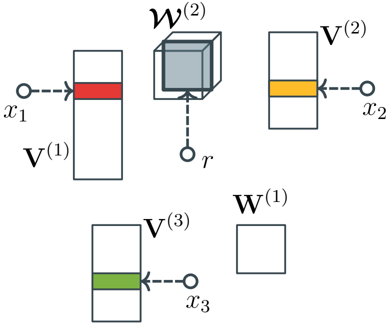
from tensor factorizations to circuits (and back)

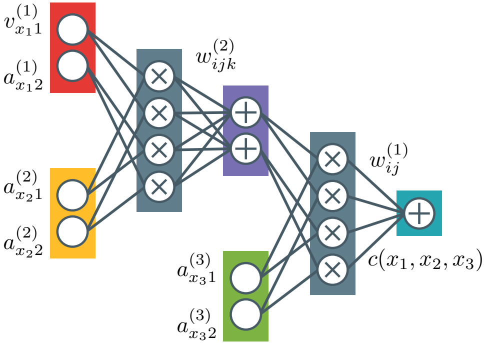
lorenzo loconte
@loreloc
@loreloc_

antonio vergari
@nolovedeeplearning
@tetraduzione

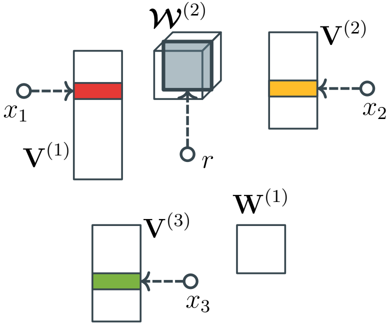
25th Feb 2025 – **AAAI-25** – *Philadelphia*



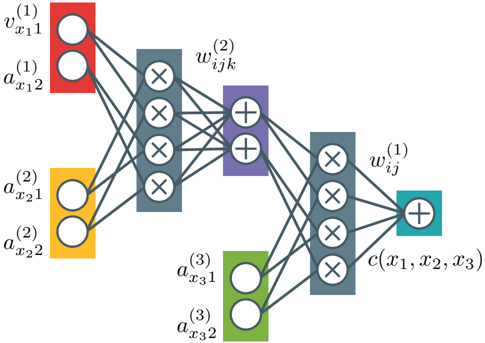
tensor factorizations



circuits

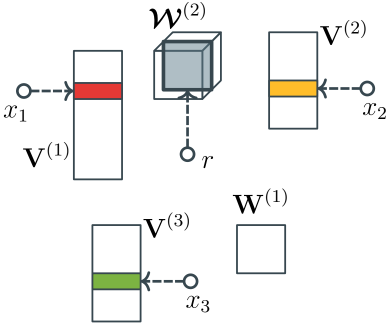


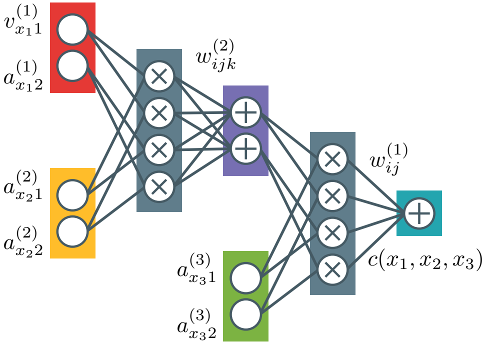
tensor factorizations



circuits

they look quite different...!?





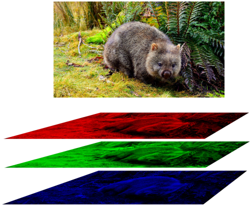
tensor factorizations

circuits

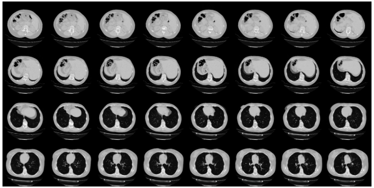
what can one take from another...!?

why tensor factorizations? (1/4)

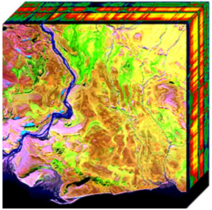
high-dimensional data as tensors



(PBS Nature)



(H. Zunair)

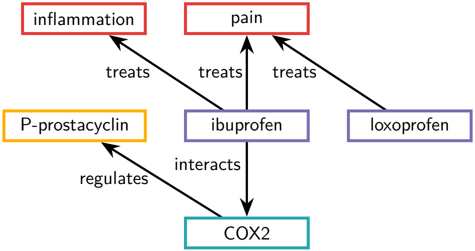


(N. M. Short)

Panagakis et al., "Tensor Methods in Computer Vision and Deep Learning", 2021
Wang et al., "Tensor Decompositions for Hyperspectral Data Processing in Remote Sensing: A Comprehensive Review", 2022

why tensor factorizations? (2/4)

graphs as tensors

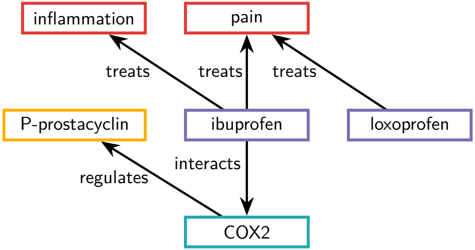


Entities:

- Drugs
- Symptoms
- Proteins
- Functions

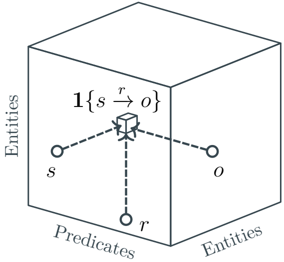
why tensor factorizations? (2/4)

graphs as tensors



Entities:

- Drugs
- Symptoms
- Proteins
- Functions

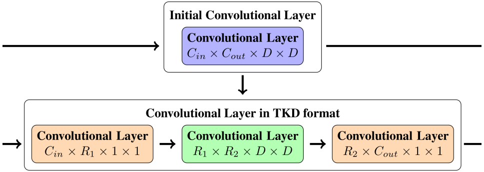


Knowledge base as Boolean tensor

[Nickel et al. 2016]

why tensor factorizations? (3/4)

compress ML models

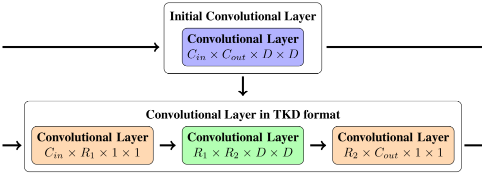


Compress convolutional layers

[Phan et al. 2020]

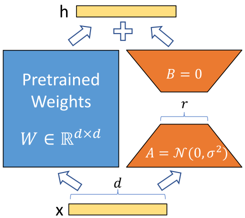
why tensor factorizations? (3/4)

compress ML models



Compress convolutional layers

[Phan et al. 2020]

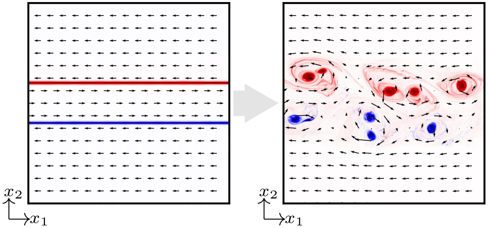


Low-rank adapters in LLMs

[Hu et al. 2022] [Bershatsky et al. 2024]

why tensor factorizations? (4/4)

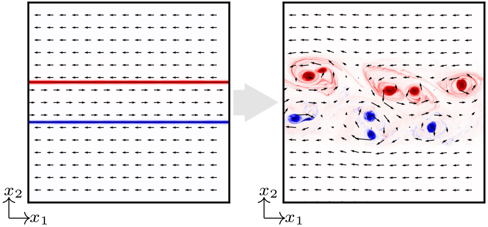
tensors 4 physics



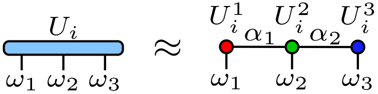
Fluid velocity vectors computed in exponentially many points...

why tensor factorizations? (4/4)

tensors 4 physics



Fluid velocity vectors computed in exponentially many points...

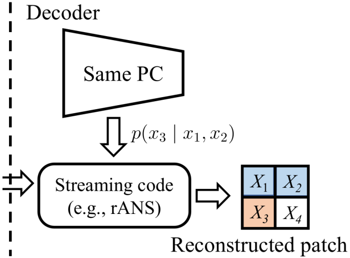


...by factorizing them into chains of low-rank tensors

[Gourianov et al. 2022] [Hölscher et al. 2025]

why circuits? (1/3)

efficient probabilistic inference

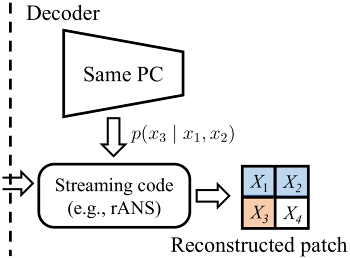


Fast lossless (de)compression

[Liu, Mandt, and Van den Broeck 2022]

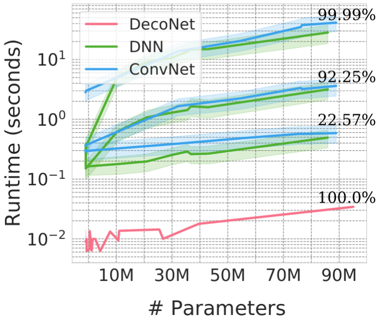
why circuits? (1/3)

efficient probabilistic inference



Fast lossless (de)compression

[Liu, Mandt, and Van den Broeck 2022]

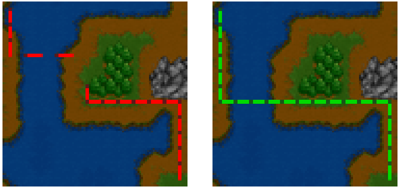


Efficient robustness to adversarial attacks

[Subramani et al. 2021]

why circuits? (2/3)

they enable neuro-symbolic AI

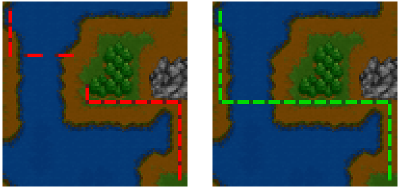


Constrained multi-label prediction (w/ guarantees)

[Ahmed et al. 2022]

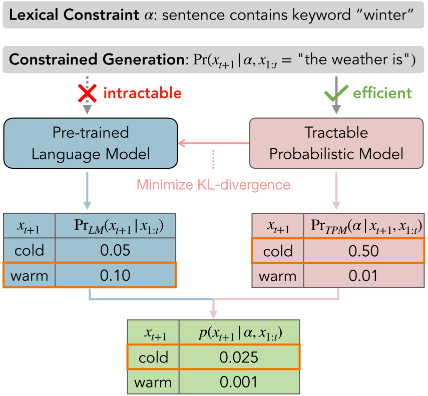
why circuits? (2/3)

they enable neuro-symbolic AI



Constrained multi-label prediction (w/ guarantees)

[Ahmed et al. 2022]

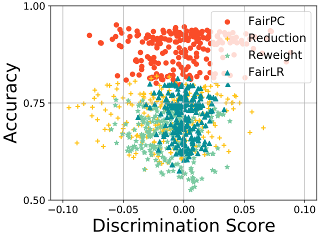


Constrained text generation

[Zhang et al. 2023]

why circuits? (3/3)

they are reliable and interpretable

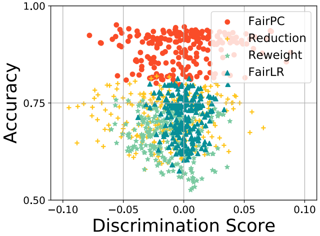


Encode group fairness

[Choi, Dang, and Van den Broeck 2020]

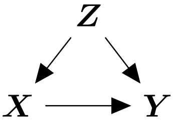
why circuits? (3/3)

they are reliable and interpretable



Encode group fairness

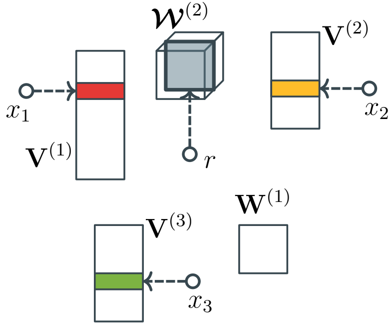
[Choi, Dang, and Van den Broeck 2020]

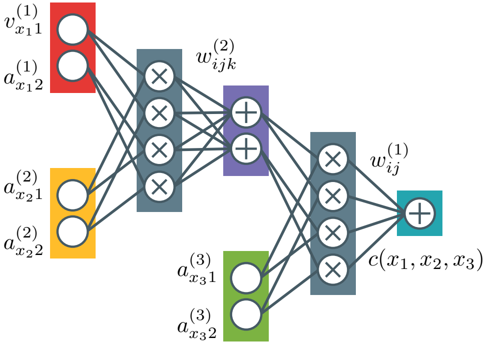


$$\sum_{\mathbf{Z}} p(\mathbf{Z}) p(\mathbf{Y} \mid \mathbf{X}, \mathbf{Z})$$

Tractable causal inference

[Wang and Kwiatkowska 2023]



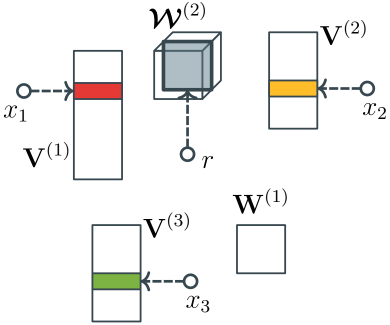


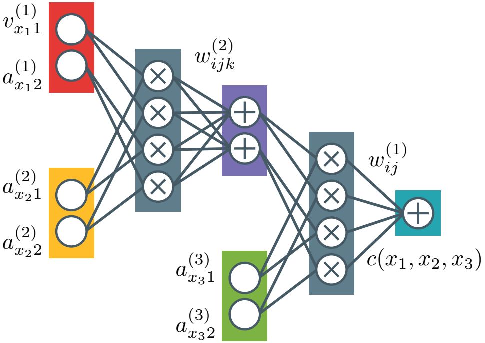
tensor factorizations

tensor compression, graph data
physics-inspired AI, speed up LLMs...

circuits

property-driven fast inference
neuro-symbolic, trustworthy AI...





tensor factorizations **circuits**

two faces of the same coin...!

outline

1 connecting ***tensor factorizations*** and ***circuits***

outline

1 connecting *tensor factorizations* and *circuits*

2 a *unifying pipeline* to build factorizations & circuits

outline

1 connecting *tensor factorizations* and *circuits*

2 a *unifying pipeline* to build factorizations & circuits

3 a *property-driven* approach to inference & reasoning

outline

1. connecting *tensor factorizations* and *circuits*
2. a *unifying pipeline* to build factorizations & circuits
3. a *property-driven* approach to inference & reasoning
4. *expressiveness analysis*: known and new results

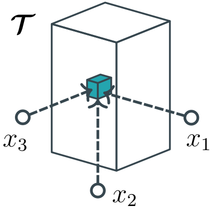
outline

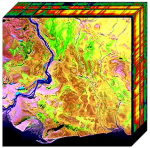
1 connecting ***tensor factorizations*** and ***circuits***

tl;dr

"Understand when and how
a tensor factorization can be
exactly encoded as a circuit representation"

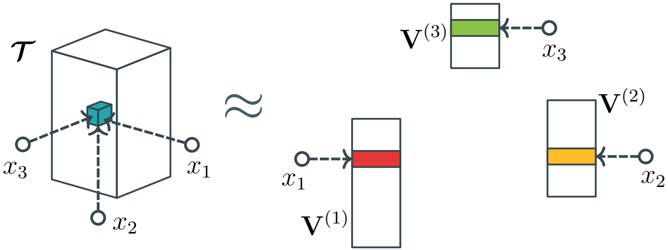
Canonical polyadic (CP)





(N. M. Short)

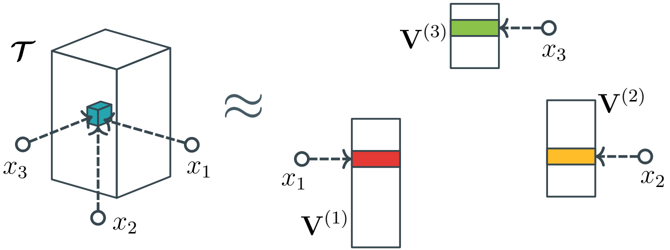
Canonical polyadic (CP)

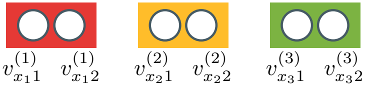


$$t_{x_1 x_2 x_3} \approx \sum_{r=1}^{R} v^{(1)}_{x_1 r} v^{(2)}_{x_2 r} v^{(3)}_{x_3 r}$$

$$\mathbf{V}^{(1)} \in \mathbb{R}^{I_1 \times R}, \mathbf{V}^{(2)} \in \mathbb{R}^{I_2 \times R}, \mathbf{V}^{(3)} \in \mathbb{R}^{I_3 \times R}$$

Canonical polyadic (CP)

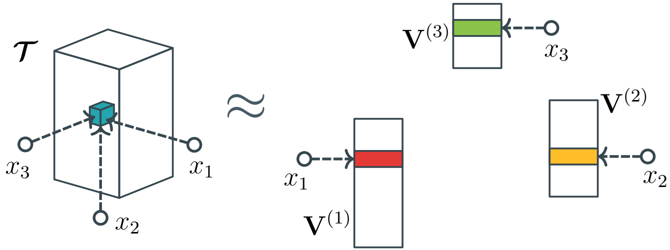


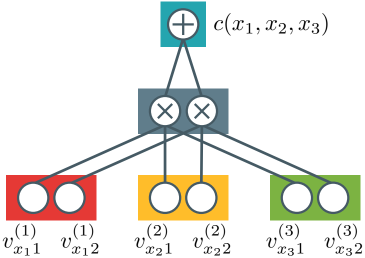


$$t_{x_1 x_2 x_3} \approx \sum_{r=1}^{R} v^{(1)}_{x_1 r} \, v^{(2)}_{x_2 r} \, v^{(3)}_{x_3 r}$$

$$\mathbf{V}^{(1)} \in \mathbb{R}^{I_1 \times R}, \mathbf{V}^{(2)} \in \mathbb{R}^{I_2 \times R}, \mathbf{V}^{(3)} \in \mathbb{R}^{I_3 \times R}$$

Canonical polyadic (CP)





$$t_{x_1 x_2 x_3} \approx \sum_{r=1}^{R} v^{(1)}_{x_1 r} v^{(2)}_{x_2 r} v^{(3)}_{x_3 r}$$

$$\mathbf{V}^{(1)} \in \mathbb{R}^{I_1 \times R}, \mathbf{V}^{(2)} \in \mathbb{R}^{I_2 \times R}, \mathbf{V}^{(3)} \in \mathbb{R}^{I_3 \times R}$$

