subtractive mixture models
representation, learning & inference

antonio vergari (he/him)

@tetraduzione

4th Apr 2025 - **3rd GeMSS** *Sophia Antipolis*

april

april-tools.github.io

april

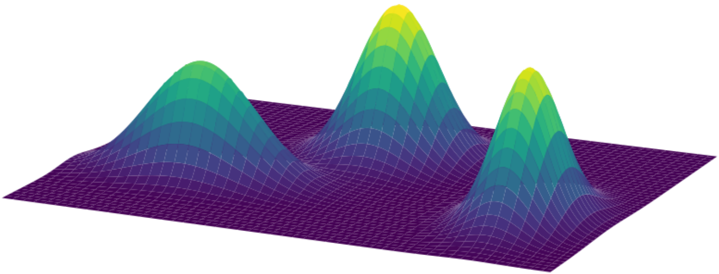
about
probabilities
integrals &
logic

april

autonomous &
provably
reliable
intelligent
learners

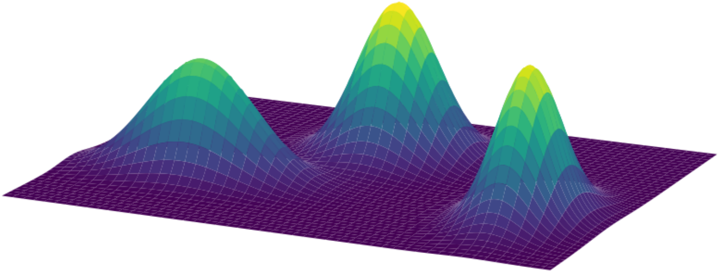
april

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



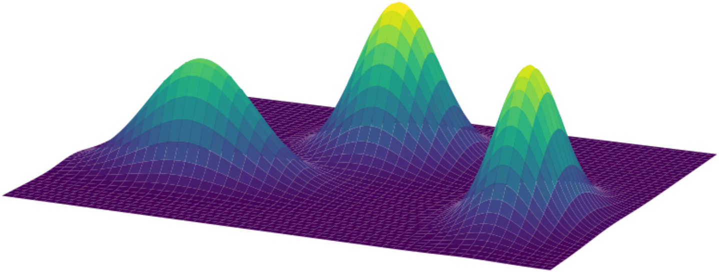
who knows mixture models?

image taken from Hao Tang's course on ASR



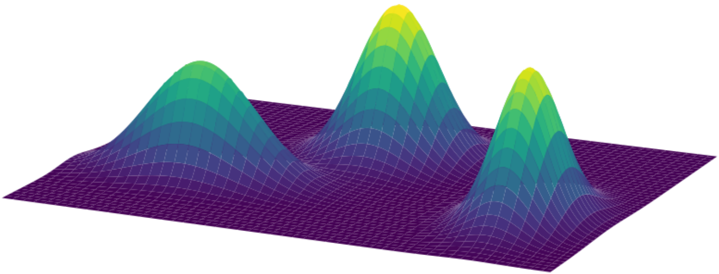
who loves mixture models?

image taken from Hao Tang's course on ASR



a brief recap...

image taken from Hao Tang's course on ASR

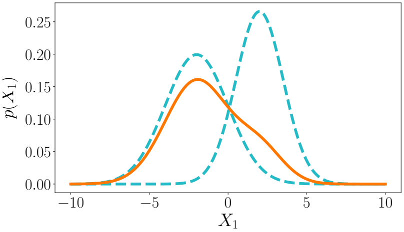


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

GMMs

as computational graphs

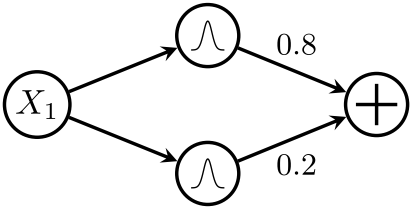


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

⟹ *translating inference to data structures...*

GMMs

as computational graphs

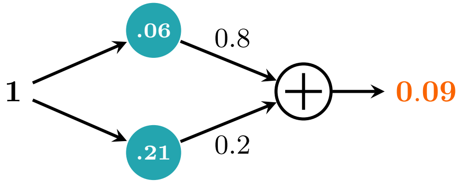


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

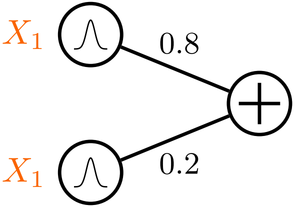


$$p(X=1) = 0.2 \cdot p_1(X_1 = 1) + 0.8 \cdot p_2(X_1 = 1)$$

$\Longrightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



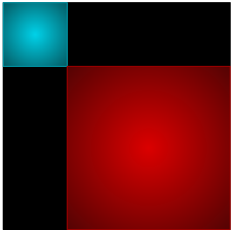
A simplified notation:

⇒ ***scopes*** *attached to inputs*

⇒ *edge directions omitted*

GMMs

as computational graphs

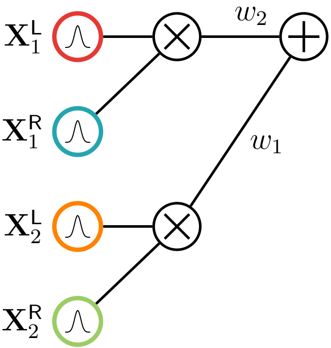


$$p(\mathbf{X}) = w_1 \cdot p_1(\mathbf{X}_1^{\mathsf{L}}) \cdot p_1(\mathbf{X}_1^{\mathsf{R}}) + w_2 \cdot p_2(\mathbf{X}_2^{\mathsf{L}}) \cdot p_2(\mathbf{X}_2^{\mathsf{R}})$$

$\Longrightarrow$ *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*

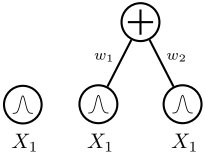
⇒ *e.g., a multivariate Gaussian or orthonormal polynomial*

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*



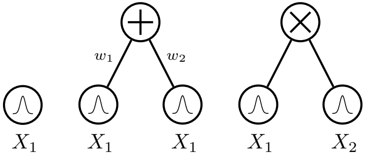
probabilistic circuits (PCs)

a grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

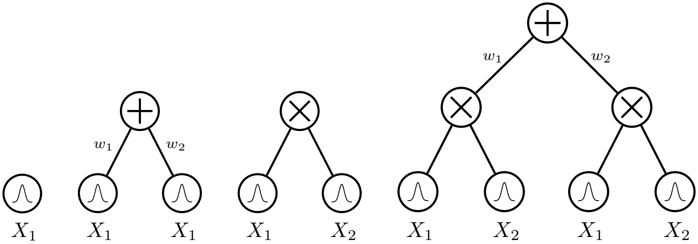
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



