



from tensor factorizations to circuits (and back)

lorenzo loconte

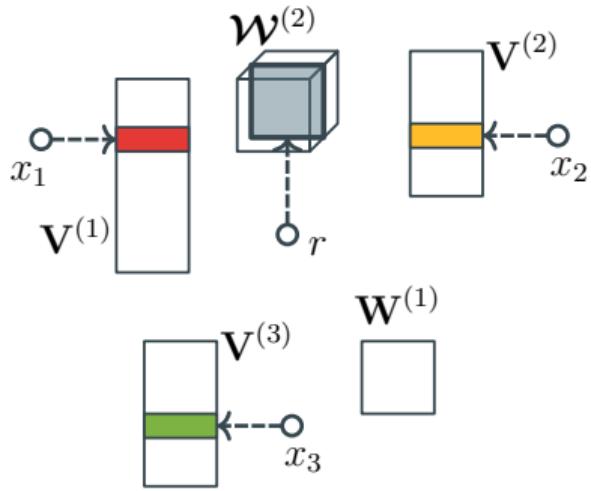
 @loreloc

X @loreloc_

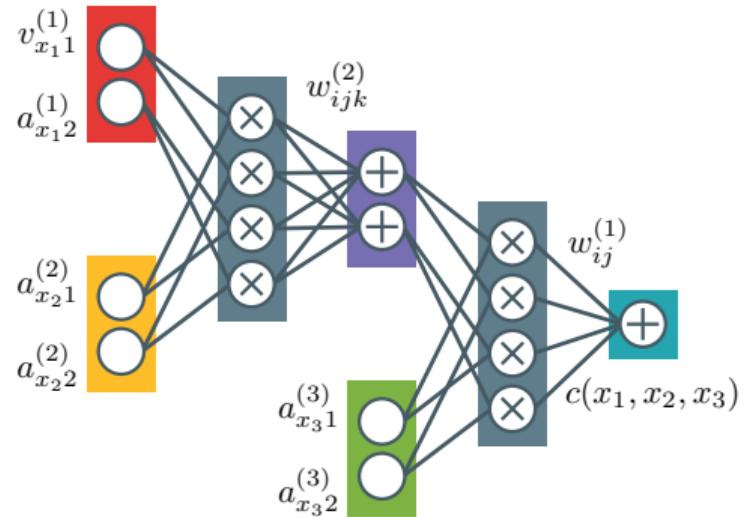
antonio vergari

 @nolovedeeplearning

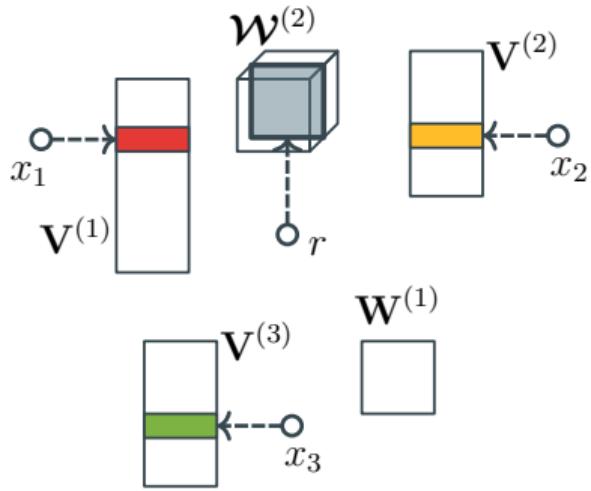
X @tetraduzione



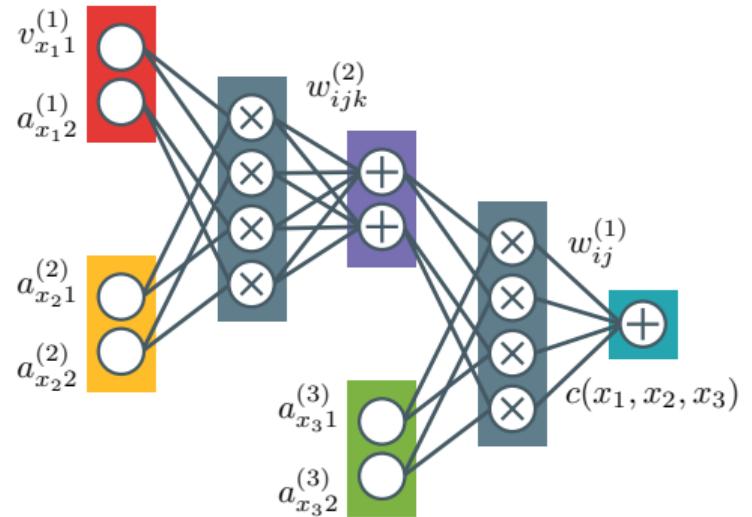
tensor factorizations



circuits

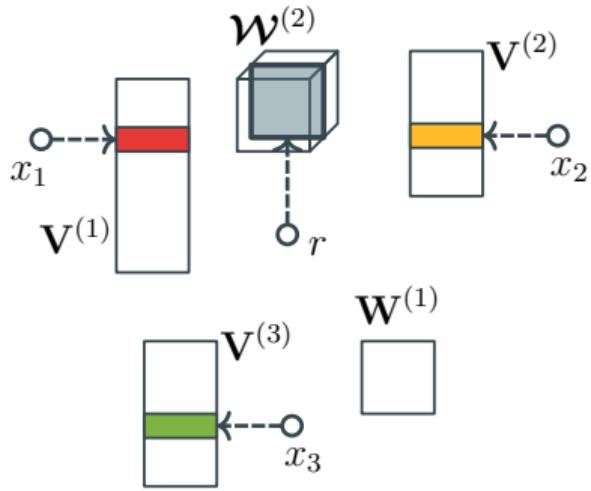


tensor factorizations