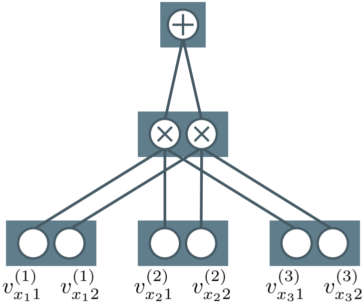
how to evaluate a circuit?

a circuit computes a tensor entry at some index

$$\mathbf{V}^{(1)} = \begin{bmatrix} 0.1 & 1.2 \\ 3.5 & -0.2 \\ -0.1 & 0.2 \end{bmatrix}$$

$$\mathbf{V}^{(2)} = \begin{bmatrix} 2.5 & 0.0 \\ -3.4 & -0.5 \\ -0.1 & 2.2 \end{bmatrix}$$

$$\mathbf{V}^{(3)} = \begin{bmatrix} -2.3 & 1.0 \\ 0.8 & -2.4 \\ 0.7 & 1.5 \end{bmatrix}$$



$$t_{x_1 x_2 x_3} \approx c(x_1, x_2, x_3) = \sum_{r=1}^{R} v^{(1)}_{x_1 r}\, v^{(2)}_{x_2 r}\, v^{(3)}_{x_3 r}$$

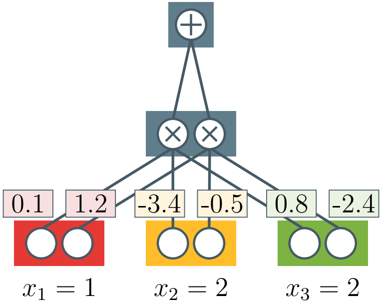
how to evaluate a circuit?

a circuit computes a tensor entry at some index

$$\mathbf{V}^{(1)} = \begin{bmatrix} \boxed{0.1} & \boxed{1.2} \\ 3.5 & -0.2 \\ -0.1 & 0.2 \end{bmatrix}$$

$$\mathbf{V}^{(2)} = \begin{bmatrix} 2.5 & 0.0 \\ \boxed{-3.4} & \boxed{-0.5} \\ -0.1 & 2.2 \end{bmatrix}$$

$$\mathbf{V}^{(3)} = \begin{bmatrix} -2.3 & 1.0 \\ \boxed{0.8} & \boxed{-2.4} \\ 0.7 & 1.5 \end{bmatrix}$$



$$t_{122} \approx c(1,2,2) = \sum_{r=1}^{R} v_{1r}^{(1)} \, v_{2r}^{(2)} \, v_{2r}^{(3)}$$

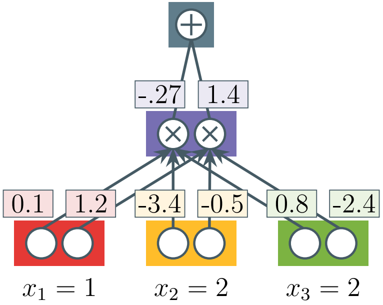
how to evaluate a circuit?

a circuit computes a tensor entry at some index

$$\mathbf{V}^{(1)} = \begin{bmatrix} 0.1 & 1.2 \\ 3.5 & -0.2 \\ -0.1 & 0.2 \end{bmatrix}$$

$$\mathbf{V}^{(2)} = \begin{bmatrix} 2.5 & 0.0 \\ -3.4 & -0.5 \\ -0.1 & 2.2 \end{bmatrix}$$

$$\mathbf{V}^{(3)} = \begin{bmatrix} -2.3 & 1.0 \\ 0.8 & -2.4 \\ 0.7 & 1.5 \end{bmatrix}$$



$$t_{122} \approx c(1,2,2) = \sum_{r=1}^{R} v_{1r}^{(1)} \, v_{2r}^{(2)} \, v_{2r}^{(3)}$$

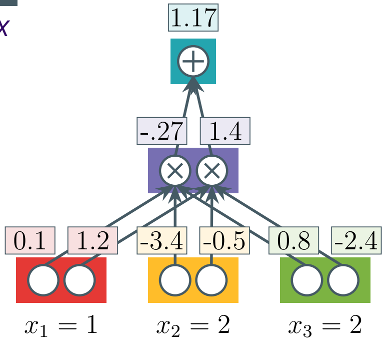
how to evaluate a circuit?

a circuit computes a tensor entry at some index

$$\mathbf{V}^{(1)} = \begin{bmatrix} 0.1 & 1.2 \\ 3.5 & -0.2 \\ -0.1 & 0.2 \end{bmatrix}$$

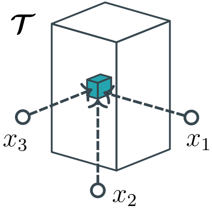
$$\mathbf{V}^{(2)} = \begin{bmatrix} 2.5 & 0.0 \\ -3.4 & -0.5 \\ -0.1 & 2.2 \end{bmatrix}$$

$$\mathbf{V}^{(3)} = \begin{bmatrix} -2.3 & 1.0 \\ 0.8 & -2.4 \\ 0.7 & 1.5 \end{bmatrix}$$

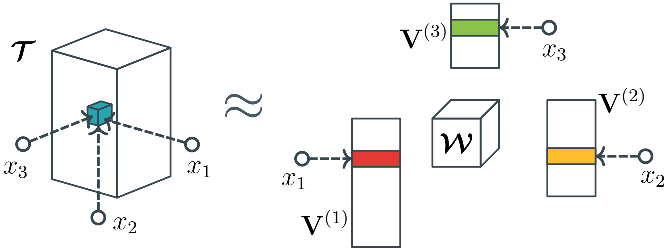


$$t_{122} \approx c(1,2,2) = \sum_{r=1}^{R} v_{1r}^{(1)} v_{2r}^{(2)} v_{2r}^{(3)}$$

Tucker



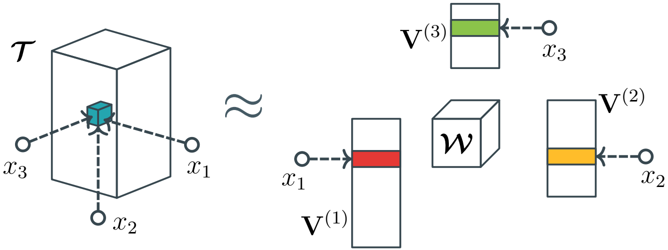
Tucker

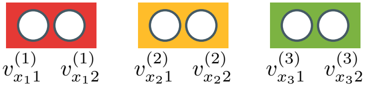


$$t_{x_1x_2x_3} \approx \sum_{r_1=1}^{R_1}\sum_{r_1=1}^{R_2}\sum_{r_1=1}^{R_3} w_{r_1r_2r_3}\, v^{(1)}_{x_1r_1}\, v^{(2)}_{x_2r_2}\, v^{(3)}_{x_3r_3}$$

$$\mathcal{W} \in \mathbb{R}^{R_1\times R_2\times R_3}$$

Tucker

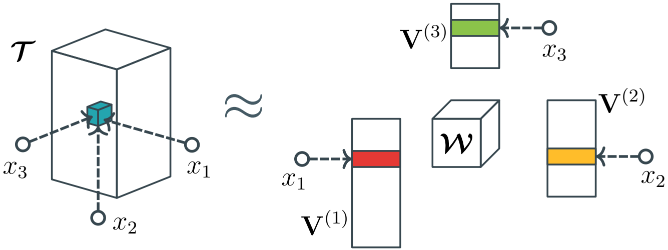


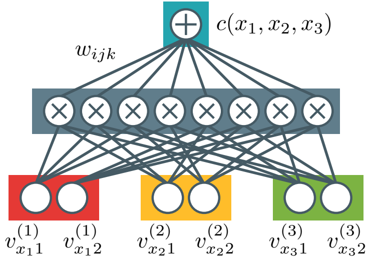


$$t_{x_1x_2x_3} \approx \sum_{r_1=1}^{R_1}\sum_{r_1=1}^{R_2}\sum_{r_1=1}^{R_3} w_{r_1r_2r_3}\, v^{(1)}_{x_1r_1}\, v^{(2)}_{x_2r_2}\, v^{(3)}_{x_3r_3}$$

$$\mathcal{W} \in \mathbb{R}^{R_1\times R_2\times R_3}$$

Tucker





$$t_{x_1x_2x_3} \approx \sum_{r_1=1}^{R_1} \sum_{r_1=1}^{R_2} \sum_{r_1=1}^{R_3} w_{r_1r_2r_3} \, v^{(1)}_{x_1r_1} \, v^{(2)}_{x_2r_2} \, v^{(3)}_{x_3r_3}$$

$$\mathcal{W} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$$

The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

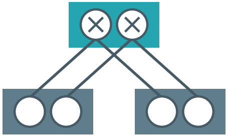
The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer
$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)

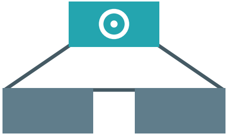


The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)



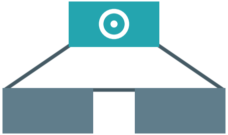
The layer-wise circuit definition

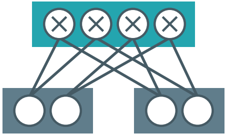
A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Kronecker)





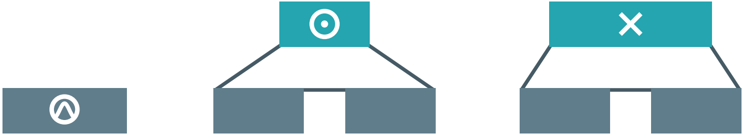
The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Kronecker)



The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Kronecker)

A linear projection of a layer is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \mathbf{W}\boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x})$

The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

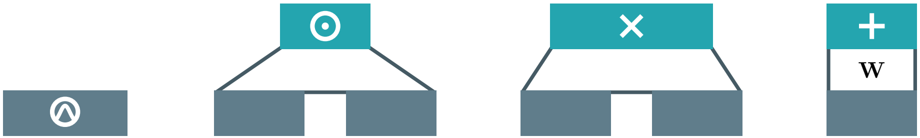
The product of two layers is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Hadamard)

$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x})$ (Kronecker)

A linear projection of a layer is a layer

$\boldsymbol{\ell}(\mathbf{x}) = \mathbf{W}\boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x})$



The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

$$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}) \qquad \text{(Hadamard)}$$

$$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}) \qquad \text{(Kronecker)}$$

A linear projection of two layers is a layer

$$\boldsymbol{\ell}(\mathbf{x}) = \mathbf{W}\mathrm{concat}(\boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}), \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}))$$

The layer-wise circuit definition

A collection of input units is a circuit layer $\boldsymbol{\ell}(\mathbf{x}) \in \mathbb{R}^K$

The product of two layers is a layer

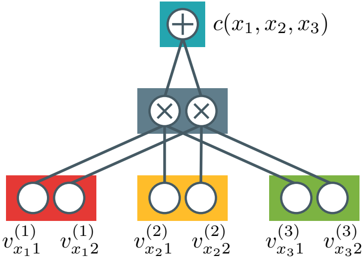
$$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \odot \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}) \qquad \text{(Hadamard)}$$

$$\boldsymbol{\ell}(\mathbf{x}) = \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}) \otimes \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}) \qquad \text{(Kronecker)}$$

A linear projection of two layers is a layer

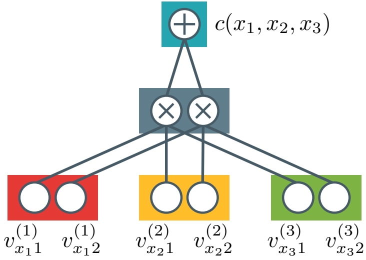
$$\boldsymbol{\ell}(\mathbf{x}) = \mathbf{W}\mathrm{concat}(\boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x}), \boldsymbol{\ell}_{\mathsf{ii}}(\mathbf{x}))$$

The layer-wise circuit definition

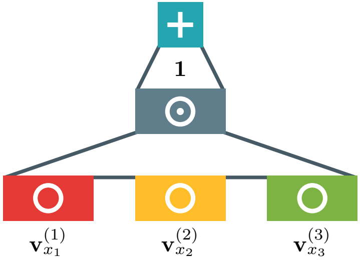


CP circuit

The layer-wise circuit definition

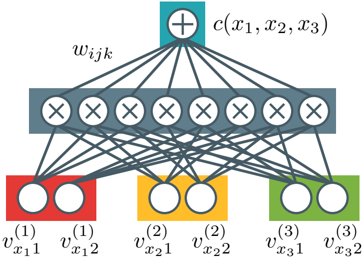


CP circuit



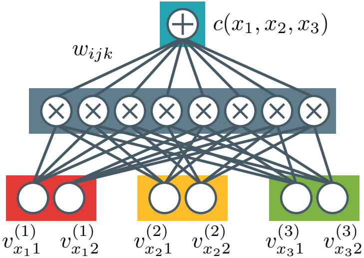
CP circuit

The layer-wise circuit definition

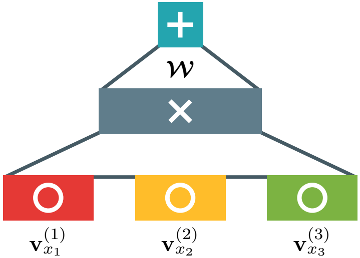


Tucker circuit

The layer-wise circuit definition



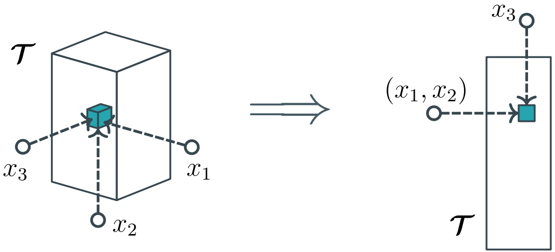
Tucker circuit



Tucker circuit

going deeper: hierarchical Tucker

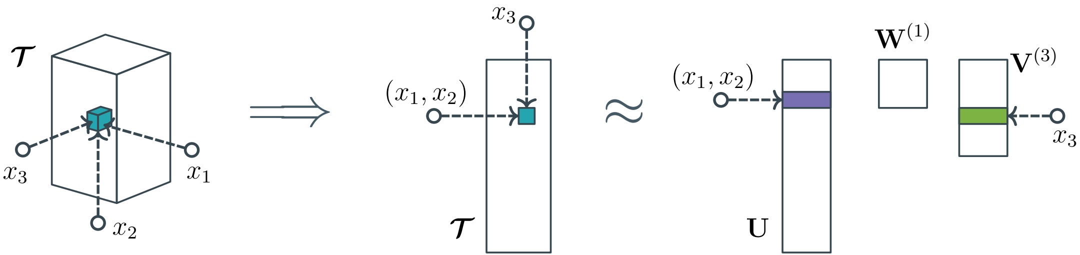
level-one factorization



Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

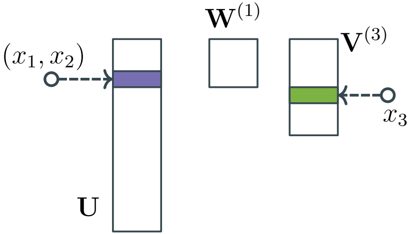
level-one factorization



Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

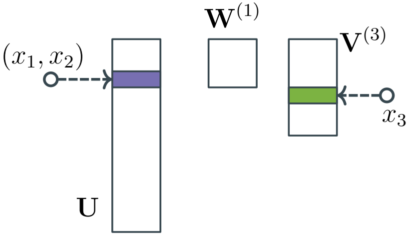
level-one factorization as a circuit

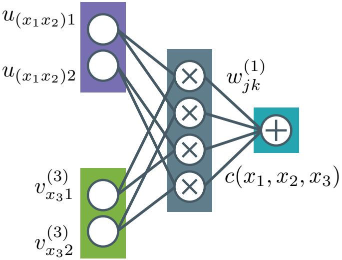


Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

level-one factorization as a circuit

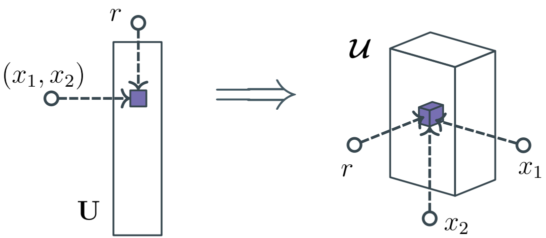




Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

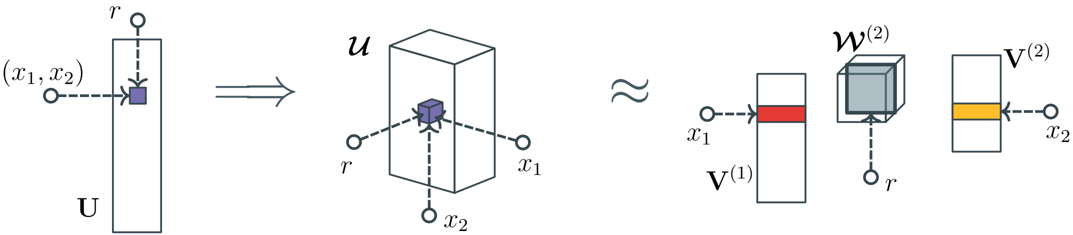
level-two factorization



Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

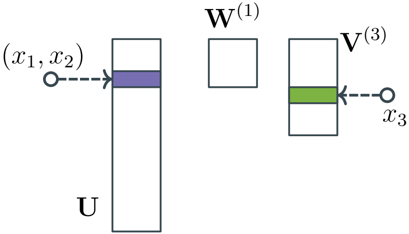
level-two factorization

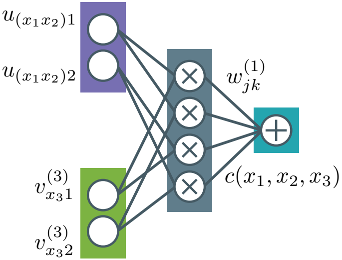


Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

nested factorizations are deep circuits

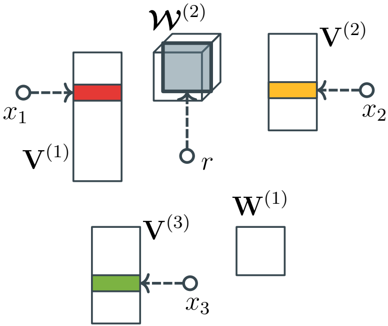


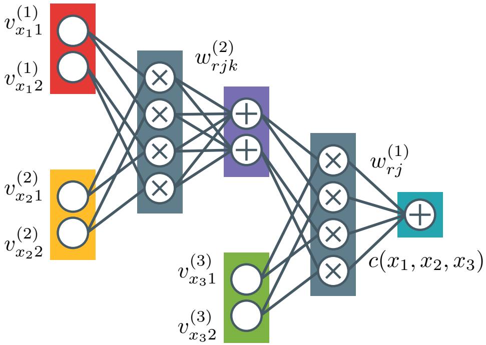


Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

going deeper: hierarchical Tucker

nested factorizations are deep circuits

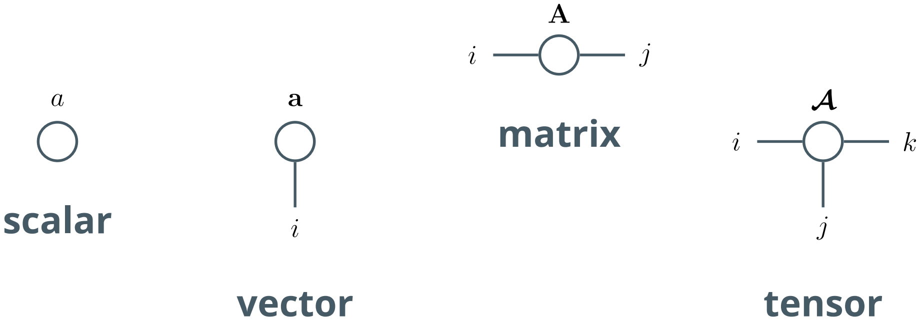




Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

Tensor networks

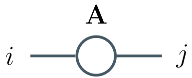
the Penrose graphical notation



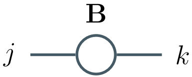
Biamonte and Bergholm, “Tensor Networks in a Nutshell”, 2017

