et al., "Scalable Expectation Estimation with Subtractive Mixture Models", Under submission, 2025



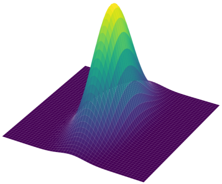
oh mixtures, you're so fine you blow my mind!

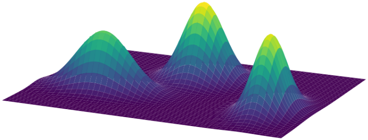
image taken from Hao Tang's course on ASR





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

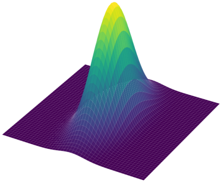


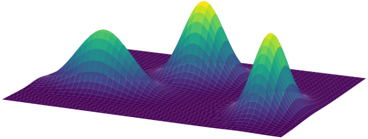


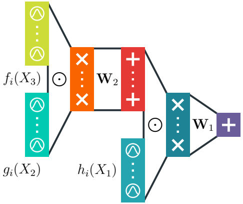
$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

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

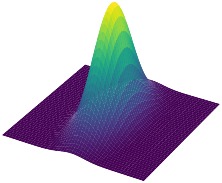
A. Vergari

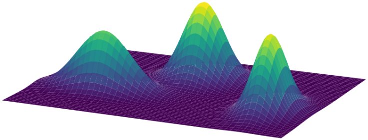


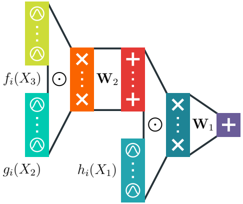




$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \longrightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

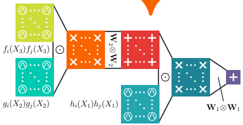






$$p(\mathbf{X}) \rightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \rightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

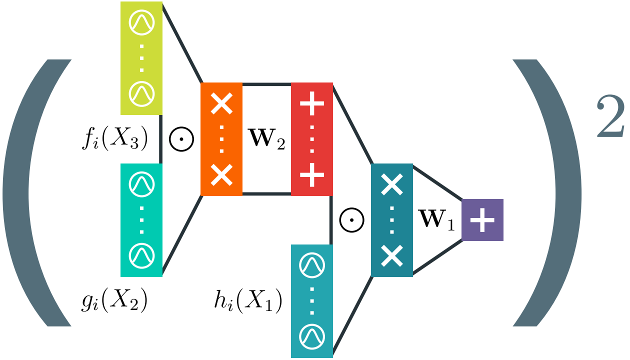
$$\left(\sum_{i=1}^{K} w_i p_i(\mathbf{X})\right)^2 \quad w_i \in \mathbb{R} \rightarrow$$



cirkit

learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`



questions?