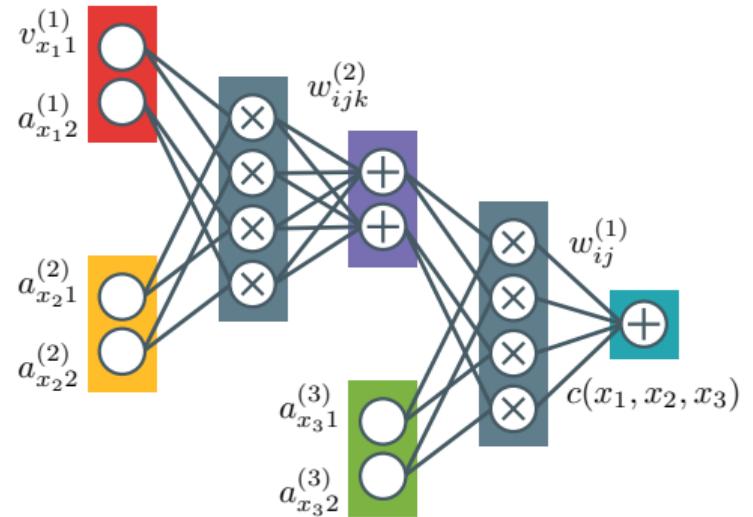


circuits

they look quite different...!?



tensor factorizations



circuits

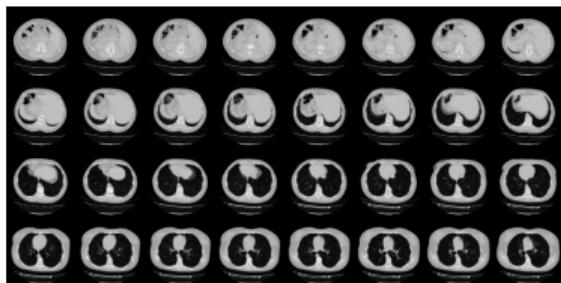
what can one take from another...!?

why tensor factorizations? (1/4)

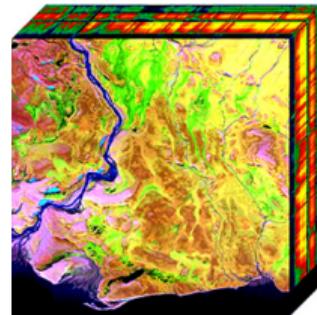
high-dimensional data as tensors



(PBS Nature)



(H. Zunair)



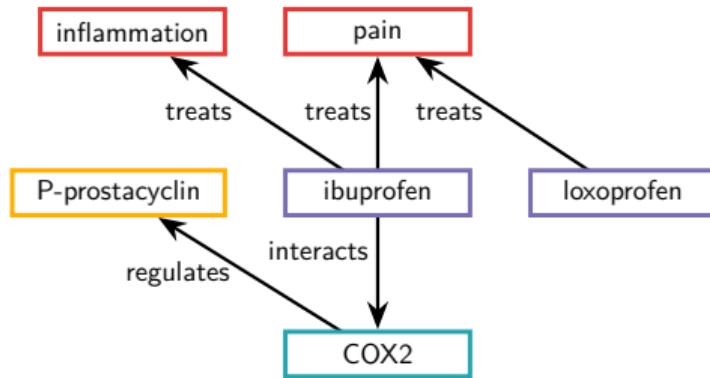
(N. M. Short)