Tensor networks

matrix factorization & contraction



$\mathbf{A} \in \mathbb{R}^{N \times R}$

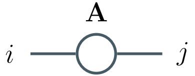


$\mathbf{B} \in \mathbb{R}^{R \times M}$

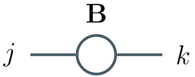
Biamonte and Bergholm, "Tensor Networks in a Nutshell", 2017
Orús, "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States", 2013

Tensor networks

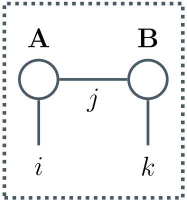
matrix factorization & contraction



$\mathbf{A} \in \mathbb{R}^{N \times R}$



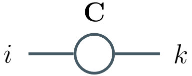
$\mathbf{B} \in \mathbb{R}^{R \times M}$



$\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{N \times M}$

$$c_{ik} = \sum_{j=1}^{R} a_{ij}\, b_{jk}$$

=



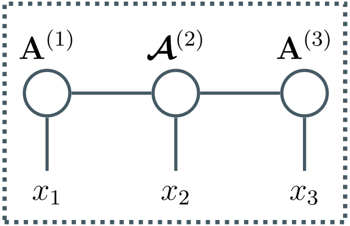
$\mathbf{C} \in \mathbb{R}^{N \times M}$

Biamonte and Bergholm, "Tensor Networks in a Nutshell", 2017

Orús, "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States", 2013

Tensor trains are circuits

also called matrix-product states

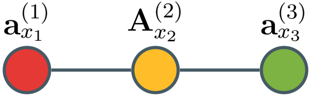


$\boldsymbol{\mathcal{T}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$

$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1} \, a^{(2)}_{r_1 x_2 r_2} \, a^{(3)}_{r_2 x_3}$$

Tensor trains are circuits

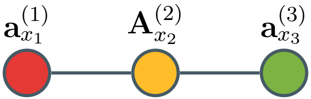
also called matrix-product states



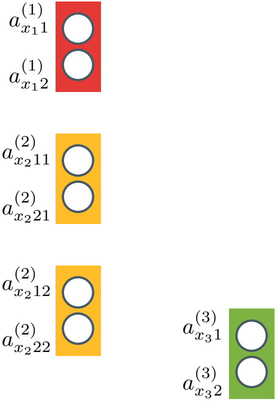
$$t_{x_1x_2x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1r_1} a^{(2)}_{r_1x_2r_2} a^{(3)}_{r_2x_3}$$

Tensor trains are circuits

also called matrix-product states

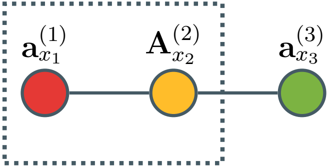


$$t_{x_1x_2x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1r_1} \, a^{(2)}_{r_1x_2r_2} \, a^{(3)}_{r_2x_3}$$

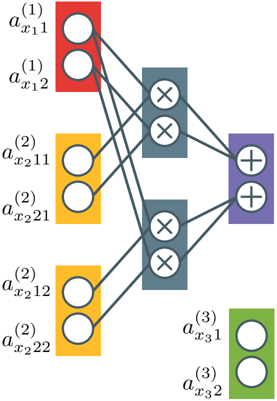


Tensor trains are circuits

also called matrix-product states

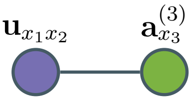


$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1} \, a^{(2)}_{r_1 x_2 r_2} \, a^{(3)}_{r_2 x_3}$$

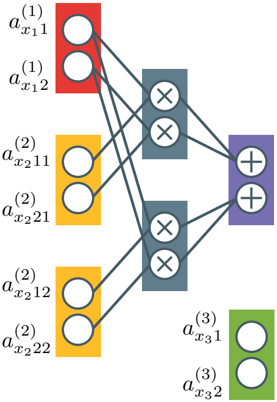


Tensor trains are circuits

also called matrix-product states

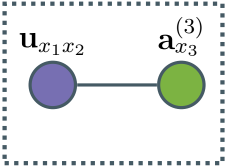


$$t_{x_1x_2x_3} = \sum_{r_2=1}^{R} u_{x_1x_2r_2} \; a^{(3)}_{r_2x_3}$$

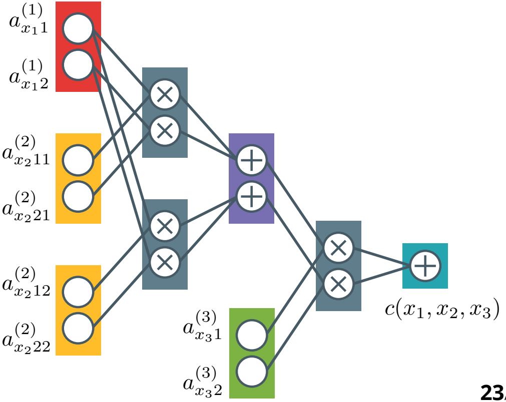


Tensor trains are circuits

also called matrix-product states



$$t_{x_1 x_2 x_3} = \sum_{r_2=1}^{R} u_{x_1 x_2 r_2} \, a^{(3)}_{r_2 x_3}$$

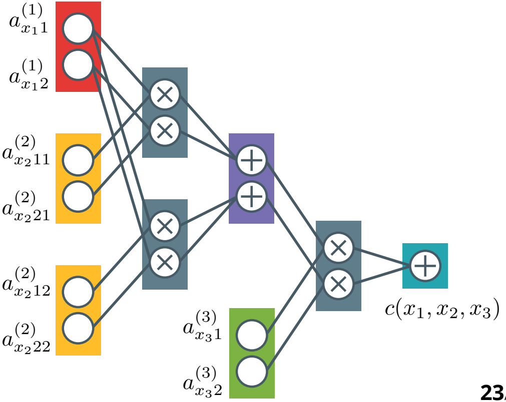


Tensor trains are circuits

also called matrix-product states

$t_{x_1 x_2 x_3}$

$t_{x_1 x_2 x_3}$



Many tensor factorizations are circuits

CP

RESCAL

Tucker

Hierarchical Tucker

Tensor train

Matrix-product state

Hierarchical Tucker

Tree tensor network

ComplEx

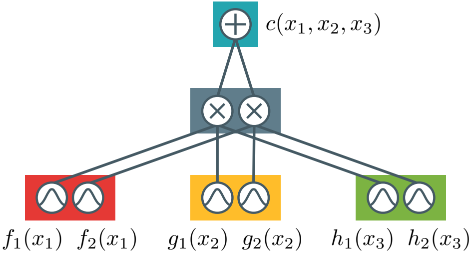
⋮

"What do we gain from circuits?"

More input functions with circuits

Input unit functions $f(x)$ compute:

— an entry of matrix (or tensor):
$f(x) = v_{xr}$ (**embedding layer**)



More input functions with circuits

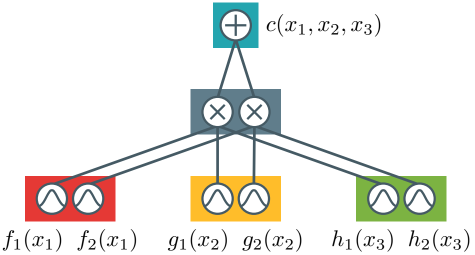
Input unit functions $f(x)$ compute:

— an entry of matrix (or tensor):

$f(x) = v_{xr}$ (**embedding layer**)

— probability mass functions:

$f(x) = \mathrm{Binomial}(x; n, p)$ (**more compact!**)



More input functions with circuits

Input unit functions $f(x)$ compute:

— an entry of matrix (or tensor):

$$f(x) = v_{xr} \quad \textbf{(embedding layer)}$$

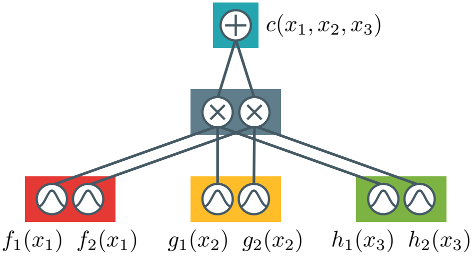
— probability mass functions:

$$f(x) = \text{Binomial}(x; n, p) \quad \textbf{(more compact!)}$$

— continuous functions:

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$
$$f(x) = \text{Normal}(x; \mu, \sigma^2)$$

$\Rightarrow$ **infinite-dimensional tensors (or functions)**



Townsend and Trefethen, "Continuous analogues of matrix factorizations", 2015
Novikov, Panov, and Oseledets, "Tensor-train density estimation", 2021

Probabilistic circuits (PCs)

PC == a circuit c encoding a non-negative function

$$\forall \mathbf{x} \in \text{dom}(\mathbf{X}): \; c(\mathbf{x}) = c(x_1, \ldots, x_n) \geq 0$$

Probabilistic circuits (PCs)

PC == a circuit c encoding a non-negative function
$\forall \mathbf{x} \in \text{dom}(\mathbf{X}) : \; c(\mathbf{x}) = c(x_1, \ldots, x_n) \geq 0$

$$p(\mathbf{x}) = \frac{1}{Z}\, c(\mathbf{x}),$$

where $\quad Z = \sum_{\mathbf{x}} c(\mathbf{x})$ (PMF) or $\quad Z = \int c(\mathbf{x})\, \mathrm{d}\mathbf{x}$ (PDF)

Probabilistic circuits (PCs)

PC == a circuit c encoding a non-negative function

$\forall \mathbf{x} \in \text{dom}(\mathbf{X}):\ \ c(\mathbf{x}) = c(x_1, \ldots, x_n) \geq 0$

$$p(\mathbf{x}) = \frac{1}{Z}\, c(\mathbf{x}),$$

where $\quad Z = \sum_{\mathbf{x}} c(\mathbf{x})$ (PMF) $\quad$ or $\quad Z = \int c(\mathbf{x})\, \mathrm{d}\mathbf{x}$ (PDF)

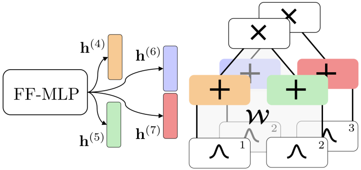
Non-negative sum parameters $\wedge$ non-negative input functions

$\implies$ a circuit is a **PC**

Cichocki and Phan, "Fast Local Algorithms for Large Scale Nonnegative Matrix and Tensor Factorizations", 2009

How to parameterize circuits?

(i.e., the weights of sums and input functions)

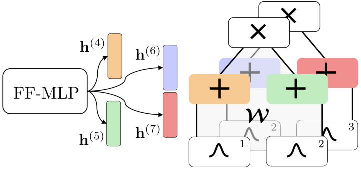


Functions: neural networks

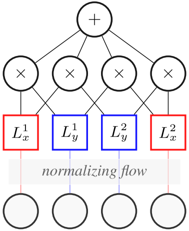
Gala et al., "Scaling Continuous Latent Variable Models as Probabilistic Integral Circuits", 2024
Shao et al., "Conditional sum-product networks: Imposing structure on deep probabilistic architectures", 2020
Sidheekh, Kersting, and Natarajan, "Probabilistic Flow Circuits: Towards Unified Deep Models for Tractable Probabilistic Inference", 2023

How to parameterize circuits?

(i.e., the weights of sums and input functions)



Functions: neural networks



Deep generative models

Gala et al., "Scaling Continuous Latent Variable Models as Probabilistic Integral Circuits", 2024
Shao et al., "Conditional sum-product networks: Imposing structure on deep probabilistic architectures", 2020
Sidheekh, Kersting, and Natarajan, "Probabilistic Flow Circuits: Towards Unified Deep Models for Tractable Probabilistic Inference", 2023

cirkit

learning & reasoning with circuits in pytorch

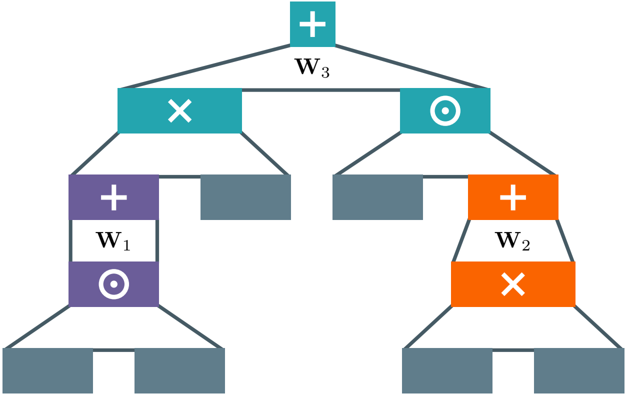
`https://github.com/april-tools/cirkit`

```python
from cirkit.symbolic.layers import (
    EmbeddingLayer, SumLayer, KroneckerLayer,
    Scope
)

# Tensor shape and rank
shape = (3, 1280, 720)
rank = 42

# Construct the Tucker factorization layers
v1 = EmbeddingLayer(Scope([0]), rank, num_states=shape[0])
v2 = EmbeddingLayer(Scope([1]), rank, num_states=shape[1])
v3 = EmbeddingLayer(Scope([2]), rank, num_states=shape[2])
kron = KroneckerLayer(rank, arity=3)
tucker = SumLayer(rank ** 3, num_output_units=1)
```

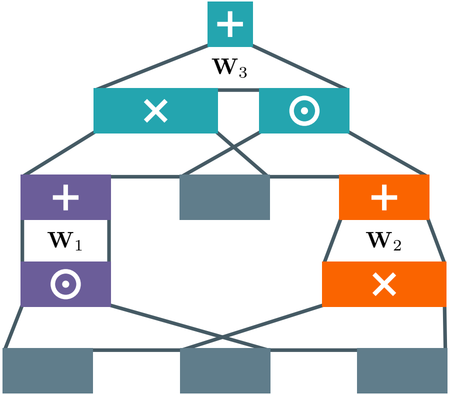
```python

# Construct the Tucker circuit
circuit = Circuit(
    layers=[v1, v2, v3, kron, tucker],
    in_layers={   # The layers input connections
        kron: [v1, v2, v3],
        tucker: [kron]
    },
    outputs=[tucker]
)
```

```python
# Compile the circuit to PyTorch code
from cirkit.pipeline import compile
pth_circuit = compile(circuit)

print(pth_circuit)  # Tucker factorization
# TorchCircuit(
#   (0): TorchEmbeddingLayer(...)
#   (1): TorchEmbeddingLayer(...)
#   (2): TorchEmbeddingLayer(...)
#   (3): TorchKroneckerLayer(...)
#   (4): TorchSumLayer(...)
#)

# Compute one entry of the encoded tensor
x = torch.tensor([[1, 500, 300]])
t_x = pth_circuit(x)
```

```python
from cirkit.templates import tensor_factorizations

shape = (3, 1280, 720)

# CP factorization
circuit = tensor_factorizations.cp(shape, rank=42)

# Tucker factorization
circuit = tensor_factorizations.tucker(shape, rank=42)

# Tensor-train / matrix-product state
circuit = tensor_factorizations.tensor_train(shape, rank=42)
```



Stack layers to build a deep factorization!



Save computation by sharing sub-factorizations!

```python
from cirkit.symbolic.layers import (
    EmbeddingLayer, SumLayer, KroneckerLayer,
    HadamardLayer, Scope)

# Tensor shape and ranks
shape = (17, 3, 1280, 720)
rank1, rank2 = 2, 4

# Construct the layers
v1 = EmbeddingLayer(Scope([0]), rank1, num_states=shape[0])
v2 = EmbeddingLayer(Scope([1]), rank1, num_states=shape[1])
v3 = EmbeddingLayer(Scope([2]), rank1, num_states=shape[2])
v4 = EmbeddingLayer(Scope([3]), rank1, num_states=shape[3])
kron1 = KroneckerLayer(rank1, arity=2)
kron2 = KroneckerLayer(rank1, arity=2)
hada1 = HadamardLayer(rank1, arity=2)
```

```python
hada2 = HadamardLayer(rank1, arity=2)
sum1 = SumLayer(rank1, num_output_units=rank1)
sum2 = SumLayer(rank1 ** 2, num_output_units=rank1)
sum3 = SumLayer(rank1 + rank2, num_output_units=1, arity=2)

# Construct the "Fankenstein" circuit
circuit = Circuit(
    layers=[v1, v2, v3, v4, kron1, kron2, ...],
    in_layers={   # The layers input connections
        hada1: [v1, v2],
        kron1: [v2, v3],
        sum1: [hada1],
        sum2: [kron1],
        kron2: [sum1, v4],
        ...
```

Takeaways

1 circuits ***unify*** many (deep) tensor factorizations

Takeaways

1 **circuits *unify* many (deep) tensor factorizations**

2 **A *Lego block* approach to tensor factorizations**
(different parameterizations)

Takeaways

1 **circuits *unify* many (deep) tensor factorizations**

2 **A *Lego block* approach to tensor factorizations**
(different parameterizations)

3 **build *new* tensor factorizations by connecting layers**
*(easy to do within the **cirkit** library)*

Takeaways

Questions?