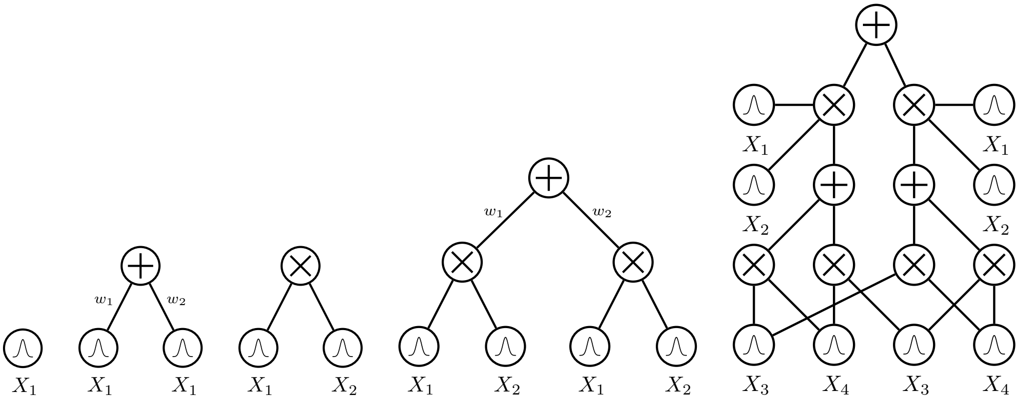
probabilistic circuits (PCs)

a grammar for tractable computational graphs



probabilistic circuits (PCs)

a grammar for tractable computational graphs



probabilistic circuits (PCs)

a tensorized definition

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

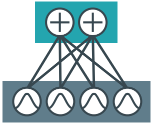
probabilistic circuits (PCs)

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)

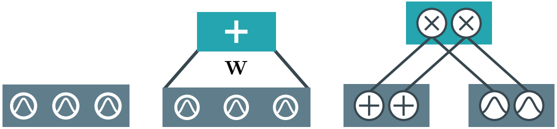
a tensorized definition

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)

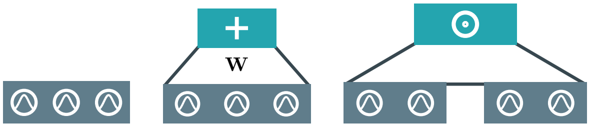
a tensorized definition

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)

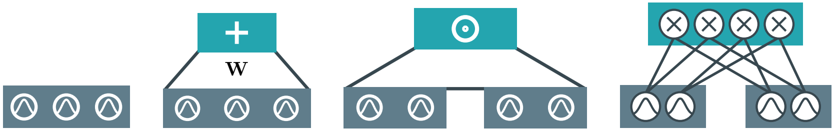
a tensorized definition

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)

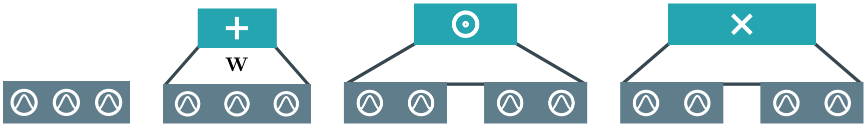
a tensorized definition

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

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

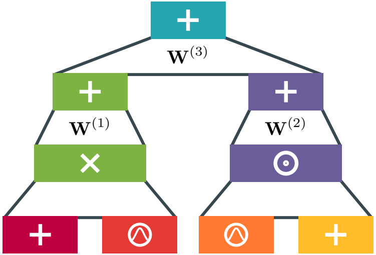
III. *The product of two layers is a circuit layer*

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



probabilistic circuits (PCs)

a tensorized definition



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

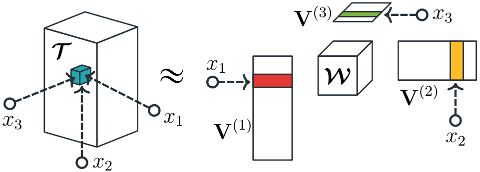
II. *A linear projection of a layer is a circuit layer*

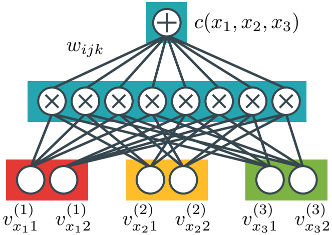
III. *The product of two layers is a circuit layer*

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

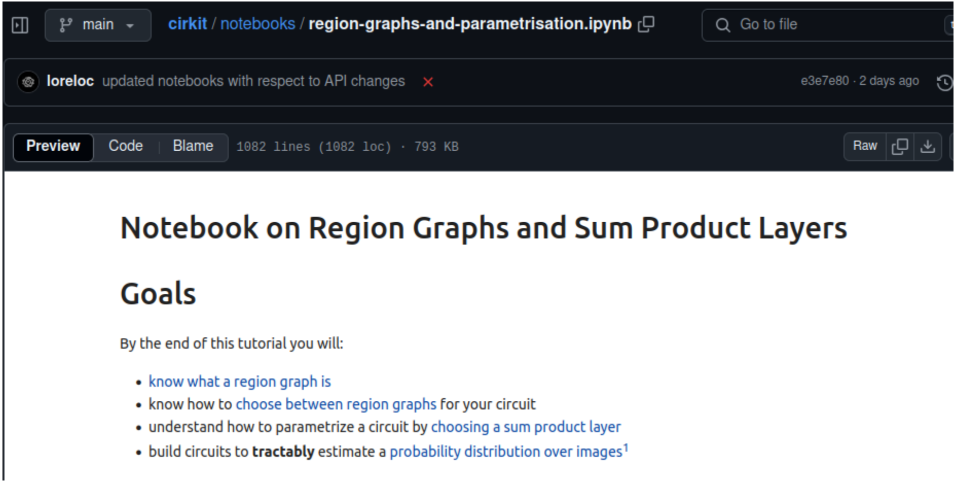
cirkit

learning & reasoning with circuits in pytorch

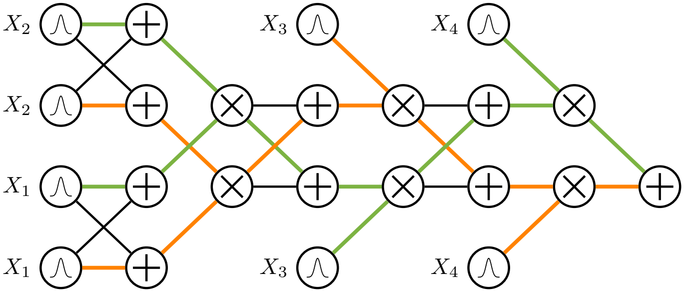
`github.com/april-tools/cirkit`

```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'
    )
)
```

```python
from cirkit.pipeline import compile
circuit = compile(symbolic_circuit)

with torch.no_grad():
    test_lls = 0.0
    for batch, _ in test_dataloader:
        batch = batch.to(device).unsqueeze(dim=1)
        log_likelihoods = circuit(batch)
        test_lls += log_likelihoods.sum().item()
    average_ll = test_lls / len(data_test)
    bpd = -average_ll / (28 * 28 * np.log(2.0))
    print(f"Average LL: {average_ll:.3f}") # Average LL: -682.916
    print(f"Bits per dim: {bpd:.3f}") # Bits per dim: 1.257
```