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

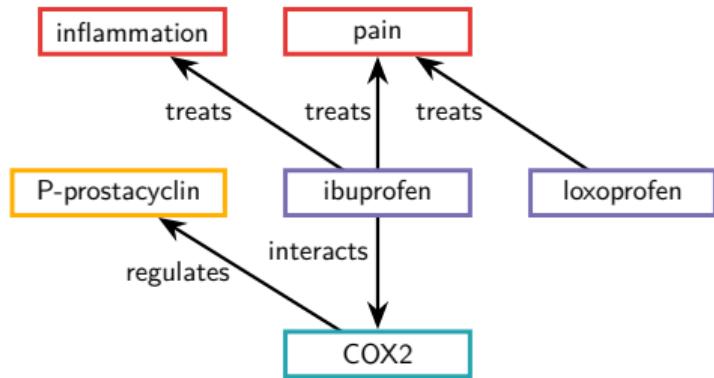


Entities:

- Drugs
- Proteins
- Symptoms
- Functions

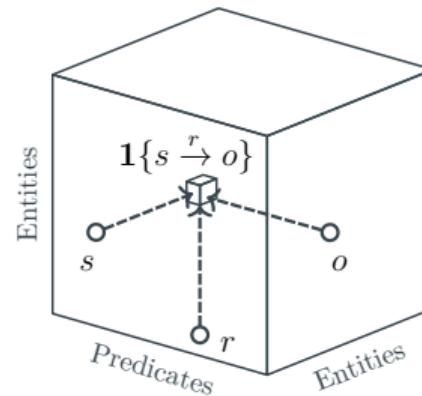
Why tensor factorizations? (2/4)

graphs as tensors



Entities:

- Drugs
- Proteins
- Symptoms
- Functions

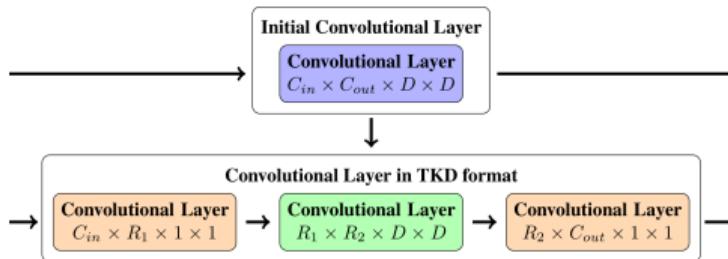


Knowledge base as Boolean tensor

[Nickel et al. 2016]

why tensor factorizations? (3/4)

compress ML models

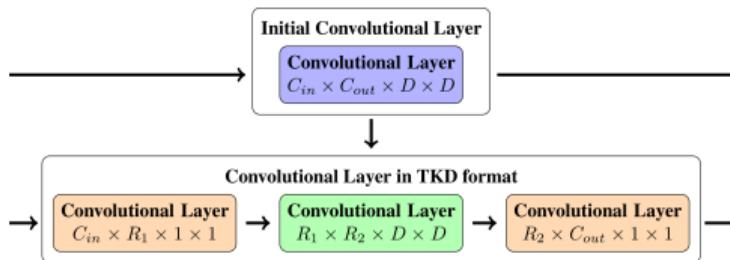


Compress convolutional layers

[Phan et al. 2020]

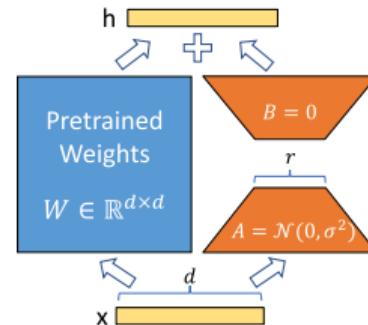
why tensor factorizations? (3/4)

compress ML models



Compress convolutional layers

[Phan et al. 2020]