1 **circuits *unify* many (deep) tensor factorizations**

2 **A *Lego block* approach to tensor factorizations**
(different parameterizations)

3 **build *new* tensor factorizations by connecting layers**
*(easy to do within the **cirkit** library)*

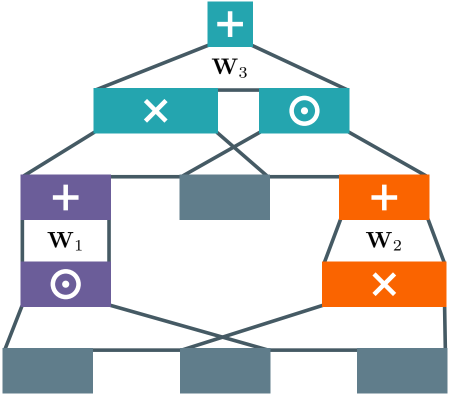
outline

1 connecting *tensor factorizations* and *circuits*

2 a *unifying pipeline* to build factorizations & circuits

tl;dr

"Understand when and how we can build a deep circuit that is a deep factorization"



...but do we always get a tensor factorization?

Multilinear forms

tensor factorizations are typically multilinear

linear in each univariate basis/input function

$$\sum_i \alpha_i \prod_{j=1}^{d} f_{i,j}(x_j)$$

$$\begin{cases} t_{x_1 \cdots x_n} = \sum_{r=1}^{R} \prod_{j=1}^{d} v^{(j)}_{x_j r} & \text{(CP)} \\ t_{x_1 \cdots x_n} = \sum_{r_1, \cdots, r_d = 1}^{R_1, \cdots, R_d} w_{r_1 \cdots r_d} \prod_{j=1}^{d} v^{(j)}_{x_j r_j} & \text{(Tucker)} \end{cases}$$

Vasilescu and Terzopoulos, "Multilinear Image Analysis for Facial Recognition", 2002
Kolda, Multilinear operators for higher-order decompositions, 2006

“How to enforce multilinearity in deep circuits?”

Structural properties

smoothness

decomposability

compatibility

Structural properties

property A

property B

property C

Structural properties

smoothness

decomposability

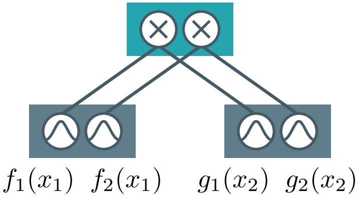
property C

smoothness $\wedge$ decomposability

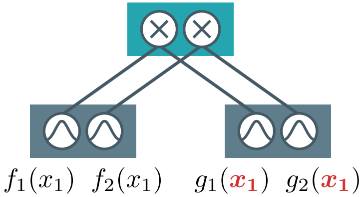
$\Longrightarrow$ multilinearity

Multilinearity in circuits

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



✓ **multilinear**

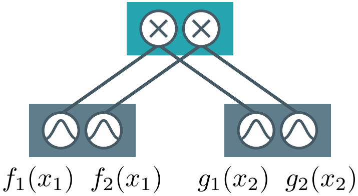


✗ **not multilinear**

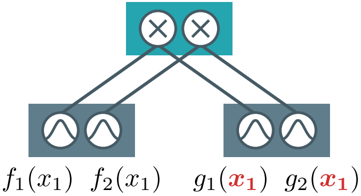
Darwiche and Marquis, "A knowledge compilation map", 2002

Multilinearity in circuits

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



decomposable circuit

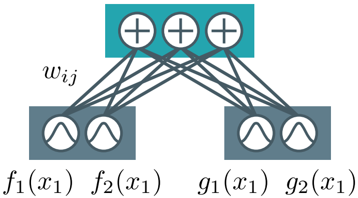


non-decomposable circuit

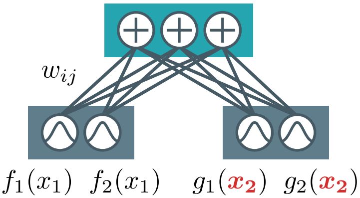
Darwiche and Marquis, "A knowledge compilation map", 2002

Multilinearity in circuits

the inputs of sum units are defined over the same variables



✓ **multilinear**

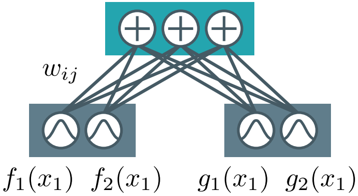


✗ **not multilinear**

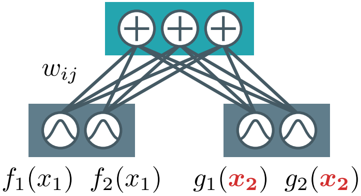
Darwiche and Marquis, "A knowledge compilation map", 2002

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

$\Longrightarrow$ multilinearity

Structural properties

smoothness

decomposability

property C

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

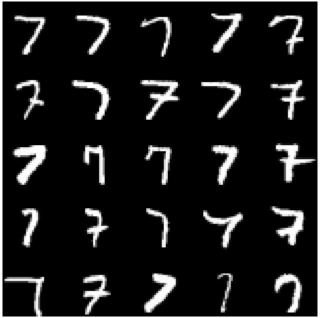
$$p(\mathbf{y}) = \int p(\mathbf{y}, \mathbf{z})\, \mathrm{d}\mathbf{z}, \quad \forall \mathbf{Y} \subseteq \mathbf{X}, \quad \mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$$

$\implies$ tractable partition function

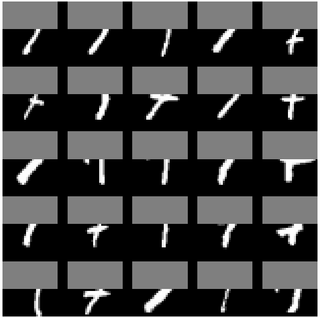
$\implies$ also any conditional is tractable

tractable marginals on PCs

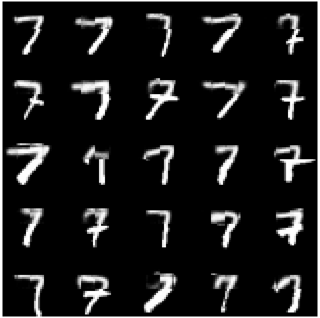
Original



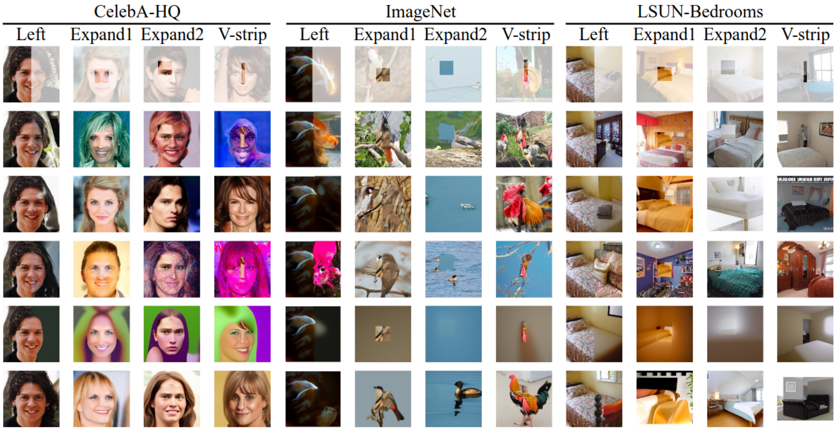
Missing



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", 2024

smooth + *decomposable* circuits = ...

compute arbitrary summations (or integrals)

$\implies$ linear time in circuit size!

E.g., partition function $\sum_{\mathbf{x}} c(\mathbf{x})$

or $\int c(\mathbf{x})\,\mathrm{d}\mathbf{x}$ in the continuous case

Choi, Vergari, and Van den Broeck, [Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling](), 2020

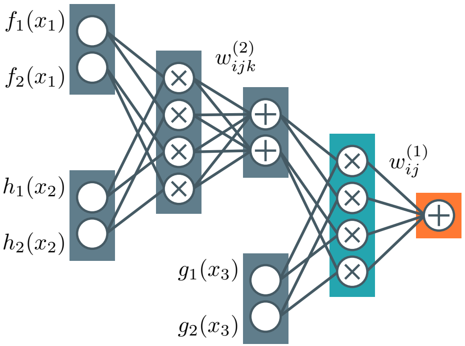
smooth + decomposable circuits = ...

If $c(\mathbf{x}) = \sum_i w_i c_i(\mathbf{x})$ (**smoothness**):

$$\int c(\mathbf{x}) \mathrm{dx} = \int \sum_i w_i c_i(\mathbf{x}) \mathrm{dx}$$

$$= \sum_i w_i \int c_i(\mathbf{x}) \mathrm{dx}$$

$\Rightarrow$ integrals are "pushed down" to the inputs



Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

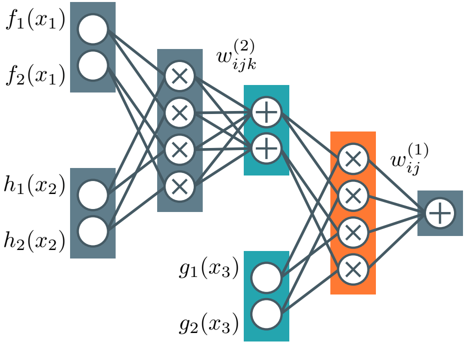
smooth + ***decomposable*** circuits = ...

If $c(\mathbf{x}) = c_1(\mathbf{y})\, c_2(\mathbf{z})$ (**decomposability**):

$$\int c(\mathbf{x})\,\mathrm{dx} = \int\int c_1(\mathbf{y})\, c_2(\mathbf{z})\,\mathrm{dydz}$$

$$= \left(\int c_1(\mathbf{y})\,\mathrm{dy}\right)\left(\int c_2(\mathbf{z})\,\mathrm{dz}\right)$$

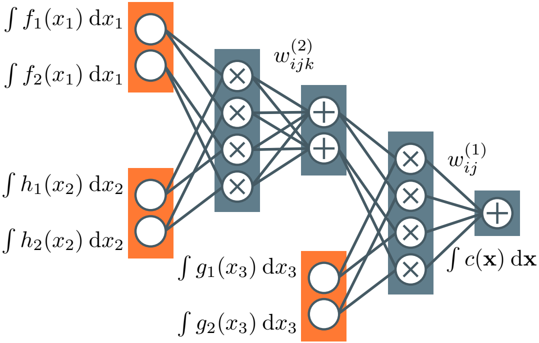
$\Rightarrow$ integrals “decompose” into easier ones



Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

smooth + decomposable circuits = ...

Integrate simple input functions $f(x)$
$\Rightarrow$ Gaussians, polynomials, splines, ...



Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

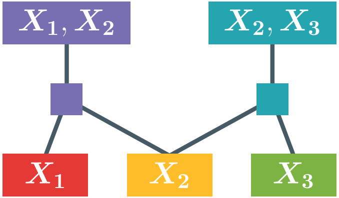
“How to build smooth & decomposable circuits?”

“Can we re-use known tensor factorization methods?”

A zoo of probabilistic circuits...

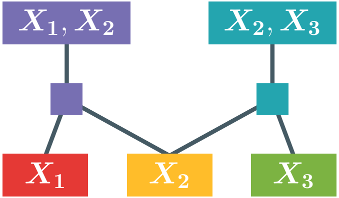
PC ARCHITECTURE
Poon&Domingos (Poon & Domingos, 2011)
RAT-SPN (Peharz et al., 2020c)
EiNet (Peharz et al., 2020a)
HCLT (Liu & Van den Broeck, 2021b)
HMM/MPS$_{\mathbb{R}\geqslant 0}$ (Glasser et al., 2019)
BM (Han et al., 2018)
TTDE (Novikov et al., 2021)
NPC^2 (Loconte et al., 2024)
TTN (Cheng et al., 2019)

Building smooth & decomposable circuits

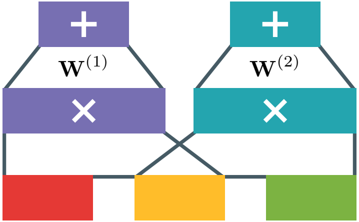


1) choose a template

Building smooth & decomposable circuits



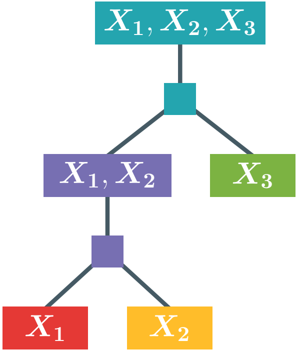
1) choose a template



2) pick a layer to parameterize the chosen template

Region graphs

A bipartite graph to build smooth and decomposable circuits:

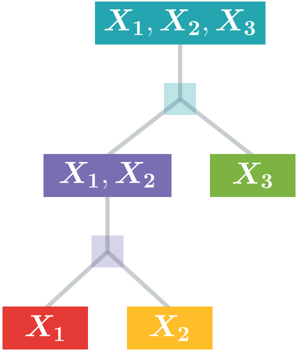


Dennis and Ventura, "Learning the architecture of sum-product networks using clustering on variables", 2012

Region graphs

A bipartite graph to build smooth and decomposable circuits:

Region node:
set of variables (or dimensions)



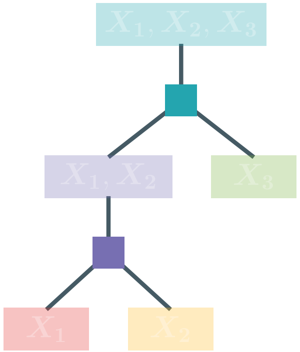
Dennis and Ventura, "Learning the architecture of sum-product networks using clustering on variables", 2012

Region graphs

A bipartite graph to build smooth and decomposable circuits:

Region node:
set of variables (or dimensions)

Partition node:
decomposition of a region



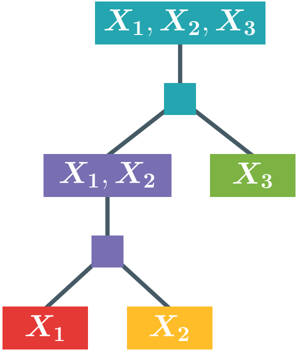
Dennis and Ventura, "Learning the architecture of sum-product networks using clustering on variables", 2012

Region graphs

A bipartite graph to build smooth and decomposable circuits:

Region node:
set of variables (or dimensions)

Partition node:
decomposition of a region



Dennis and Ventura, "Learning the architecture of sum-product networks using clustering on variables", 2012

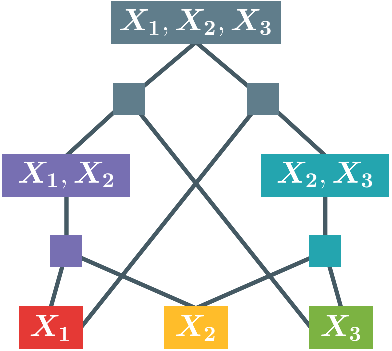
Region graphs

A bipartite graph to build smooth and decomposable circuits:

Region node:
set of variables (or dimensions)

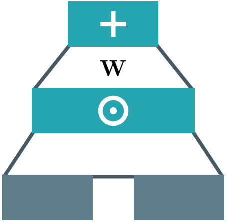
Partition node:
decomposition of a region

$\Longrightarrow$ generalizes mode cluster trees

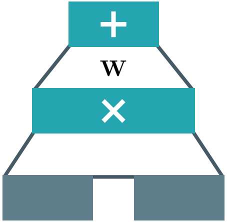


Grasedyck, "Hierarchical singular value decomposition of tensors", 2010

Circuit sum-product layers as factorizations

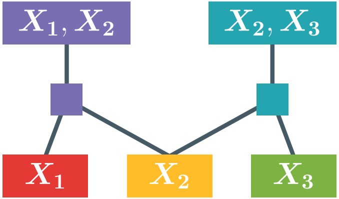


CP layer



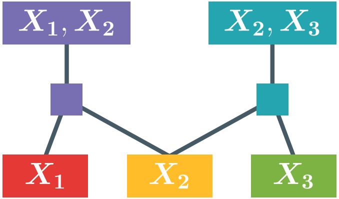
Tucker layer

From region graphs to circuits

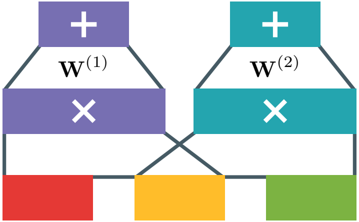


1) choose a region graph

From region graphs to circuits

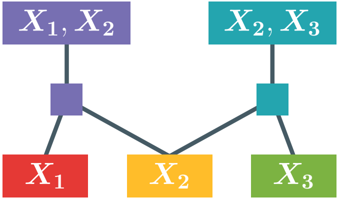


1) choose a region graph

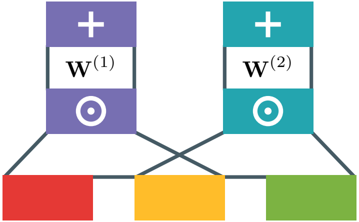


2) pick layer and number of units (e.g., Tucker layer)

From region graphs to circuits



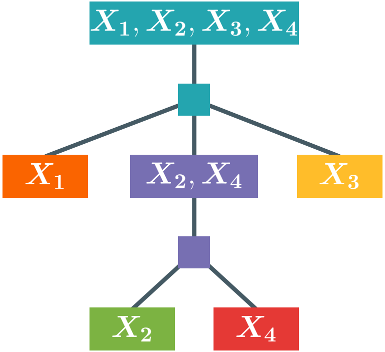
1) choose a region graph



2) pick layer and number of units (e.g., CP layer)

Which region graph?