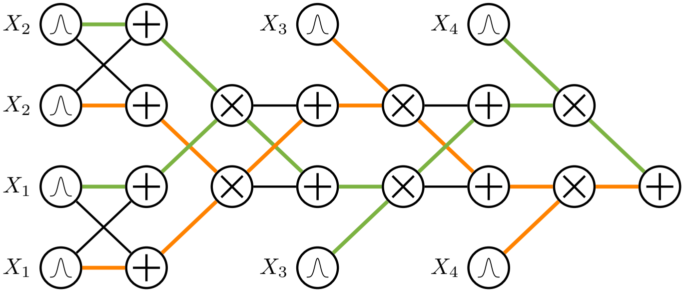

main
cirkit / notebooks / region-graphs-and-parametrisation.ipynb
Go to file
loreloc updated notebooks with respect to API changes
e3e7e80 · 2 days ago
Preview
Code
Blame
1082 lines (1082 loc) · 793 KB
Raw
Notebook on Region Graphs and Sum Product Layers
Goals
By the end of this tutorial you will:
know what a region graph is
know how to choose between region graphs for your circuit
understand how to parametrize a circuit by choosing a sum product layer
build circuits to **tractably** estimate a probability distribution over images[1]

deep mixtures



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

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

structural properties

property A

property B

property C

structural properties

smoothness

decomposability

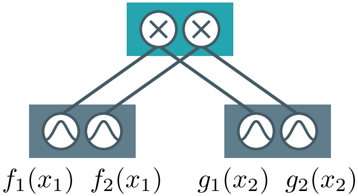
property C

smoothness $\wedge$ decomposability

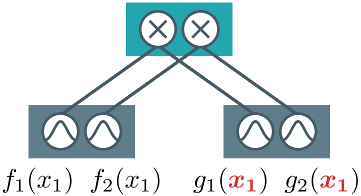
$\implies$ multilinearity

Multilinearity in circuits

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



✓ **multilinear**

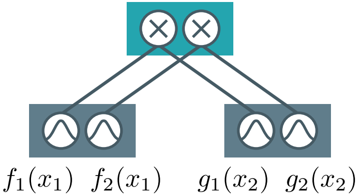


✗ **not multilinear**

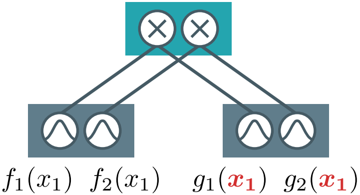
Darwiche and Marquis, "A knowledge compilation map", , 2002

Multilinearity in circuits

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



decomposable circuit

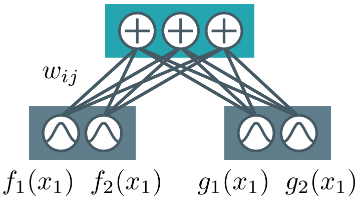


non-decomposable circuit

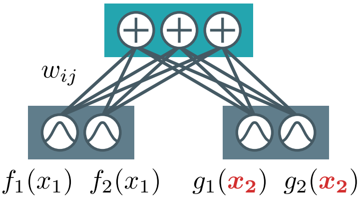
Darwiche and Marquis, "A knowledge compilation map", , 2002

Multilinearity in circuits

the inputs of sum units are defined over the same variables



✓ **multilinear**

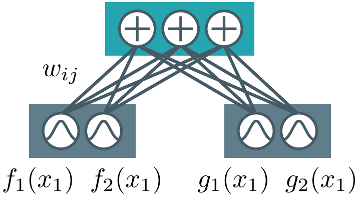


✗ **not multilinear**

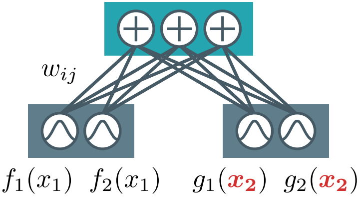
Darwiche and Marquis, "A knowledge compilation map", , 2002

Multilinearity in circuits

the inputs of sum units are defined over the same variables



smooth circuit



non-smooth circuit

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

structural properties

smoothness

decomposability

property C

smoothness $\wedge$ decomposability

$\implies$ multilinearity

structural properties

smoothness

decomposability

property C

tractable computation of **arbitrary integrals** in probabilistic circuits

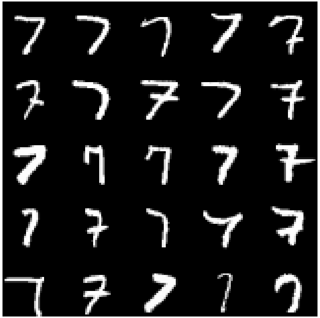
$$p(\mathbf{y}) = \int p(\mathbf{y}, \mathbf{z})\, d\mathbf{z}, \quad \forall \mathbf{Y} \subseteq \mathbf{X}, \quad \mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$$

$\Longrightarrow$ tractable partition function

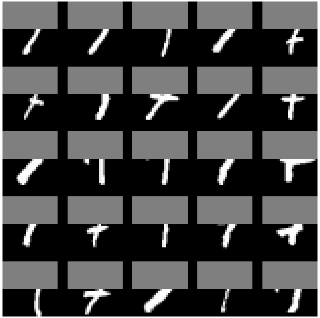
$\Longrightarrow$ also any conditional is tractable