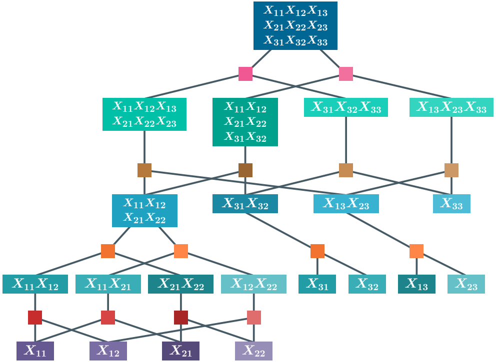
learned or randomized trees



Peharz et al., "Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning", 2020
Liu and Broeck, "Tractable Regularization of Probabilistic Circuits", 2021

Which region graph?

image-tailored graphs

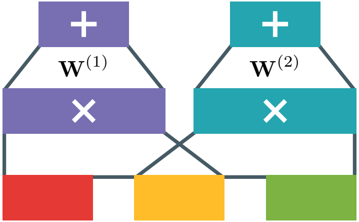


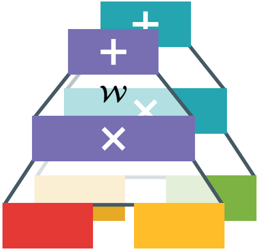
Mari, Vessio, and Vergari, "Unifying and Understanding Overparameterized Circuit Representations via Low-Rank Tensor Decompositions", 2023

folding

to speed-up inference

parallelize layers that can be evaluated independently





Peharz et al., "Einsum networks: Fast and scalable learning of tractable probabilistic circuits", 2020

A unifying circuit construction pipeline

PC Architecture	Region Graph	Sum-Product Layer	Fold
Poon&Domingos (Poon & Domingos, 2011)	PD	$\mathrm{CP}^\top$	✗
RAT-SPN (Peharz et al., 2020c)	RND	Tucker	✗
EiNet (Peharz et al., 2020a)	$\{\mathrm{RND}, \mathrm{PD}\}$	Tucker	✓
HCLT (Liu & Van den Broeck, 2021b)	CL	$\mathrm{CP}^\top$	✓
HMM/$\mathrm{MPS}_{\mathbb{R}\geqslant 0}$ (Glasser et al., 2019)	LT	$\mathrm{CP}^\top$	✗
BM (Han et al., 2018)	LT	$\mathrm{CP}^\top$	✗
TTDE (Novikov et al., 2021)	LT	$\mathrm{CP}^\top$	✗
NPC^2 (Loconte et al., 2024)	$\{\mathrm{LT}, \mathrm{RND}\}$	$\{\mathrm{CP}^\top, \mathrm{Tucker}\}$	✓
TTN (Cheng et al., 2019)	QT-2	Tucker	✗
Mix & Match (our pipeline)	$\left\{\begin{matrix}\mathrm{RND}, \mathrm{PD}, \mathrm{LT}, \\ \mathrm{CL}, \mathrm{QG}, \text{QT-2}, \text{QT-4}\end{matrix}\right\}$ $\times$	$\{\mathrm{Tucker}, \mathrm{CP}, \mathrm{CP}^\top\} \cup \{\mathrm{CP}^{\mathrm{S}}, \mathrm{CP}^{\mathrm{XS}} \mid \textsc{Fold}\ ✓\}$ $\times$	$\{✗, ✓\}$

enrich the pipeline

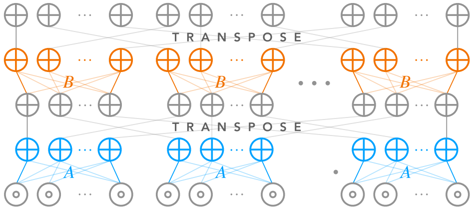
with new layers

Monarch matrix factorization:

$$\mathbf{W}_{\mathcal{M}} = \mathbf{P}_L \mathbf{L} \mathbf{P}_R \mathbf{R}$$

Monarch circuit layer:

$$\boldsymbol{\ell}(\mathbf{x}) = \mathbf{W}_{\mathcal{M}}\, \boldsymbol{\ell}_{\mathsf{i}}(\mathbf{x})$$



More tensor factorizations with new layers!

Zhang et al., "Scaling up Probabilistic Circuits via Monarch Transformations", 2025

```python
# Construct a region graph
from cirkit.templates.region_graph import (
    RandomBinaryTree)  # or QuadGraph, LinearTree, ...
region_graph = RandomBinaryTree(num_variables=10)

# Build the circuit from the region graph
from cirkit.symbolic.layers import EmbeddingLayer
circuit = region_graph.build_circuit(
    sum_product='tucker',            # or 'cp'
    input_factory=EmbeddingLayer    # or GaussianLayer, ...
    num_sum_units=32,
    num_input_units=32)

# Compile the circuit to PyTorch
from circkit.pipeline import compile
pth_circuit = compile(circuit)
```

Takeaways

1 **structural properties for multilinearity**

Takeaways

1 **structural properties for multilinearity**

2 **exact & efficient (*tractable*) summations**

(more properties & operations next!)

Takeaways

1 structural properties for multilinearity

2 exact & efficient (*tractable*) summations
(more properties & operations next!)

3 a *pipeline* to build circuits & tensor factorizations
(different layers and graph structures)

Takeaways

Questions?

1 **structural properties for multilinearity**

2 **exact & efficient (*tractable*) summations**
(more properties & operations next!)

3 **a *pipeline* to build circuits & tensor factorizations**
(different layers and graph structures)

outline

1 connecting *tensor factorizations* and *circuits*

2 a *unifying pipeline* to build factorizations & circuits

3 a *property-driven* approach to inference & reasoning

tl;dr

"Understand when and how we can build a deep factorization that guarantees tractable reasoning"

reasoning about ML models

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

q_2 *"How* ***fair*** *is the prediction with respect protected attribute changes?"*

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

Reasoning about ML models

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

...in the language of probabilities

Reasoning about ML models

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=0)} [f_0(\mathbf{x}_c)] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s=1)} [f_1(\mathbf{x}_c)]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

hard to compute in general!

Reasoning about ML models

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m)\, d\mathbf{X}_m$
(missing values)

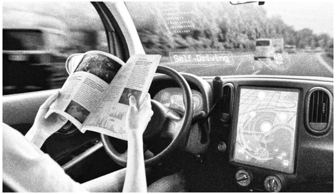
q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$
(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

it is crucial we compute them exactly and in polytime!

Which structural properties

for complex reasoning



$\mathbf{q}_1$ $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$

(missing values)

$\mathbf{q}_2$ $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$

(fairness)

$\mathbf{q}_3$ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

smooth + decomposable

Which structural properties

for complex reasoning

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$

(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

smooth + decomposable

???????

???????

Which properties for expectations?