tractable marginals on PCs

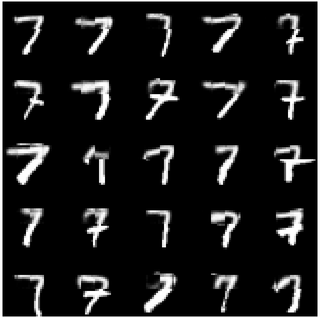
Original



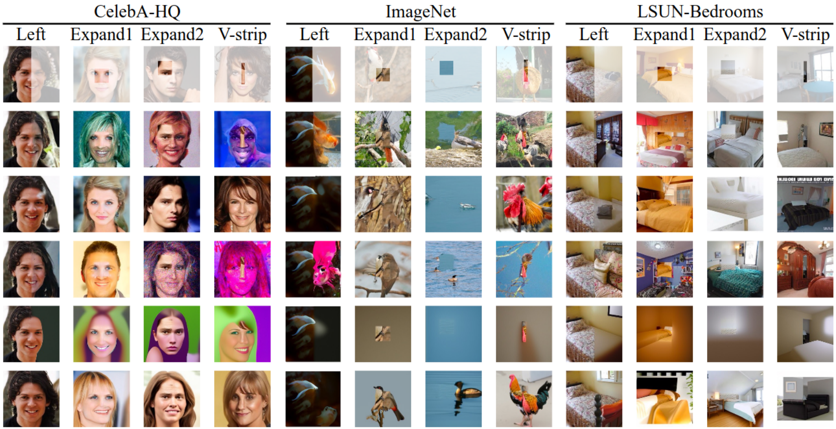
Missing



Conditional sample



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



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

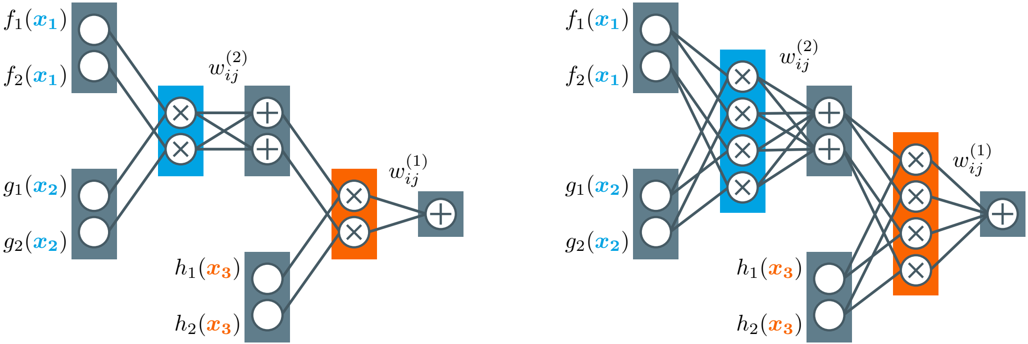
compatibility

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

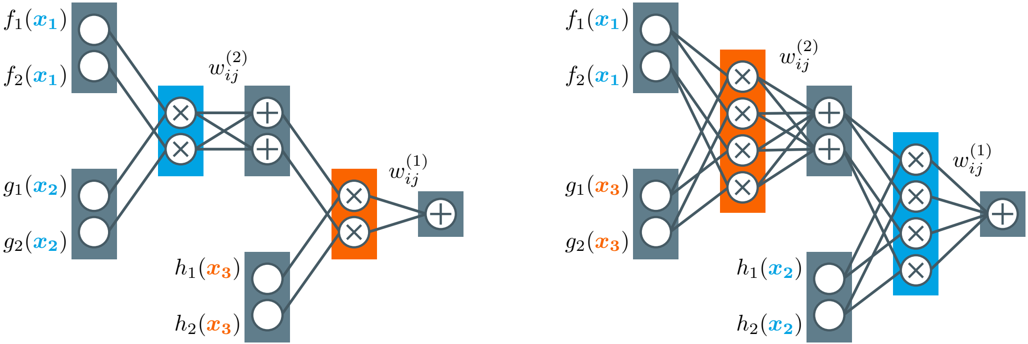
compatibility



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

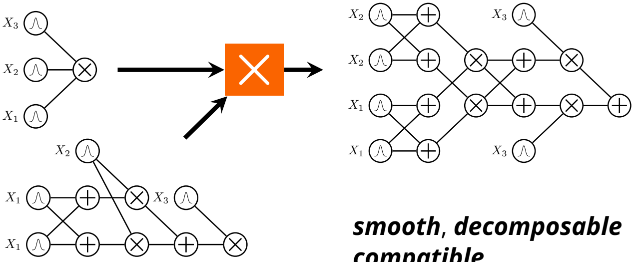
compatibility

$\Downarrow$

smoothness $\wedge$ **decomposability**

compatiblity $\Rightarrow$ tractable expectations

Tractable products

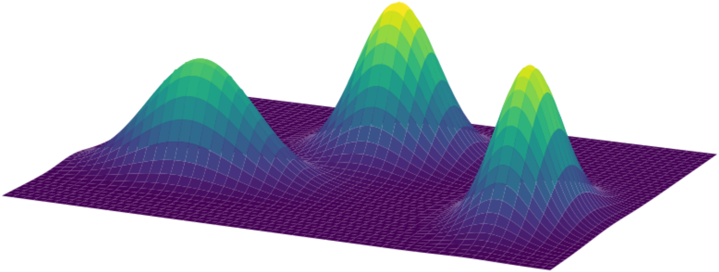


smooth**, **decomposable
compatible

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*

```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)
```

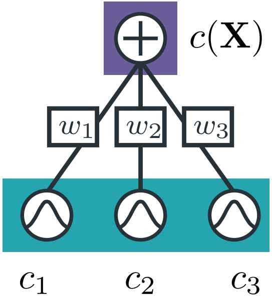


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

additive MMs

are so cool!



easily represented as shallow PCs

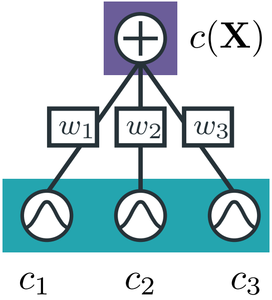
these are **monotonic** PCs

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

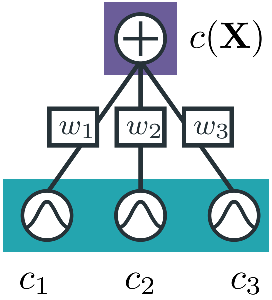
these are **monotonic** PCs

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

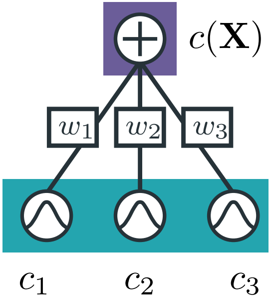
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



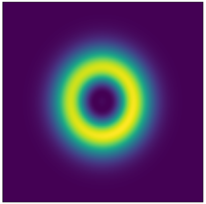
easily represented as shallow PCs

these are **monotonic** PCs

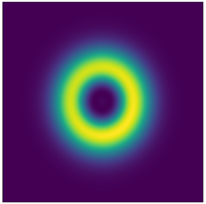
if marginals/conditionals are tractable for the components, they are tractable for the MM

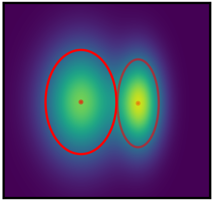
universal approximators...

however...



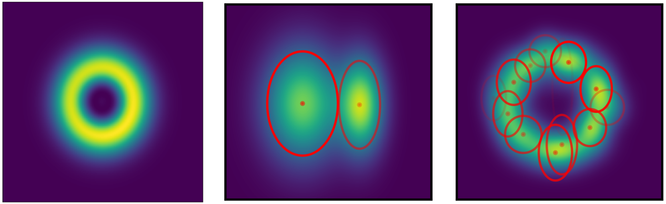
however...





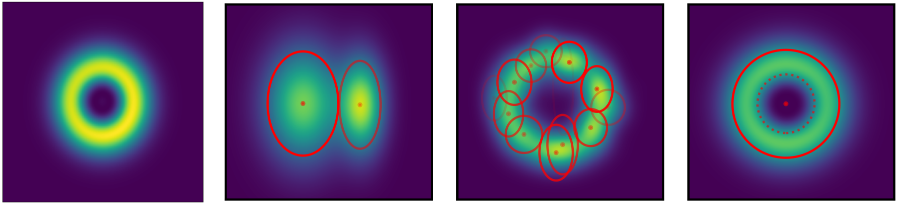
GMM ($K = 2$)

however...



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

however...



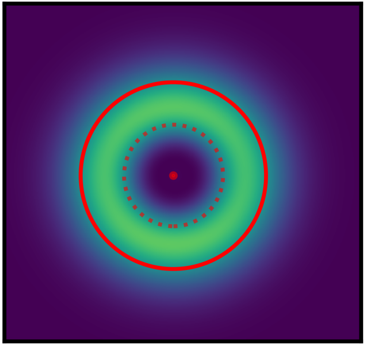
GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

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

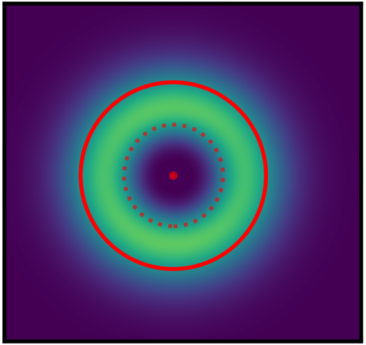
⇒ *or **non-monotonic** circuits,...*

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

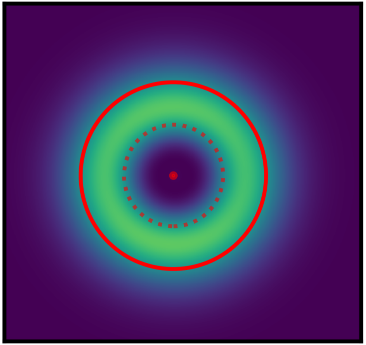
⇒ *or **non-monotonic** circuits,...*

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

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

issue: how to preserve non-negative outputs?