smoothness

decomposability

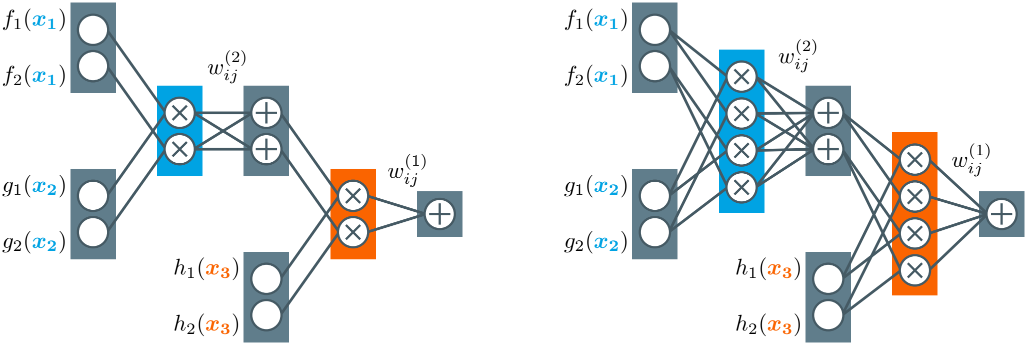
compatibility

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

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

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

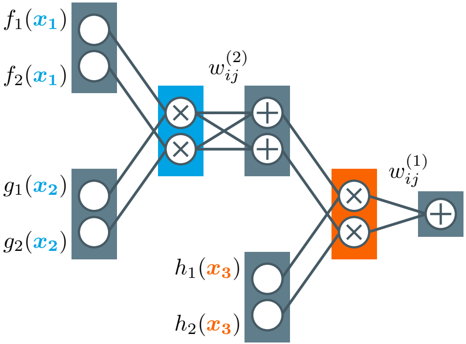
Compatibility

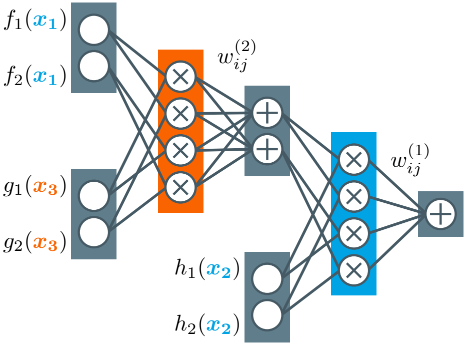


Compatibile circuits

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

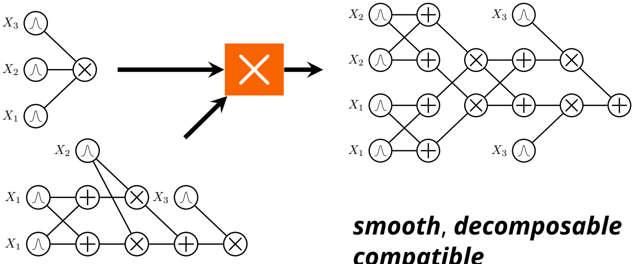
compatibility

$\Downarrow$

smoothness $\wedge$ **decomposability**

compatiblity $\implies$ tractable expectations

Tractable products



smooth**, **decomposable
compatible

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

Vergari et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", 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)
```

Which structural properties

for complex reasoning

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$

(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

smooth + decomposability

compatibility

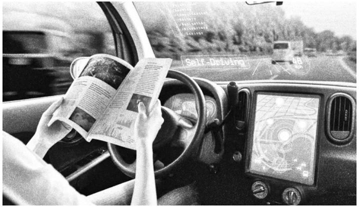
compatibility

What if compatibility does not apply?

$$\mathbb{E}_{\mathbf{e}\sim p_{\text{noise}}(\mathbf{E})}\left[f(\mathbf{x}+\mathbf{e})\right]$$

p_{noise} a circuit

f **not** a circuit (e.g., neural net)



q_3 $\mathbb{E}_{\mathbf{e}\sim p_{\text{noise}}(\mathbf{E})}\left[f(\mathbf{x}+\mathbf{e})\right]$
(adversarial robust.)

How to approximate it by sampling?

wait...!

"How can we sample from a deep factorization or tensor network?"

approximate inference

e.g., via sampling

We can use ***autoregressive inverse transform sampling***:

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

$\Longrightarrow$ *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)?", 2025

approximate inference

e.g., via sampling

We can use ***autoregressive inverse transform sampling***:

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

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

can we do better?

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

approximate inference

e.g., via sampling

We can use ***autoregressive inverse transform sampling***:

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

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

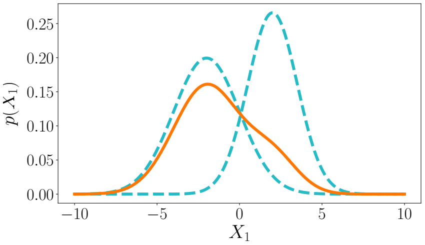
can we do better?

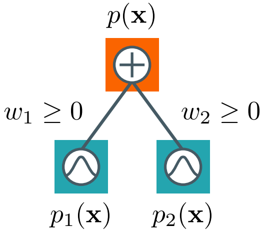
$\Longrightarrow$ *yes, for non-negative factorizations/monotonic PCs*

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

How to sample?

non-negative factorizations as latent-variable models

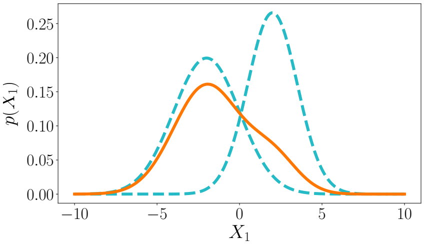


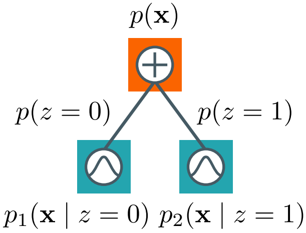


$$p(\mathbf{x}) = w_1 \; p_1(\mathbf{x}) + w_2 \; p_2(\mathbf{x})$$

How to sample?

non-negative factorizations as latent-variable models





$$\boxed{p(\mathbf{x})} = p(z=0)\,\boxed{p_1(\mathbf{x} \mid z=0)} + p(z=1)\,\boxed{p_2(\mathbf{x} \mid z=1)}$$

Structural properties

smoothness

decomposability

compatibility

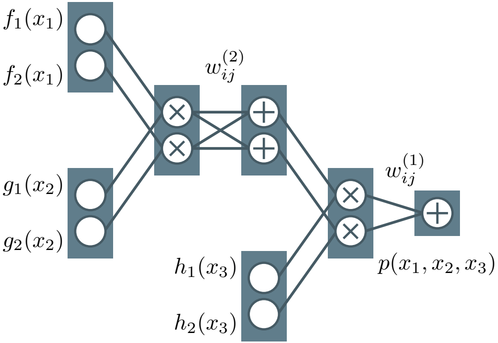
sampling in a single backward pass

draw $\mathbf{x} \sim p(\mathbf{X})$

$\Longrightarrow$ **exact** sampling method

smooth + decomposable circuits = ...

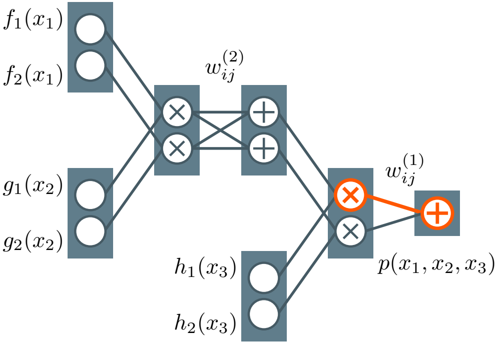
sample variables $x_1, \dots, x_n$ from $p(\mathbf{x})$
$\implies$ linear time in circuit size!



Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

smooth + decomposable circuits = ...

If $p(\mathbf{x}) = \sum_i w_i\, p_i(\mathbf{x})$
(**smoothness**):

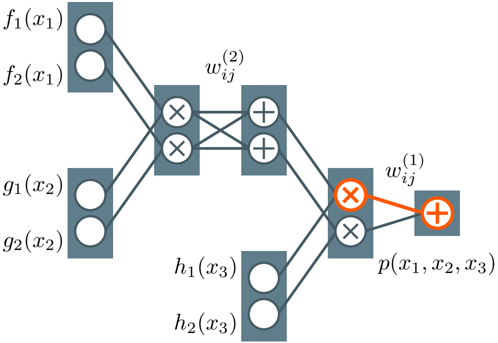


Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

smooth + decomposable circuits = ...

If $p(\mathbf{x}) = \sum_i p(z=i)\, p_i(\mathbf{x} \mid z=i)$ (**smoothness**):

sample $z=i$ from $p(z)$,
then sample $\mathbf{x}$ from $p_i(\mathbf{x} \mid z=i)$

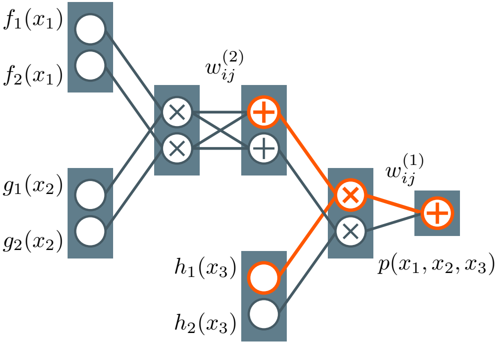


Choi, Vergari, and Van den Broeck, *Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling*, 2020

smooth + *decomposable* circuits = ...

If $p(\mathbf{x}) = p_1(\mathbf{y})\, p_2(\mathbf{z})$ (**decomposability**):

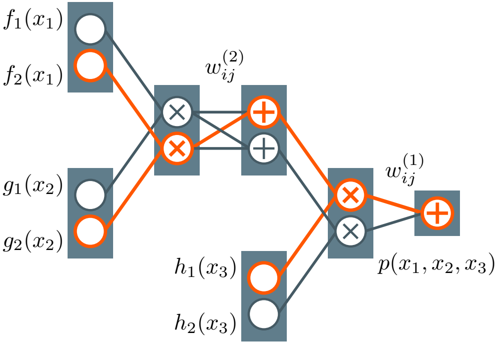
sample $\mathbf{y}$ from p_1 and $\mathbf{z}$ from p_2 (as they are disjoint)



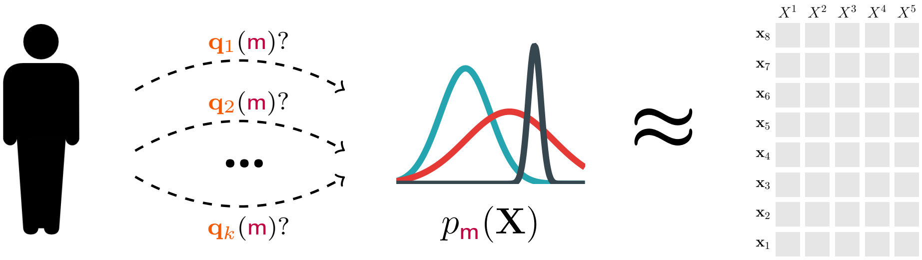
Choi, Vergari, and Van den Broeck, *Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling*, 2020

smooth + decomposable circuits = ...

Sample from simple input distributions:
$\Rightarrow$ easy for Categorical, Gaussian, ...



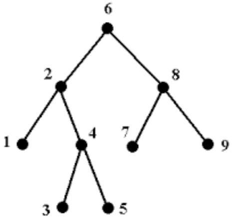
Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020



generative models that can reason probabilistically

...but some events are certain!

When?



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

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

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

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

hierarchical multi-label classification

Giunchiglia and Lukasiewicz, "Coherent Hierarchical Multi-Label Classification Networks", 2020

When?

Ground Truth

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

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

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

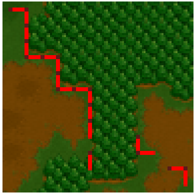
// for a 12×12 grid, 2^{144} states but only 10^{10} valid ones!

nesy structured output prediction (SOP) tasks

Pogančić et al., "Differentiation of Blackbox Combinatorial Solvers", 2020

When?

Ground Truth

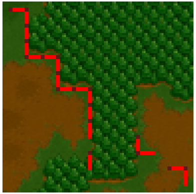


ResNet-18

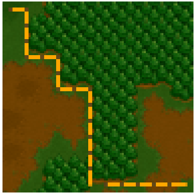
neural nets struggle to satisfy validity constraints!

Constraint losses

Ground Truth



ResNet-18

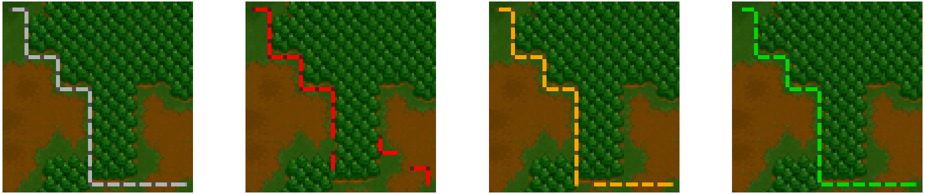


Semantic Loss

...but cannot guarantee consistency at test time!

Xu et al., "A semantic loss function for deep learning with symbolic knowledge", 2018

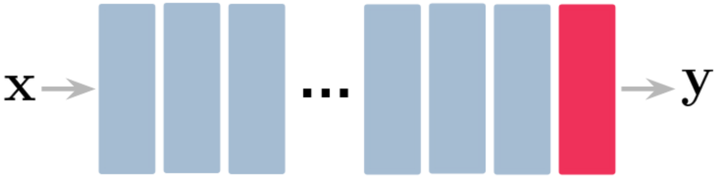
SPL



Ground Truth | ResNet-18 | Semantic Loss | circuits

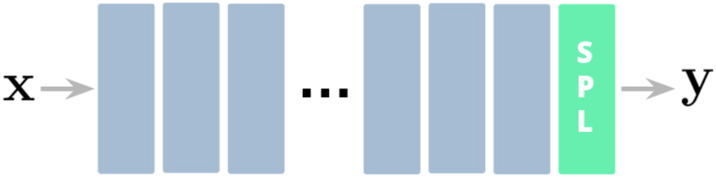
you can predict valid paths 100% of the time!

How?



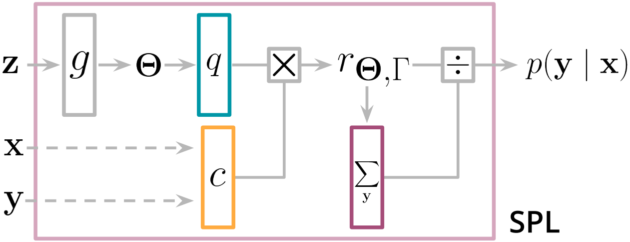
take an unreliable neural network architecture...

How?



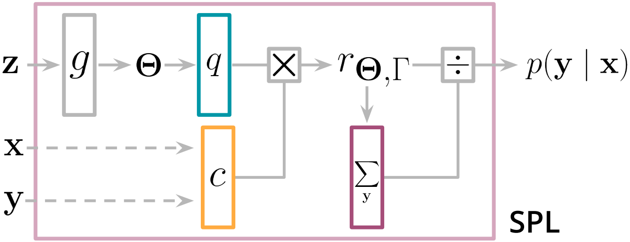
......and replace the last layer with a semantic probabilistic layer (SPL)

SPL



$\mathbf{z}$
g
$\boldsymbol{\Theta}$
q
$r_{\boldsymbol{\Theta},\Gamma}$
$p(\mathbf{y} \mid \mathbf{x})$
$\mathbf{x}$
$\mathbf{y}$
c
$\sum_{\mathbf{y}}$
SPL

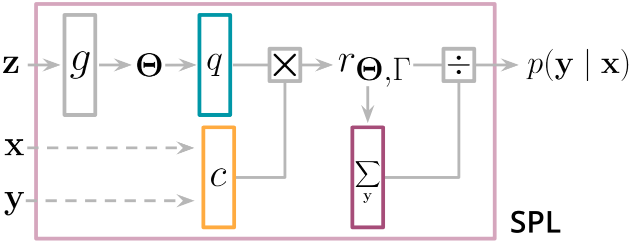
SPL



$$p(\mathbf{y} \mid \mathbf{x}) = \boldsymbol{q}_{\boldsymbol{\Theta}}(\mathbf{y} \mid g(\mathbf{z}))$$

$\boldsymbol{q}_{\boldsymbol{\Theta}}(\mathbf{y} \mid g(\mathbf{z}))$ is an expressive distribution over labels

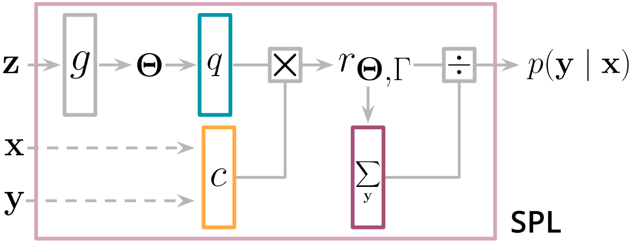
SPL



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

$\boldsymbol{c}_{\mathsf{K}}(\mathbf{x}, \mathbf{y})$ encodes the constraint $\mathbb{1}\{\mathbf{x}, \mathbf{y} \models \mathsf{K}\}$

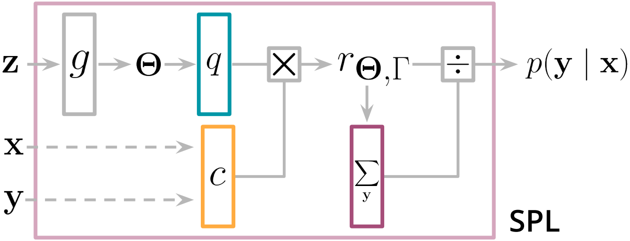
SPL



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

a product of experts : (

SPL



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

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

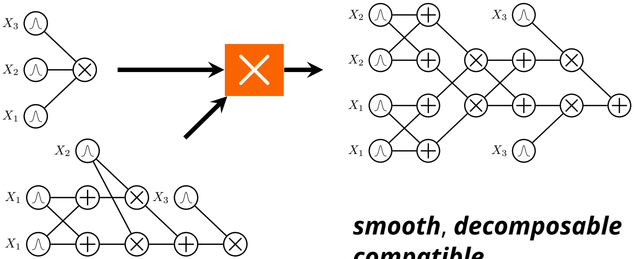
Goal

Can we design q and c
to be deep factorizations
yet yielding a tractable product?

Goal

Can we design q and c
to be deep factorizations
yet yielding a tractable product?

Tractable products

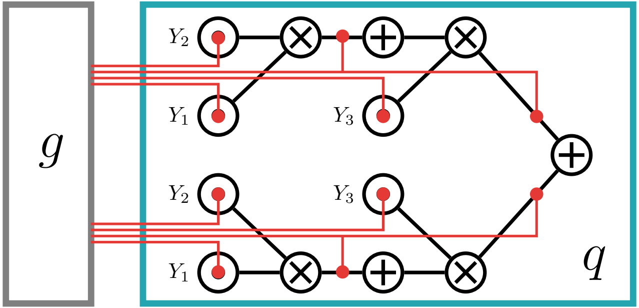


smooth*, *decomposable
compatible

exactly compute $\mathcal{Z}$ in time $O(|\boldsymbol{q}||\boldsymbol{c}|)$

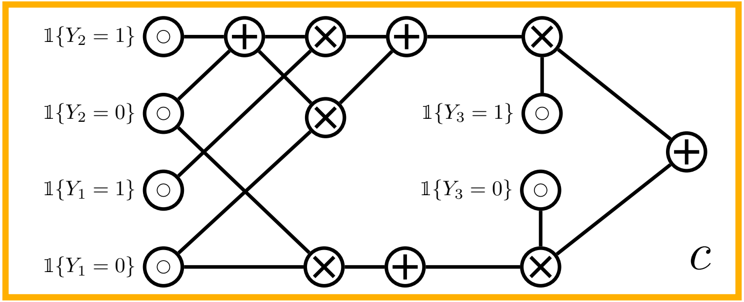
Vergari et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", 2021

SPL



a conditional circuit $q(\mathbf{y}; \Theta = g(\mathbf{z}))$

SPL



and a logical circuit $c(\mathrm{y}, \mathrm{x})$ **encoding** K

knowledge compilation

(as a Boolean tensor factorization)

$$\mathsf{K}: \; (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \, (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0,1\}^3$

$$k_{y_1 y_2 y_3} = \mathbf{1}\{y_1 y_2 y_3 \models \mathsf{K}\}$$

$\mathbb{1}\{Y_1 = 0\} \; \odot$ $\quad \mathbb{1}\{Y_1 = 1\} \; \odot$ $\quad \mathbb{1}\{Y_2 = 0\} \; \odot$

$\mathbb{1}\{Y_2 = 1\} \; \odot$ $\quad \mathbb{1}\{Y_3 = 0\} \; \odot$ $\quad \mathbb{1}\{Y_3 = 1\} \; \odot$

Pipatsrisawat and Darwiche, "New Compilation Languages Based on Structured Decomposability.", 2008

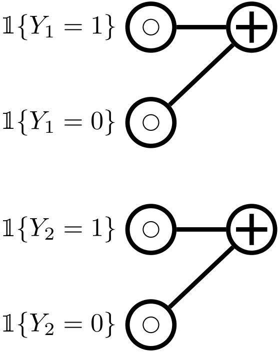
knowledge compilation

(as a Boolean tensor factorization)

$$\mathsf{K}: \ (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \ (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0,1\}^3$

$$k_{y_1 y_2 y_3} = \mathbf{1}\{y_1 y_2 y_3 \models \mathsf{K}\}$$



Pipatsrisawat and Darwiche, "New Compilation Languages Based on Structured Decomposability.", 2008

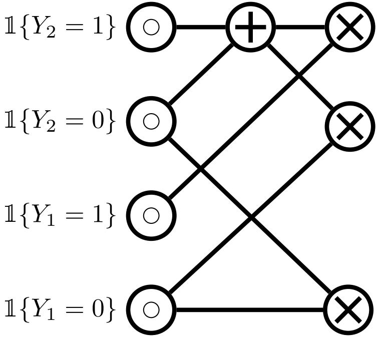
knowledge compilation

(as a Boolean tensor factorization)

$$\mathsf{K}: \ (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \ (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0,1\}^3$

$$k_{y_1 y_2 y_3} = \mathbb{1}\{y_1 y_2 y_3 \models \mathsf{K}\}$$



Pipatsrisawat and Darwiche, "New Compilation Languages Based on Structured Decomposability.", 2008

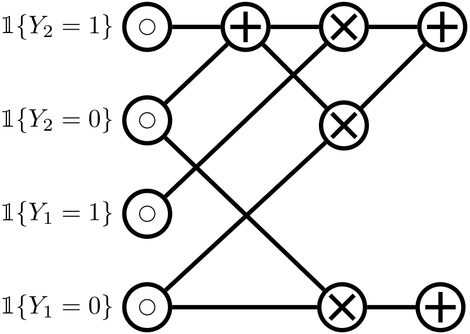
knowledge compilation

(as a Boolean tensor factorization)

$$\mathsf{K}: \ (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \ (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0,1\}^3$

$$k_{y_1 y_2 y_3} = \mathbb{1}\{y_1 y_2 y_3 \models \mathsf{K}\}$$



Pipatsrisawat and Darwiche, "New Compilation Languages Based on Structured Decomposability.", 2008

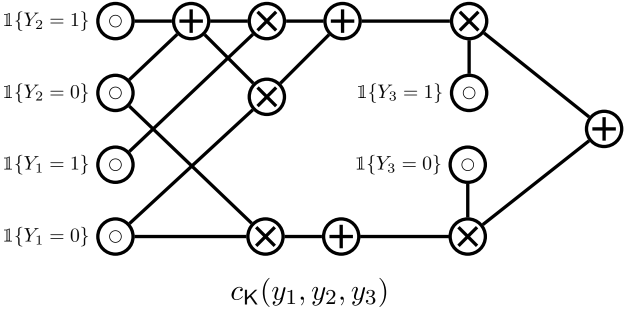
knowledge compilation

(as a Boolean tensor factorization)

$$\mathsf{K}: \ (Y_1 = 1 \implies Y_3 = 1)$$
$$\land \, (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0,1\}^3$

$$k_{y_1 y_2 y_3} = \mathbf{1}\{y_1 y_2 y_3 \models \mathsf{K}\}$$



Pipatsrisawat and Darwiche, "New Compilation Languages Based on Structured Decomposability.", 2008

Tensor Decomposition Meets Knowledge Compilation: A Study Comparing Tensor Trains with OBDDs

Ryoma Onaka, Kengo Nakamura, Masaaki Nishino, Norihito Yasuda

NTT Communication Science Laboratories, NTT Corporation, Kyoto, Japan
{ryoma.onaka,kengo.nakamura,masaaki.nishino,norihito.yasuda}@ntt.com

more tensor factorizations for NeSy at AAAI 2025

NeSy AI recipe

with circuits (and tensor factorizations)

$$\mathsf{K}: (Y_1 = 1 \implies Y_3 = 1)$$
$$\land \quad (Y_2 = 1 \implies Y_3 = 1)$$

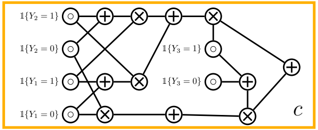
1) Take a
logical constraint

Ahmed et al., "Semantic probabilistic layers for neuro-symbolic learning", 2022

NeSy AI recipe

with circuits (and tensor factorizations)

$$\mathsf{K}: (Y_1 = 1 \implies Y_3 = 1)$$
$$\wedge \quad (Y_2 = 1 \implies Y_3 = 1)$$



1) Take a logical constraint

2) Compile it into a Boolean circuit

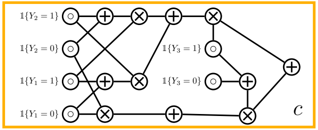
Ahmed et al., "Semantic probabilistic layers for neuro-symbolic learning", 2022

NeSy AI recipe

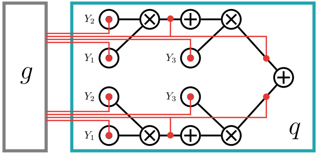
with circuits (and tensor factorizations)

$$
\begin{aligned}
\mathsf{K}: &\ (Y_1 = 1 \implies Y_3 = 1) \\
\wedge &\ (Y_2 = 1 \implies Y_3 = 1)
\end{aligned}
$$

1) Take a logical constraint



2) Compile it into a Boolean circuit

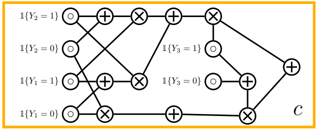


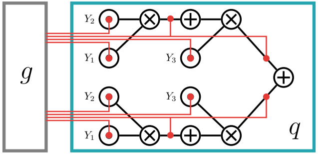
3) Multiply it by a circuit distribution

NeSy AI recipe

with circuits (and tensor factorizations)

$$
\begin{aligned}
\mathsf{K}: \ & (Y_1 = 1 \implies Y_3 = 1) \\
\land \ & (Y_2 = 1 \implies Y_3 = 1)
\end{aligned}
$$





1) Take a logical constraint

2) Compile it into a Boolean circuit

3) Multiply it by a circuit distribution

4) train end-to-end by sgd!

Ahmed et al., "Semantic probabilistic layers for neuro-symbolic learning", 2022

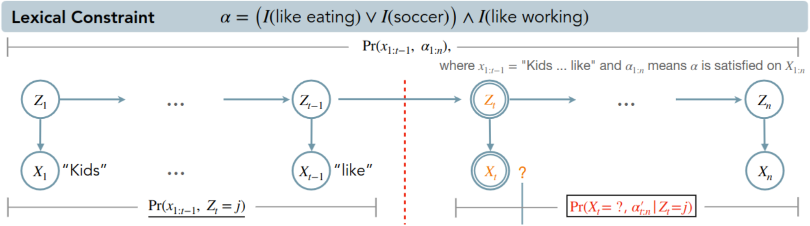
SPLs

(and more circuits)

everywhere

Tractable Control for Autoregressive Language Generation

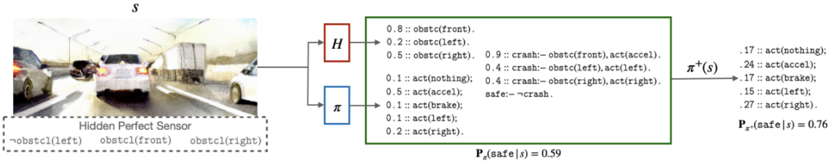
Honghua Zhang [*1] Meihua Dang [*1] Nanyun Peng [1] Guy Van den Broeck [1]



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

Which structural properties

for complex reasoning

q_1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$

(missing values)

q_2 $\mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 0)} \left[f_0(\mathbf{x}_c) \right] - \mathbb{E}_{\mathbf{x}_c \sim p(\mathbf{X}_c | X_s = 1)} \left[f_1(\mathbf{x}_c) \right]$

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

smooth + decomposability

compatibility

compatibility

wait...!

"Given a reasoning task can we automatically distill a tractable algorithm for it?"

A language for queries

Integral expressions that can be formed by composing these operators

$+$, $\times$, **pow**, **log**, **exp** and $/$

$\Longrightarrow$ *many divergences and information-theoretic queries*

A language for queries

Integral expressions that can be formed by composing these operators

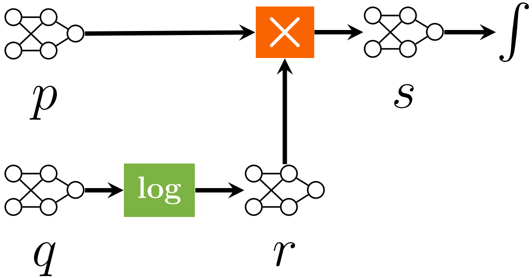
$+$, $\times$, **pow**, **log**, **exp** and $/$

$\Rrightarrow$ *many divergences and information-theoretic queries*

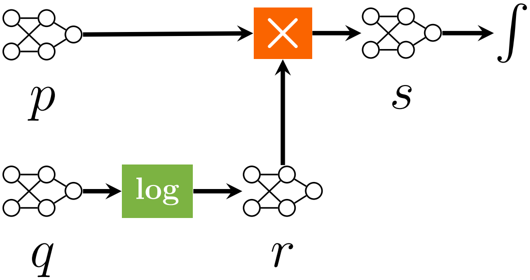
Represented as ***higher-order computational graphs***—pipelines—operating over circuits!

$\Rrightarrow$ *re-using intermediate transformations across queries*

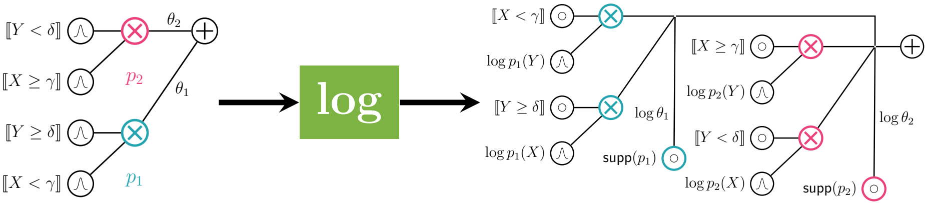
$$\mathbb{XENT}(p \,||\, q) = \int p(\mathbf{x}) \times \log q(\mathbf{x}) \; d\mathbf{X}$$



$$\mathbb{XENT}(p \,||\, q) = \int p(\mathbf{x}) \times \boxed{\log} q(\mathbf{x})\, d\mathbf{X}$$



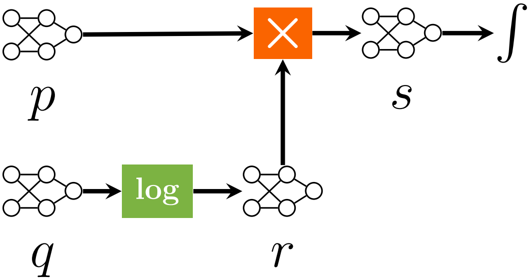
Tractable operators



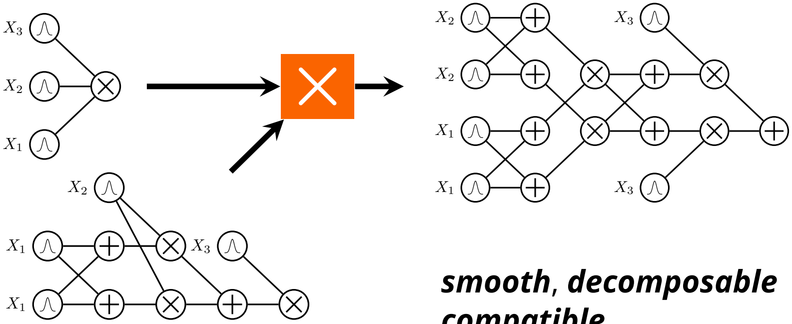
smooth, decomposable deterministic

smooth, decomposable

$$\mathbb{XENT}(p \mid\mid q) = \int p(\mathbf{x}) \times \log q(\mathbf{x})\, d\mathbf{X}$$

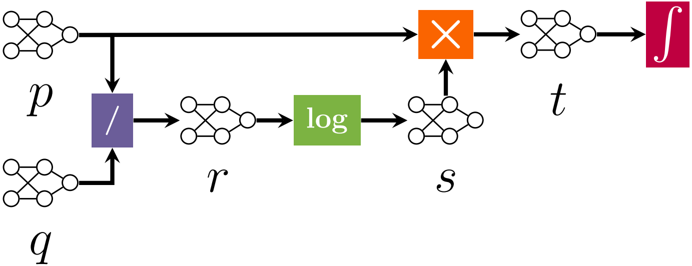


Tractable operators



smooth, decomposable compatible

$$\int p(\mathbf{x}) \times \log \left(p(\mathbf{x}) / q(\mathbf{x}) \right) \, d\mathbf{X}$$



build a LEGO-like query calculus...

	Query	Tract. Conditions	Hardness
CROSS ENTROPY	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\mathbf{X}$	Cmp, q Det	#P-hard w/o Det
SHANNON ENTROPY	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det	coNP-hard w/o Det
RÉNYI ENTROPY	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	SD	#P-hard w/o SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det	#P-hard w/o Det
MUTUAL INFORMATION	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*	coNP-hard w/o SD
KULLBACK-LEIBLER DIV.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x})) d\mathbf{X}$	Cmp, Det	#P-hard w/o Det
RÉNYI'S ALPHA DIV.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det	#P-hard w/o Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det	#P-hard w/o Det
ITAKURA-SAITO DIV.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\,\mathbf{X}$	Cmp, Det	#P-hard w/o Det
CAUCHY-SCHWARZ DIV.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp	#P-hard w/o Cmp
SQUARED LOSS	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\,\mathbf{X}$	Cmp	#P-hard w/o Cmp

*...and **compositionally** derive many more tractable algorithms*

Vergari et al., “A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference”, 2021

Takeaways

1 ***faster sampling*** **routines for non-negative factorizations**

Takeaways

1 *faster sampling* routines for non-negative factorizations

2 integration of *logical constraints* with guarantees

Takeaways

1 *faster sampling* **routines for non-negative factorizations**

2 **integration of** *logical constraints* **with guarantees**

3 *automatically distill* **efficient algorithms for tensor networks via circuit properties**

Takeaways

Questions?

1 *faster sampling* routines for non-negative factorizations

2 integration of *logical constraints* with guarantees

3 *automatically distill* efficient algorithms for tensor networks via circuit properties

outline

1 connecting *tensor factorizations* and *circuits*

2 a *unifying pipeline* to build factorizations & circuits

3 a *property-driven* approach to inference & reasoning

4 ***expressiveness analysis***: **known and new results**

tl;dr

***"Understand when and how
one factorization scheme can be
provably more expressive than others"***

Expressiveness of tensor factorizations

We care about factorization methods that yield **compact** decompositions
(minimise memory footprint & computation)

Expressiveness of tensor factorizations

We care about factorization methods that yield **compact** decompositions

(minimise memory footprint & computation)

*“if rank(s) is **exponential in** d, then it is not useful!”*

$\implies$ storing $\mathcal{T} \in \mathbb{R}^{M\times\cdots\times M}$ requires $\mathcal{O}(M^d)$ memory

Expressiveness of tensor factorizations

We care about factorization methods that yield **compact** decompositions

(minimise memory footprint & computation)

*"if rank(s) is **exponential in** d, then it is not useful!"*

$\Longrightarrow$ storing $\mathcal{T} \in \mathbb{R}^{M\times\cdots\times M}$ requires $\mathcal{O}(M^d)$ memory

One factorization method may require **exponentially smaller rank** than others $\Longrightarrow$ it is more **expressive**

Cohen, Sharir, and Shashua, "On the Expressive Power of Deep Learning: A Tensor Analysis", 2015

wait...!

what about circuits?

Expressiveness of circuits

A rigorous concept in circuit complexity theory

Valiant, "Negation can be exponentially powerful", 1979
Darwiche and Marquis, "A knowledge compilation map", 2002
Martens and Medabalimi, "On the expressive efficiency of sum product networks", 2014

Expressiveness of circuits

A rigorous concept in circuit complexity theory

Expressiveness results of circuits based on the **circuit size**

$\Longrightarrow$ number of edges between units (amount of computation)

Valiant, "Negation can be exponentially powerful", 1979
Darwiche and Marquis, "A knowledge compilation map", 2002
Martens and Medabalimi, "On the expressive efficiency of sum product networks", 2014

Expressiveness of circuits

A rigorous concept in circuit complexity theory

Expressiveness results of circuits based on the **circuit size**

$\Longrightarrow$ number of edges between units (amount of computation)

Different circuit classes have different expressive power

Valiant, "Negation can be exponentially powerful", 1979
Darwiche and Marquis, "A knowledge compilation map", 2002
Martens and Medabalimi, "On the expressive efficiency of sum product networks", 2014

“Circuit complexity theory helps proving stronger results for tensor factorizations”

Published as a conference paper at ICLR 2024

SUBTRACTIVE MIXTURE MODELS VIA SQUARING: REPRESENTATION AND LEARNING

Lorenzo Loconte[1]* **Aleksanteri M. Sladek**[2] **Stefan Mengel**[3]
Martin Trapp[2] **Arno Solin**[2] **Nicolas Gillis**[4] **Antonio Vergari**[1]

[1] School of Informatics, University of Edinburgh, UK
[2] Department of Computer Science, Aalto University, Finland
[3] University of Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France
[4] Department of Mathematics and Operational Research, Université de Mons, Belgium

Accepted at ICLR 2024 as a spotlight

Monotonic probabilistic circuits

Monotonic circuits

$$p(\mathbf{x}) = \frac{1}{Z}\, c(\mathbf{x}), \quad c(\mathbf{x}) \geq 0$$

where parameters and input functions are **positive**

Choi, Vergari, and Van den Broeck, Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling, 2020

Monotonic probabilistic circuits

Monotonic circuits

$$p(\mathbf{x}) = \frac{1}{Z}\, c(\mathbf{x}), \quad c(\mathbf{x}) \geq 0$$

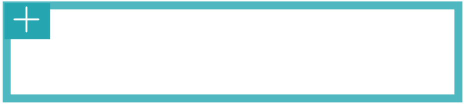
where parameters and input functions are **positive**

(represent non-negative tensor factorizations *[Cichocki and Phan 2009]*)

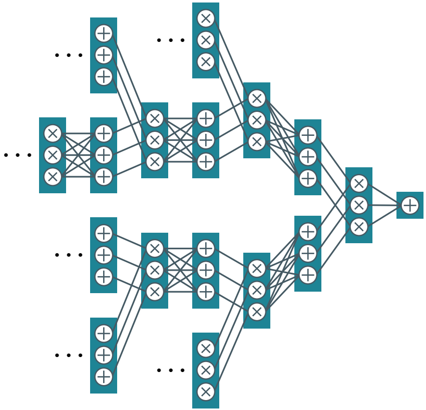
Cichocki and Phan, "Fast Local Algorithms for Large Scale Nonnegative Matrix and Tensor Factorizations", 2009