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

$\Longrightarrow$ *constraints on variance, mean*

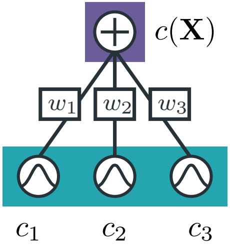
tl;dr

"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

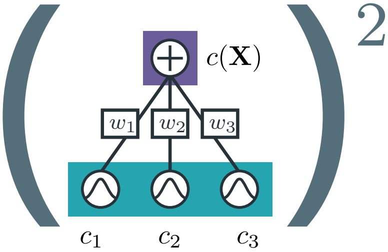


a **non-monotonic** smooth and (structured) decomposable circuit

$\Rightarrow$ *possibly with negative outputs*

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

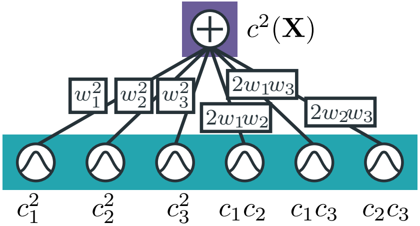
squaring shallow MMs



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

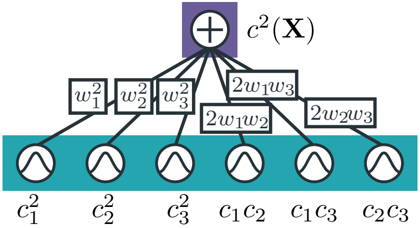
$\Longrightarrow$ *ensure non-negative output*

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

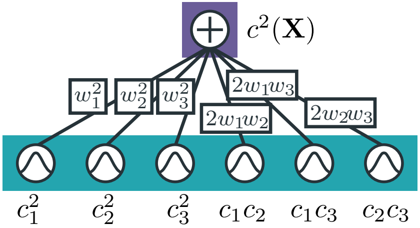


$$c^2(\mathbf{X}) = \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2$$
$$= \sum_{i=1}^{K} \sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})$$

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

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

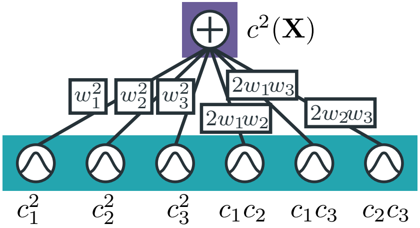
squaring shallow MMs



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

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

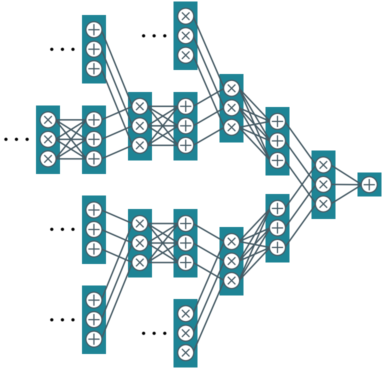
Lawrence K. Saul
Flatiron Institute
lsaul@flatironinstitute.org

orthonormal squared mixtures for VI

wait...

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

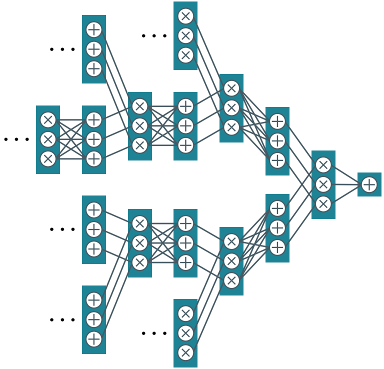
theorem

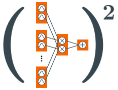


$\exists\, p$ **requiring exponentially large monotonic circuits...**

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

theorem

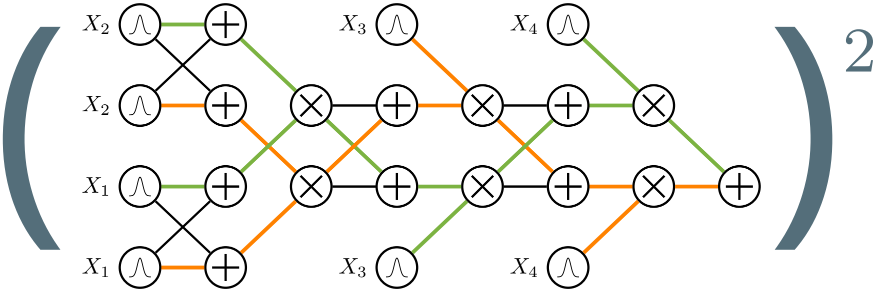




...but compact

squared non-monotonic circuits

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



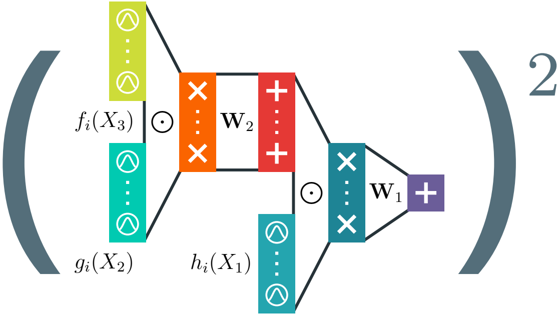
how to efficiently square (and *renormalize*) a deep PC?

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)
```

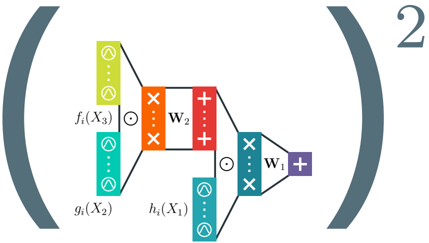


how to efficiently square (and *renormalize*) a deep PC?

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

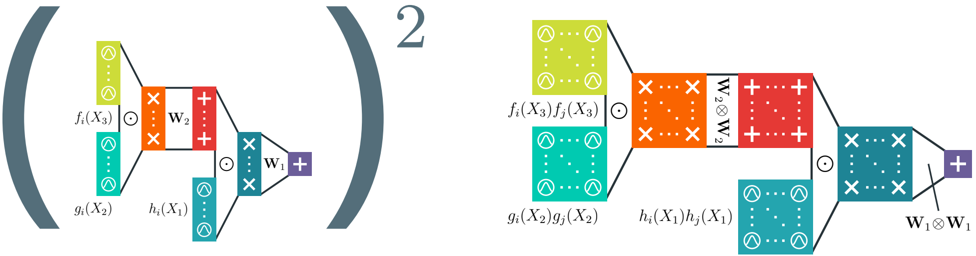
squaring deep PCs

the tensorized way



squaring deep PCs

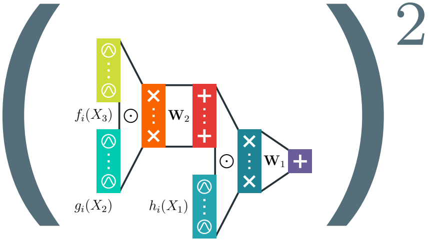
the tensorized way

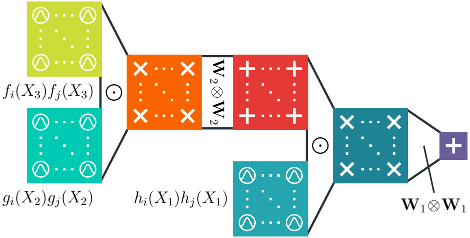


squaring a circuit = squaring layers

squaring deep PCs

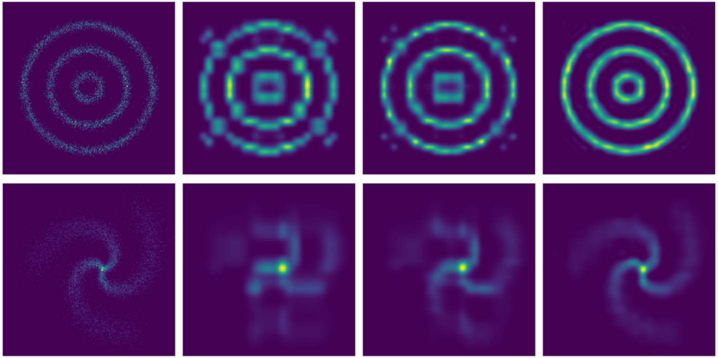
the tensorized way





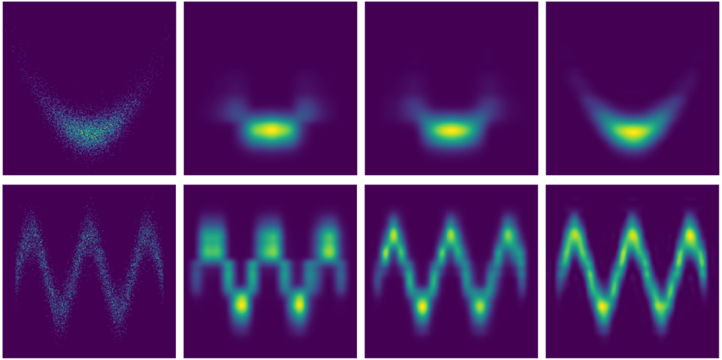
exactly compute $\int c(\mathrm{x})c(\mathrm{x})dX$ **in time** $O(LK^2)$

more expressive?



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

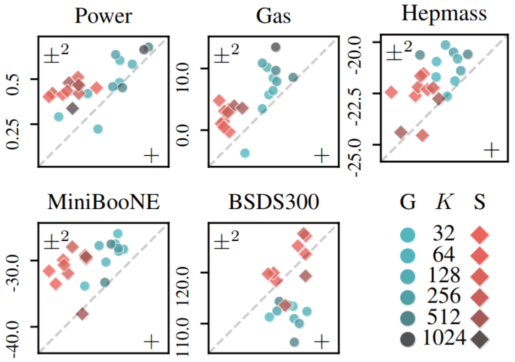
more expressive?

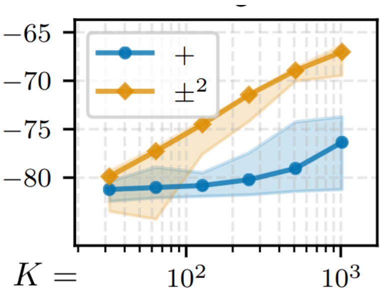


data | monoPC | monoPC^2 | $\text{non}-\text{monoPC}^2$

how more expressive?

real-world data

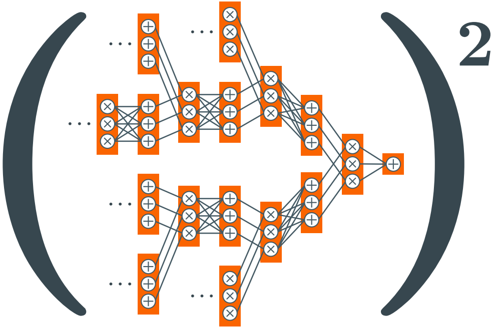




theorem

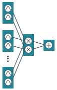
$\exists\, p$ **requiring exponentially large**

squared non-mono circuits...



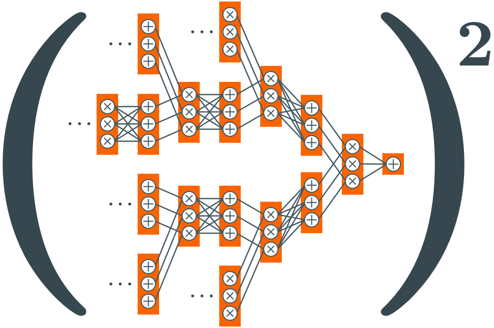
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

theorem

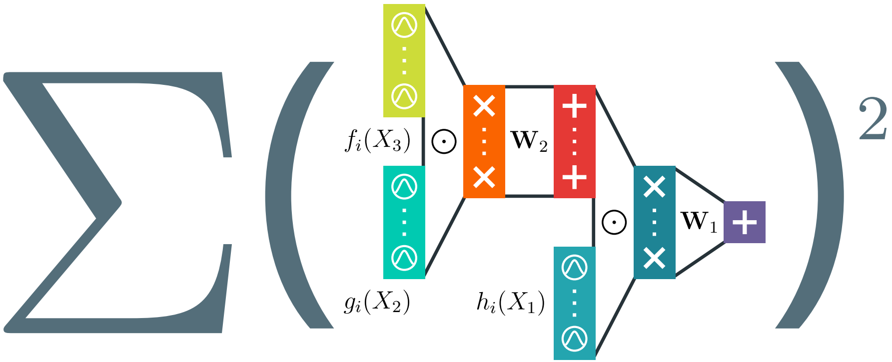


...but compact

monotonic circuits...!

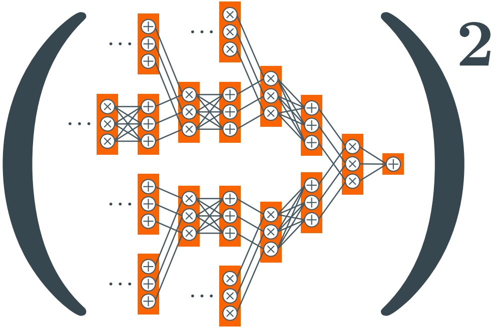


Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025



what if we use more that one square?

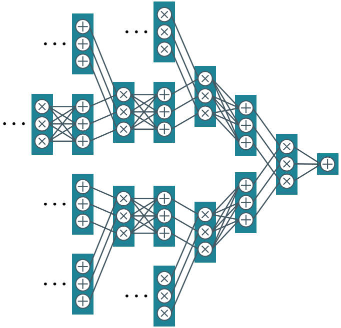
theorem

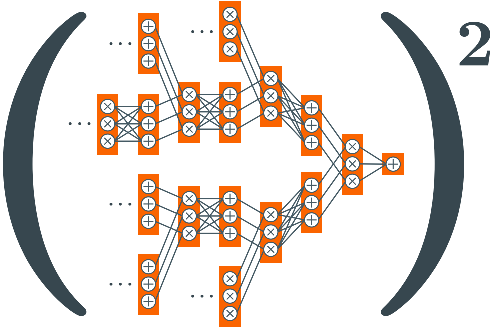


$\exists\, p$ **requiring exponentially large** **squared non-mono circuits...**

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

theorem

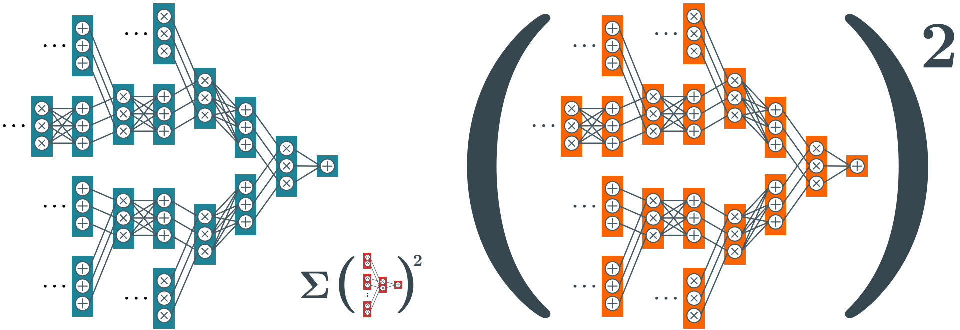




...exponentially large monotonic circuits...

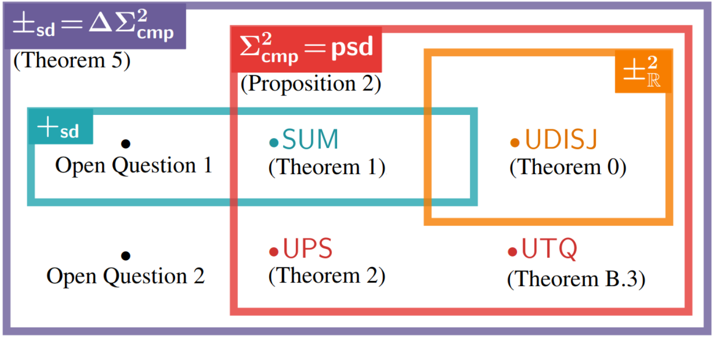
Loconte, Mengel, and Vergari, "Sum of Squares Circuits", AAAI, 2025

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:

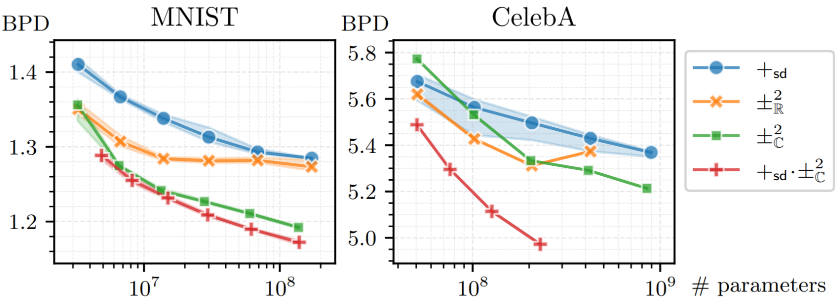
$$c^2(\mathbf{x}) = c(\mathbf{x})^\dagger c(\mathbf{x}), \qquad c(\mathbf{x}) \in \mathbb{C}$$

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"

⇒ ***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)
```