A limitation of monotonic circuits

$\square$ = set of distributions modeled by **polysize** circuits



- UDISJ



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

Squared circuits

$p(\mathbf{x}) = \frac{1}{Z}\, c^2(\mathbf{x}), \quad c(\mathbf{x}) \in \mathbb{R}$

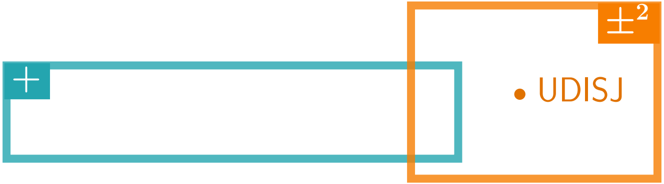
where parameters and input functions can be **negative**

Squared circuits

$$p(\mathbf{x}) = \frac{1}{Z}\, c^2(\mathbf{x}), \quad c(\mathbf{x}) \in \mathbb{R}$$

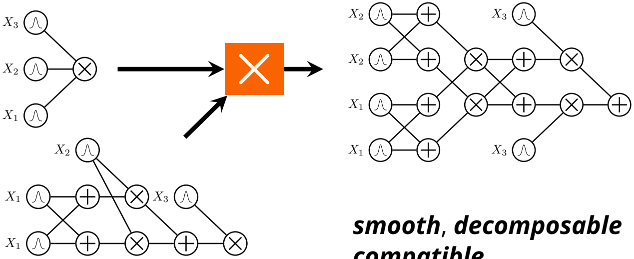
where parameters and input functions can be **negative**

▭ $=$ set of distributions modeled by **polysize** circuits



Tractable product

thanks to circuit compatibility



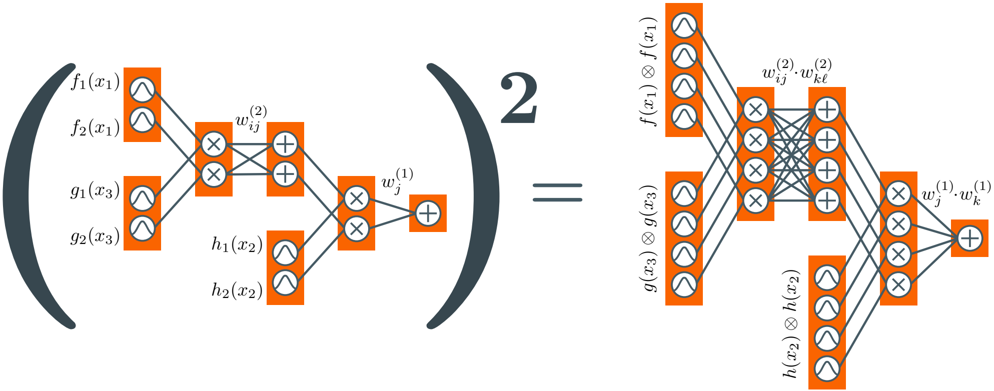
smooth*, *decomposable
compatible

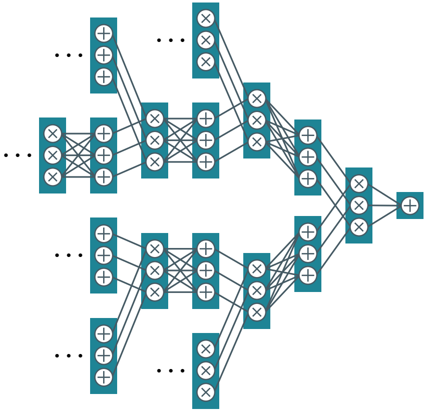
Vergari et al., "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference", 2021

Squared circuits

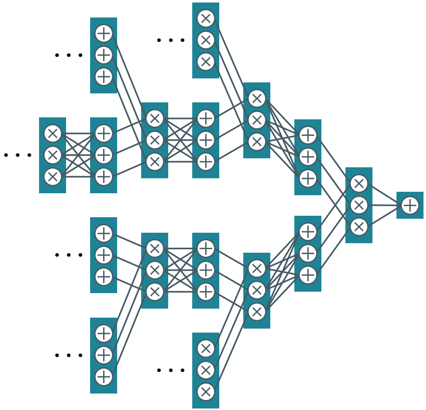
$$p(\mathbf{x}) = \frac{1}{Z}\, c^2(\mathbf{x}), \quad c(\mathbf{x}) \in \mathbb{R}$$

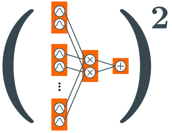
where parameters and input functions can be **negative**



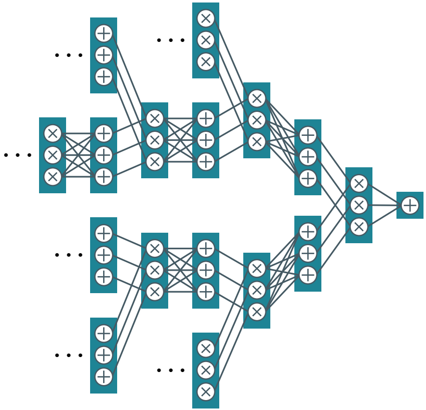


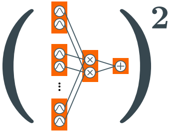
$\exists\, p$ requiring **exponentially large** **monotonic circuits**...





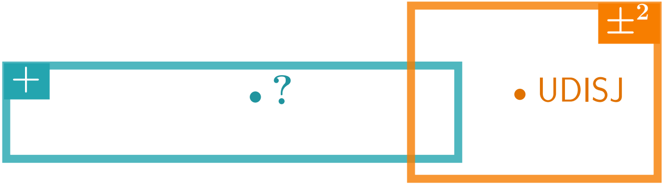
...instead **squared circuits** require **polynomial size**





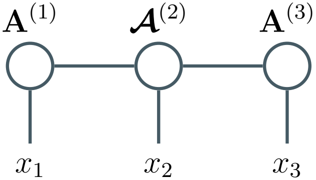
Squared circuits more expressive than **monotonic** ones

Colnet and Mengel, "A Compilation of Succinctness Results for Arithmetic Circuits", 2021



"Can monotonic circuits be more expressive than squared?"

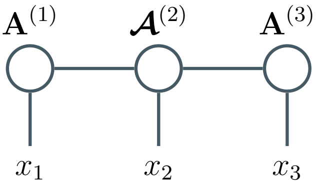
Squared circuits are sparse Born machines



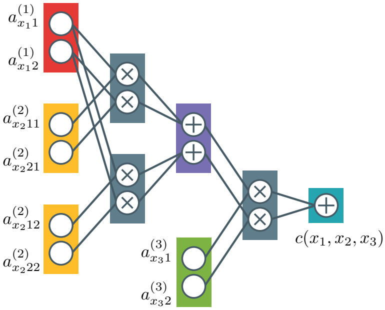
$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1}\, a^{(2)}_{r_1 x_2 r_2}\, a^{(3)}_{r_2 x_3}$$

Schollwoeck, "The density-matrix renormalization group in the age of matrix product states", 2010

Squared circuits are sparse Born machines



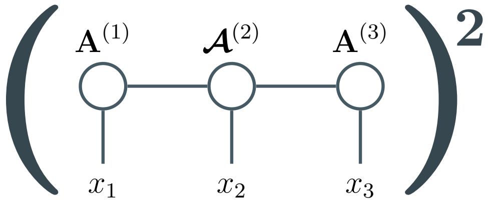
$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1} \, a^{(2)}_{r_1 x_2 r_2} \, a^{(3)}_{r_2 x_3}$$

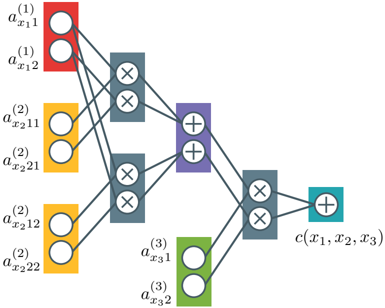


circuit compatible with itself

Schollwoeck, “The density-matrix renormalization group in the age of matrix product states”, 2010

Squared circuits are sparse Born machines





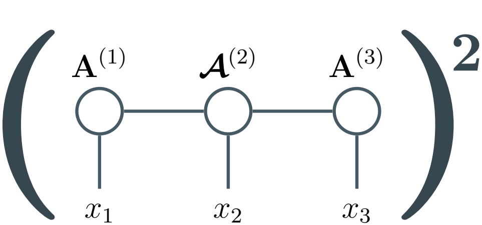
$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1}\, a^{(2)}_{r_1 x_2 r_2}\, a^{(3)}_{r_2 x_3}$$

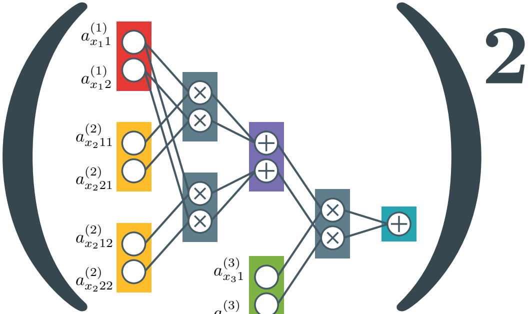
circuit compatible with itself

$$p(x_1, x_2, x_3) \propto \left(t_{x_1 x_2 x_3}\right)^2$$ (Born machine)

Schollwoeck, "The density-matrix renormalization group in the age of matrix product states", 2010

Squared circuits are sparse Born machines





$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^{R} \sum_{r_2=1}^{R} a^{(1)}_{x_1 r_1} \, a^{(2)}_{r_1 x_2 r_2} \, a^{(3)}_{r_2 x_3}$$

$$p(x_1, x_2, x_3) \propto \left(t_{x_1 x_2 x_3} \right)^2$$ (Born machine)

Schollwoeck, "The density-matrix renormalization group in the age of matrix product states", 2010

A limitation of Born machines

(with real tensor-train factorization)

Proposition 5 (Glasser et al. 2019)

There exists non-negative tensors over $2d$ variables that can be factorized as **positive TT of constant rank** 2, but **real Born machines** have at least rank $2^{\Omega(d)}$.

Glasser et al., "Expressive power of tensor-network factorizations for probabilistic modeling", 2019

A limitation of Born machines

(with real tensor-train factorization)

Proposition 5 (Glasser et al. 2019)

There exists non-negative tensors over $2d$ variables that can be factorized as **positive TT of constant rank** 2, but **real Born machines** have at least rank $2^{\Omega(d)}$.

"Can it be generalized to squared circuits?"

Glasser et al., "Expressive power of tensor-network factorizations for probabilistic modeling", 2019

"Can monotonic circuits be more expressive than squared?"

Yes!

Sum of Squares Circuits*

Lorenzo Loconte 🆘 **Stefan Mengel** 🚨 **Antonio Vergari** 🆘

🆘 School of Informatics, University of Edinburgh, UK
🚨 Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France
l.loconte@sms.ed.ac.uk, mengel@cril-lab.fr, avergari@ed.ac.uk

Accepted at AAAI 2025

Poster #840, Hall E, Thursday February 27,

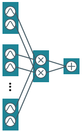
12:30pm-2:30pm

Sum of Squares Circuits*

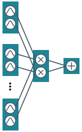
Lorenzo Loconte 🆘 **Stefan Mengel** 🚨 **Antonio Vergari** 🆘

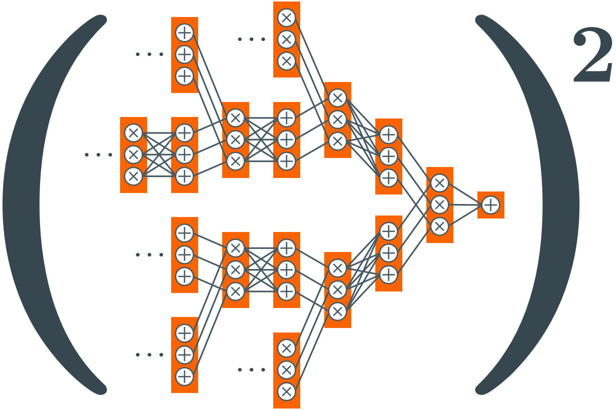
🆘 School of Informatics, University of Edinburgh, UK
🚨 Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France
l.loconte@sms.ed.ac.uk, mengel@cril-lab.fr, avergari@ed.ac.uk

Wang and Van den Broeck
"On the Relationship Between Monotone
and Squared Probabilistic Circuits"
(also at AAAI 2025)

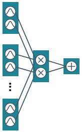


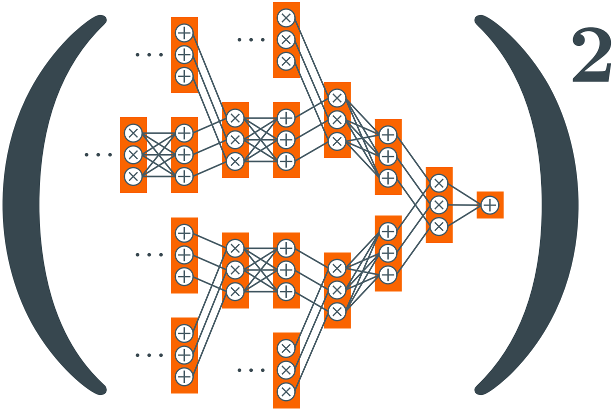
$\exists\, p$ requiring **polysize monotonic circuits**…



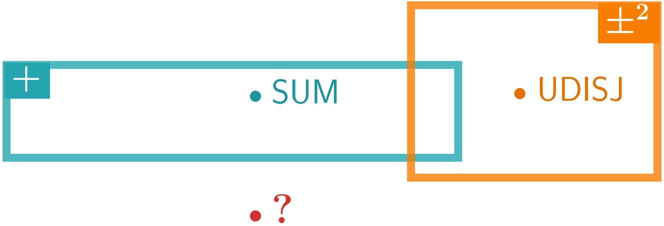


...but require **exponentially large** **squared circuits**





Squaring alone can reduce expressiveness!
(generalizes to factorizations other than tensor-trains)



"How to build circuits more expressive than <u>both</u>?"

Sum of squares (SOS) circuits

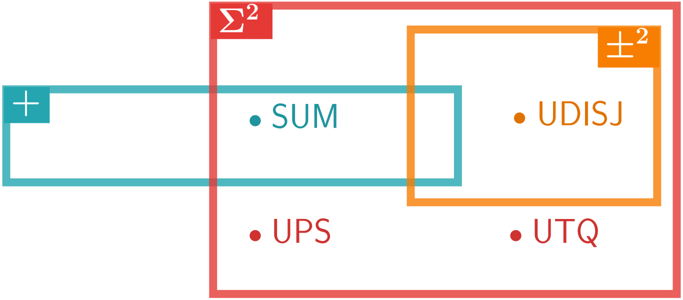
$$p(\mathbf{x}) = \frac{1}{Z}\sum_{i=1}^{r} c_i^2(\mathbf{x}), \quad c_i(\mathbf{x}) \in \mathbb{R}$$

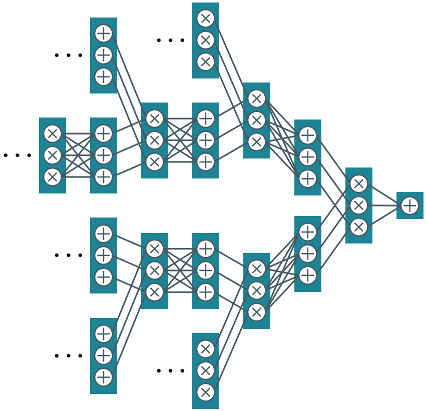
where parameters and input functions can be **negative**

Sum of squares (SOS) circuits

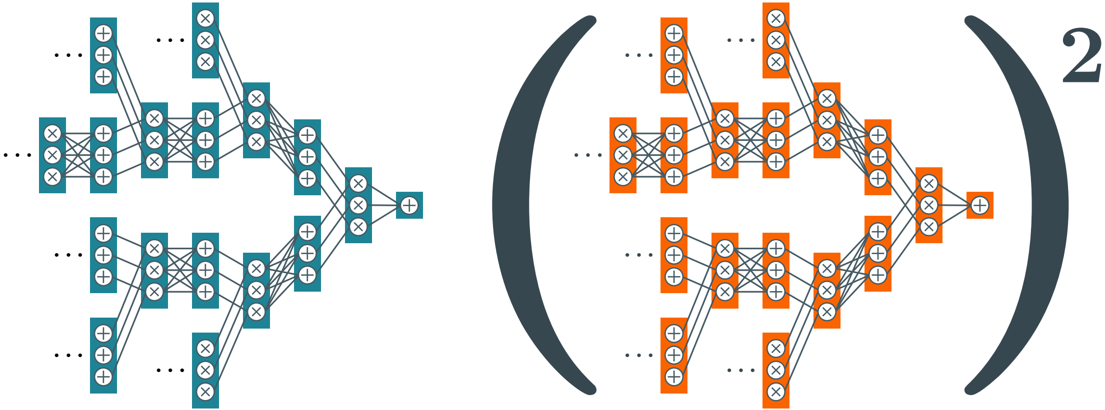
$$p(\mathbf{x}) = \frac{1}{Z}\sum_{i=1}^{r} c_i^2(\mathbf{x}), \quad c_i(\mathbf{x}) \in \mathbb{R}$$

where parameters and input functions can be **negative**

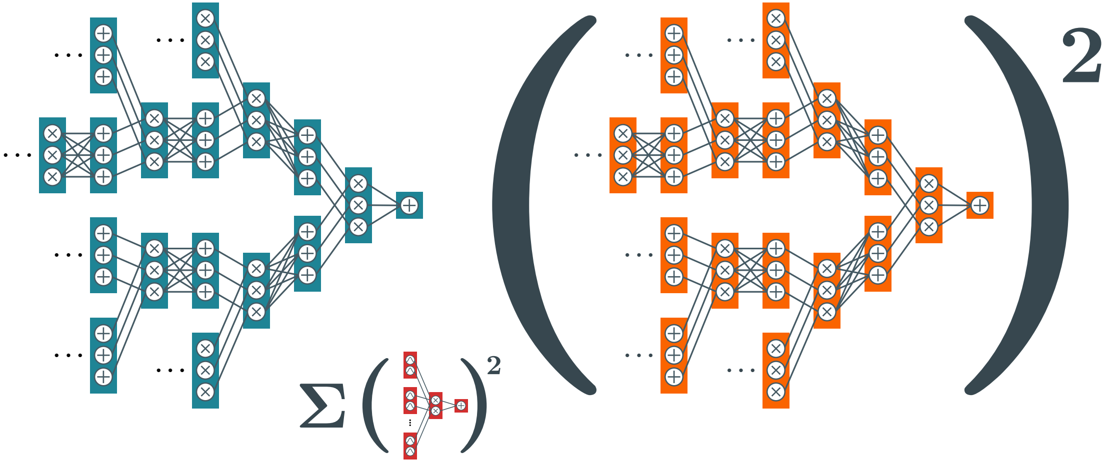




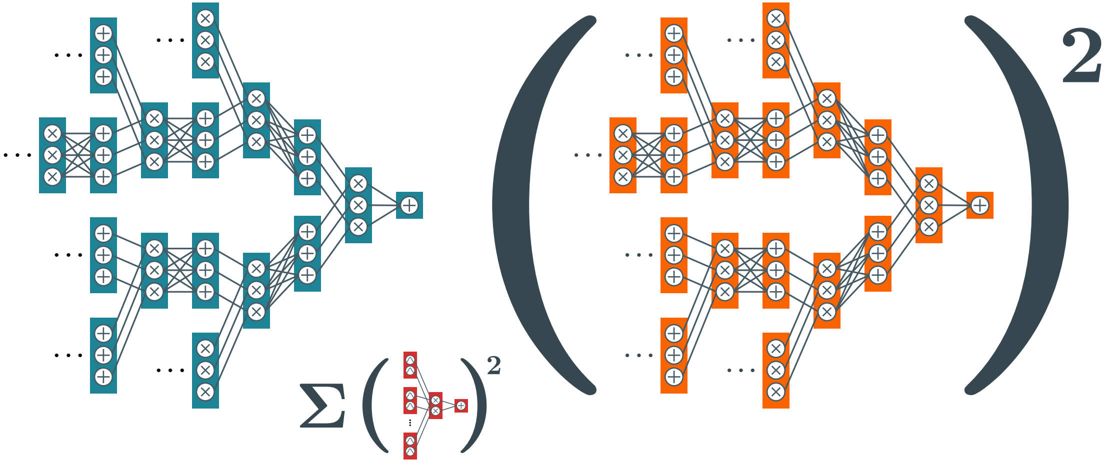
$\exists\, p$ requiring **exponentially large** **monotonic circuits**...



...and also **exponentially large** squared circuits ...

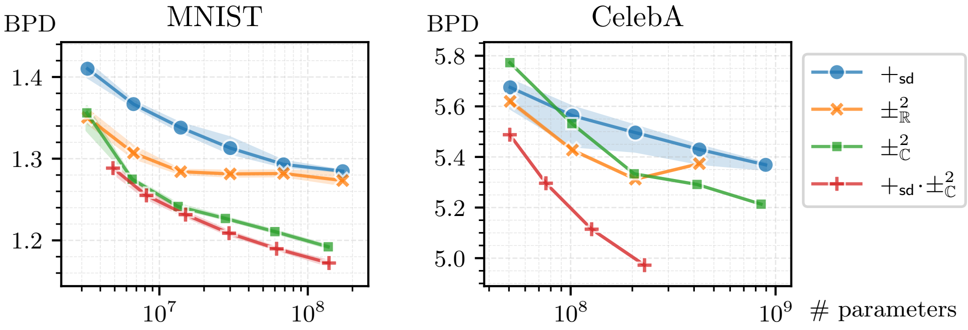


...but a **sum of squares (SOS)** **polysize** circuits



SOS can surpass both expressiveness limitations!

Experiments



***Complex* squared circuits are SOS (and scale better!)**

Takeaways

1 **factorizations and circuits expressiveness results...**
bridge rank and circuit size

Takeaways

1 **factorizations and circuits expressiveness results...**
bridge rank and circuit size

2 **circuits can help proving stronger expressiveness results**
e.g., results from Born machines to squared circuits

Takeaways

1 **factorizations and circuits expressiveness results...**
bridge rank and circuit size

2 **circuits can help proving stronger expressiveness results**
e.g., results from Born machines to squared circuits

3 ***sum of squared circuits*** **are more expressive (use them!)**

Takeaways

Questions?

1 **factorizations and circuits expressiveness results...**
bridge rank and circuit size

2 **circuits can help proving stronger expressiveness results**
e.g., results from Born machines to squared circuits

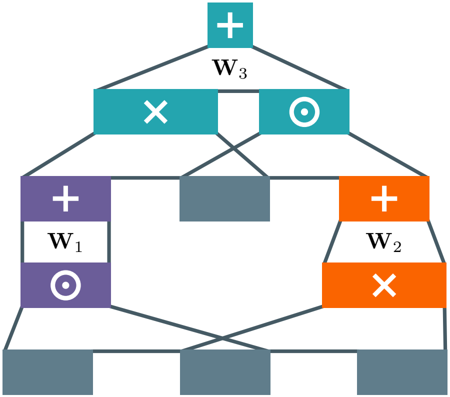
3 ***sum of squared circuits* are more expressive (use them!)**

wait...!

conclusions...?

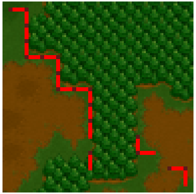
PC Architecture	Region Graph	Sum-Product Layer	Fold
Poon&Domingos (Poon & Domingos, 2011)	PD	$\mathrm{CP}^\top$	✗
RAT-SPN (Peharz et al., 2020c)	RND	Tucker	✗
EiNet (Peharz et al., 2020a)	$\{\mathrm{RND}, \mathrm{PD}\}$	Tucker	✓
HCLT (Liu & Van den Broeck, 2021b)	CL	$\mathrm{CP}^\top$	✓
HMM/$\mathrm{MPS}_{\mathbb{R}\geqslant 0}$ (Glasser et al., 2019)	LT	$\mathrm{CP}^\top$	✗
BM (Han et al., 2018)	LT	$\mathrm{CP}^\top$	✗
TTDE (Novikov et al., 2021)	LT	$\mathrm{CP}^\top$	✗
NPC^2 (Loconte et al., 2024)	$\{\mathrm{LT}, \mathrm{RND}\}$	$\{\mathrm{CP}^\top, \mathrm{Tucker}\}$	✓
TTN (Cheng et al., 2019)	QT-2	Tucker	✗
Mix & Match (our pipeline)	$\left\{\begin{array}{l}\mathrm{RND}, \mathrm{PD}, \mathrm{LT}, \\ \mathrm{CL}, \mathrm{QG}, \mathrm{QT\text{-}2}, \mathrm{QT\text{-}4}\end{array}\right\}$ ×	$\{\mathrm{Tucker}, \mathrm{CP}, \mathrm{CP}^\top\} \cup \{\mathrm{CP}^\mathrm{S}, \mathrm{CP}^\mathrm{XS} \mid \mathrm{Fold}\ ✓\}$ ×	{✗, ✓}

takeaway #1: unifying a fragmented literature

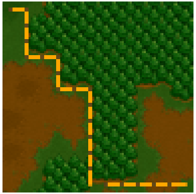


takeaway #2: easily build novel factorizations

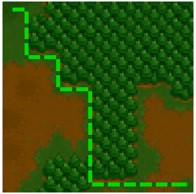
Ground Truth



ResNet-18

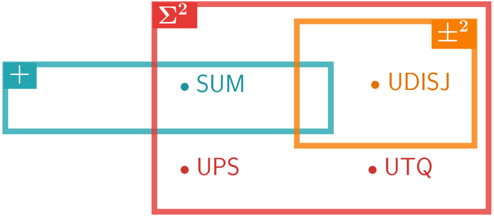


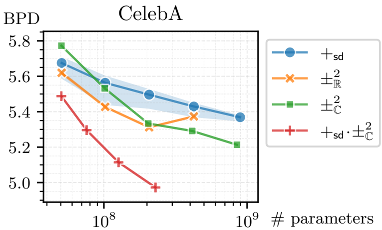
Semantic Loss



circuits

***takeaway #3*: use them for efficient & reliable inference**





***takeaway #4*: SOS circuits are provably more expressive factorizations**

What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?

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

Antonio Mari *antonio.mari@epfl.ch*
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Gennaro Gala *g.gala@tue.nl*
Eindhoven University of Technology, NL

Robert Peharz *robert.peharz@tugraz.at*
Graz University of Technology, Austria

Cassio de Campos *c.decampos@tue.nl*
Eindhoven University of Technology, NL

Erik Quaeghebeur *e.quaeghebeur@tue.nl*
Eindhoven University of Technology, NL

Gennaro Vessio *gennaro.vessio@uniba.it*
Computer Science Department, University of Bari Aldo Moro, IT

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

accepted at TMLR **featured certification**

cirkit

learning & reasoning with circuits in pytorch

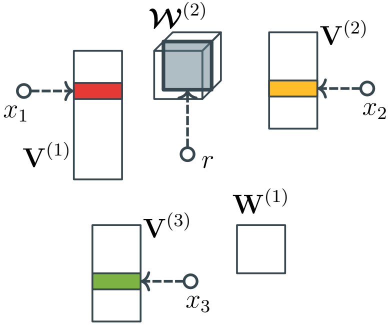
https://github.com/april-tools/cirkit

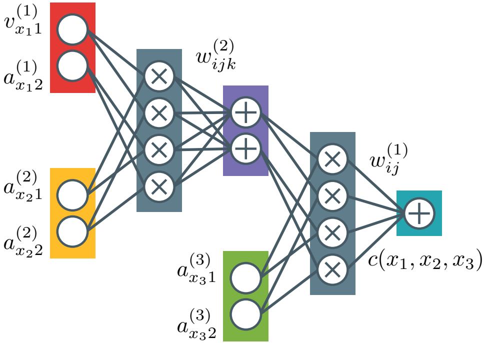
colorai

connecting
low-rank
representations
in ai

***workshop at AAAI-25*, March 4**

april-tools.github.io/colorai/


$\mathcal{W}^{(2)}$
$\mathbf{V}^{(2)}$
x_1
x_2
$\mathbf{V}^{(1)}$
r
$\mathbf{W}^{(1)}$
$\mathbf{V}^{(3)}$
x_3


$v^{(1)}_{x_1 1}$
$a^{(1)}_{x_1 2}$
$w^{(2)}_{ijk}$
$a^{(2)}_{x_2 1}$
$a^{(2)}_{x_2 2}$
$w^{(1)}_{ij}$
$a^{(3)}_{x_3 1}$
$a^{(3)}_{x_3 2}$
$c(x_1, x_2, x_3)$

questions?