On Faster Marginalization with Squared Circuits via Orthonormalization

Lorenzo Loconte[1] **Antonio Vergari**[1]

[1] School of Informatics, University of Edinburgh, UK
l.loconte@sms.ed.ac.uk, avergari@ed.ac.uk

what about deep orthonormal mixtures
and arbitrary marginals?

On Faster Marginalization with Squared Circuits via Orthonormalization

Lorenzo Loconte[1] **Antonio Vergari**[1]

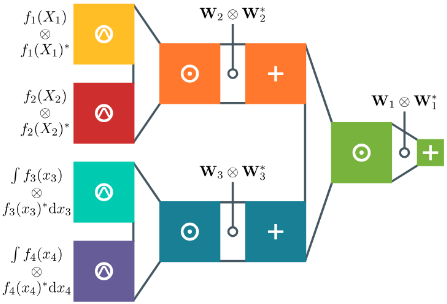
[1] School of Informatics, University of Edinburgh, UK
l.loconte@sms.ed.ac.uk, avergari@ed.ac.uk

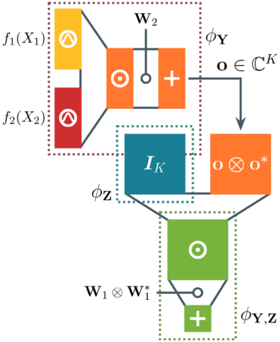
it sufficies to orthonormalize each layer!

layers depending on both $\mathbf{Y}$ and $\mathbf{Z}$

Time complexity $\mathcal{O}(|\phi_{\mathbf{Y}}|S + |\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$

layers depending on $\mathbf{Y}$ only





faster marginalization of arbitrary subsets of features

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?

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

$\Longrightarrow$ *but how to sample from 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

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

$\Longrightarrow$ *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"*

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

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)
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

Number of components (K)

faster than autoregressive sampling

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

...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 ...*

...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 ...*

3. Realiable neuro-symbolic AI

logical constraints as circuits, multiplied to probabilistic circuits

Semantic Probabilistic Layers for Neuro-Symbolic Learning

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

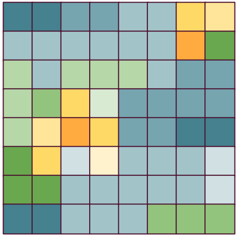
Stefano Teso
CIMeC and DISI
University of Trento
stefano.teso@unitn.it

Kai-Wei Chang
CS Department
UCLA
kwchang@cs.ucla.edu

Guy Van den Broeck
CS Department
UCLA
guyvdb@cs.ucla.edu

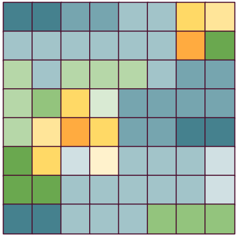
Antonio Vergari
School of Informatics
University of Edinburgh
avergari@ed.ac.uk

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

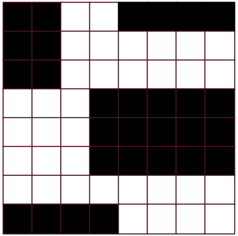


$q(\mathbf{x})$

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

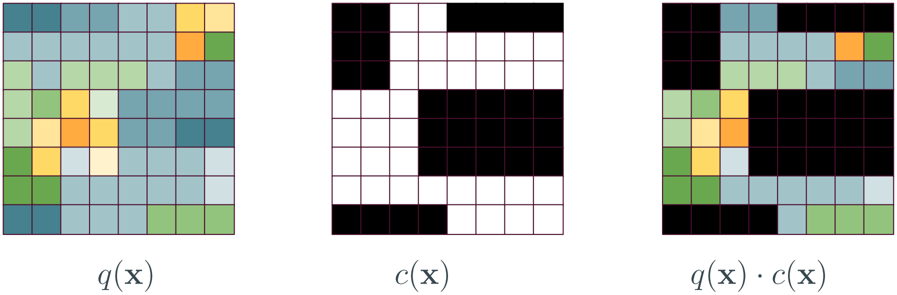


$q(\mathbf{x})$

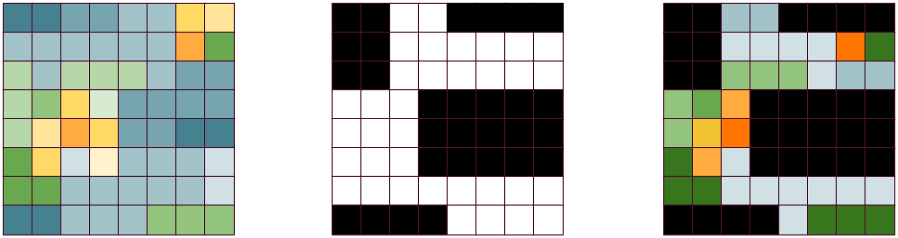


$c(\mathbf{x})$

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

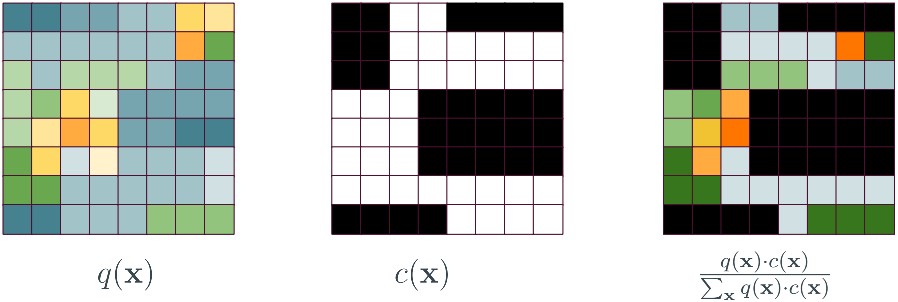


by multiplying them $q(\mathrm{x})c(\mathrm{x})$**...**



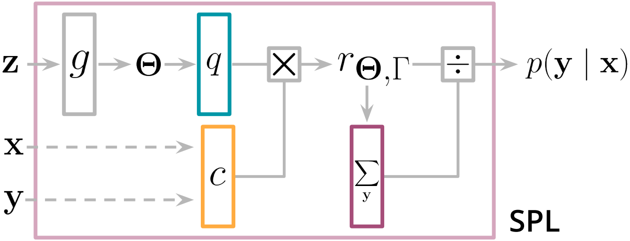
$q(\mathbf{x})$ $c(\mathbf{x})$ $\frac{q(\mathbf{x})\cdot c(\mathbf{x})}{\sum_{\mathbf{x}} q(\mathbf{x})\cdot c(\mathbf{x})}$

and then renormalizing them!



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

SPL

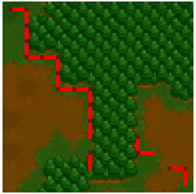


$$p(\mathbf{y} \mid \mathbf{x}) = q_{\Theta}(\mathbf{y} \mid g(\mathbf{z})) \cdot c_{\mathsf{K}}(\mathbf{x}, \mathbf{y}) / \mathcal{Z}(\mathbf{x})$$

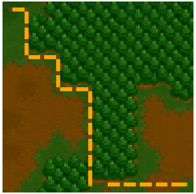
$$\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}} q_{\Theta}(\mathbf{y} \mid \mathbf{x}) \cdot c_{\mathsf{K}}(\mathbf{x}, \mathbf{y})$$

SPL

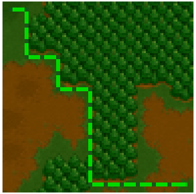
Ground Truth



ResNet-18



Semantic Loss



circuits

predictions guarantee a logical constraint 100% of the time!

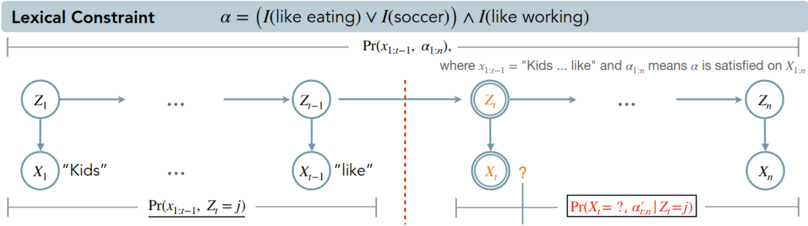
SPL

(and variants)

everywhere

Tractable Control for Autoregressive Language Generation

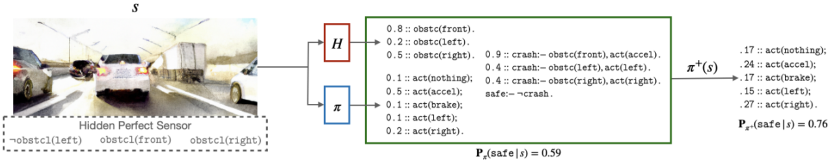
Honghua Zhang[*1] Meihua Dang[*1] Nanyun Peng[1] Guy Van den Broeck[1]



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

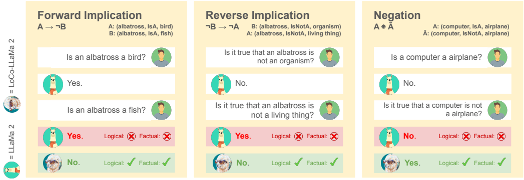
Nicola Di Mauro
University of Bari, Italy
nicola.dimauro@uniba.it

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

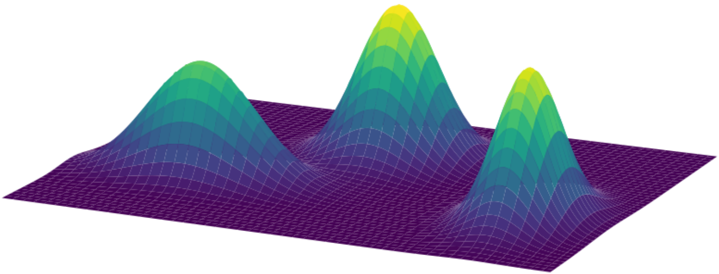


improving logical (self-)consistency in LLMs at ICLR 2025

cirkit

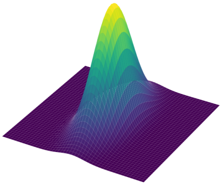
learning & reasoning with circuits in pytorch

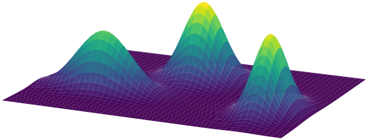
`github.com/april-tools/cirkit`



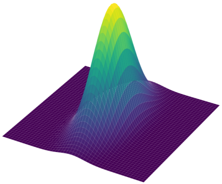
oh mixtures, you’re so fine you blow my mind!

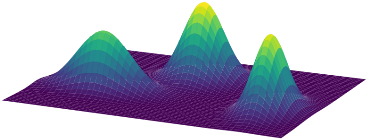
image taken from Hao Tang’s course on ASR





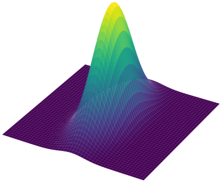
$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$$

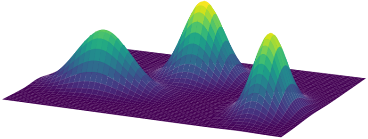


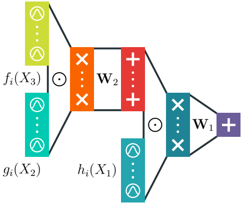


$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0$

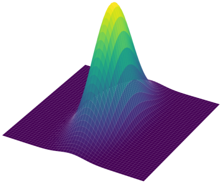
*“if someone publishes a paper on **model A**, there will be a paper about **mixtures of A** soon, with high probability”* **A. Vergari**

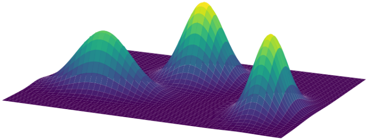


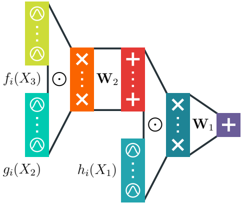




$$p(\mathbf{X}) \longrightarrow \sum_{i=1}^{K} w_i p_i(\mathbf{X}) \quad w_i > 0 \longrightarrow \sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$





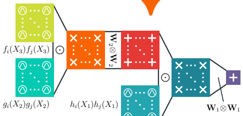


$$p(\mathbf{X})$$

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

$$\sum_{i=1}^{2^D} w_i p_i(\mathbf{X}) = \mathsf{PC}(\mathbf{X})$$

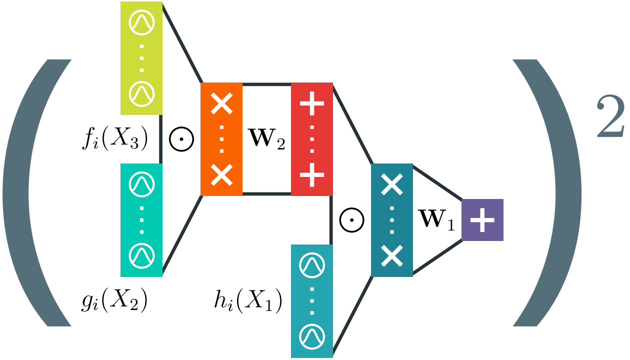
$$\left(\sum_{i=1}^{K} w_i p_i(\mathbf{X})\right)^2 \quad w_i \in \mathbb{R}$$



cirkit

learning & reasoning with circuits in pytorch

`github.com/april-tools/cirkit`



questions?