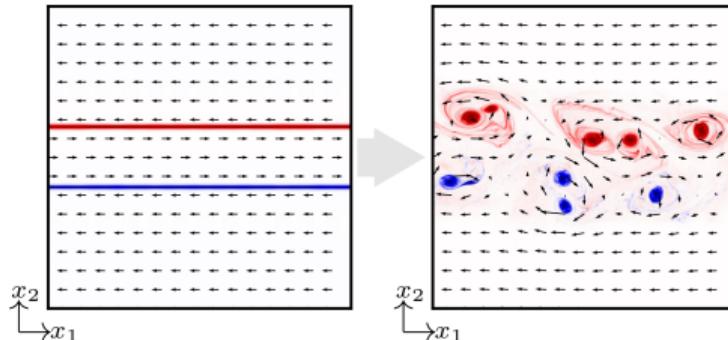


Low-rank adapters in LLMs

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

why tensor factorizations? (4/4)

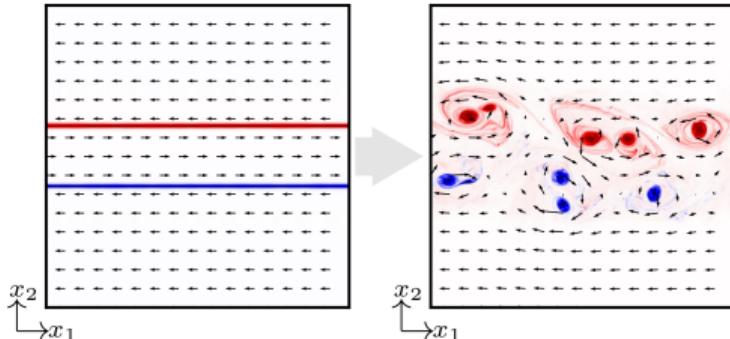
Tensors 4 physics



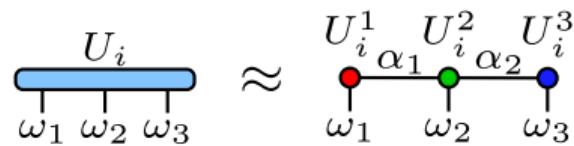
**Fluid velocity vectors computed
in exponentially many points...**

why tensor factorizations? (4/4)

Tensors 4 physics



**Fluid velocity vectors computed
in exponentially many points...**

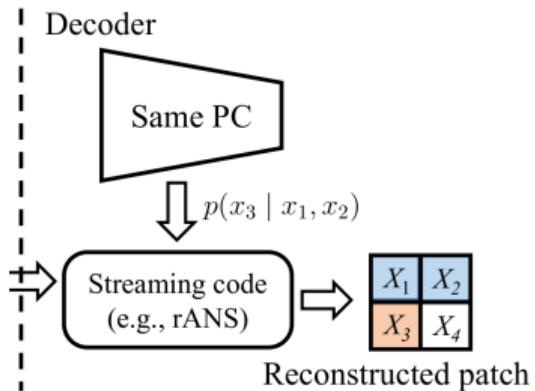


**...by factorizing them into
chains of low-rank tensors**

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

why circuits? (1/3)

efficient probabilistic inference

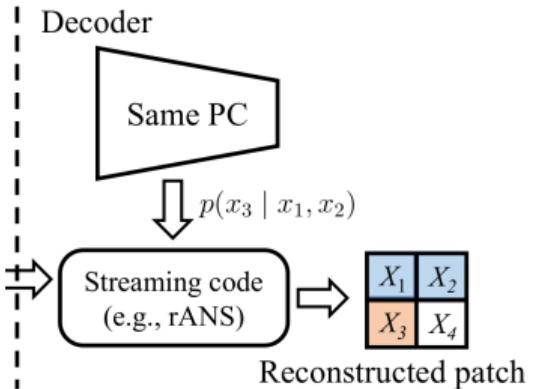


Fast lossless (de)compression

[Liu, Mandt, and Van den Broeck 2022]

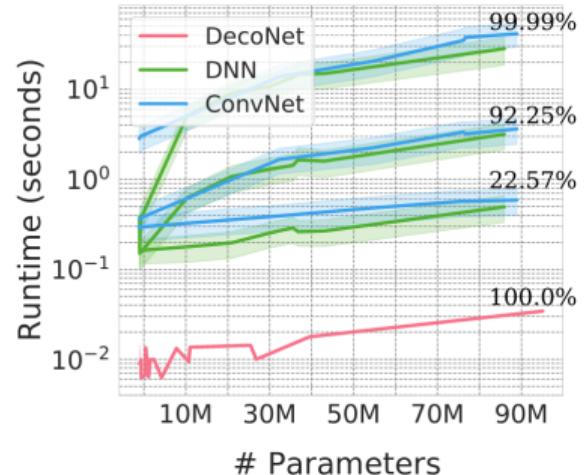
why circuits? (1/3)

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

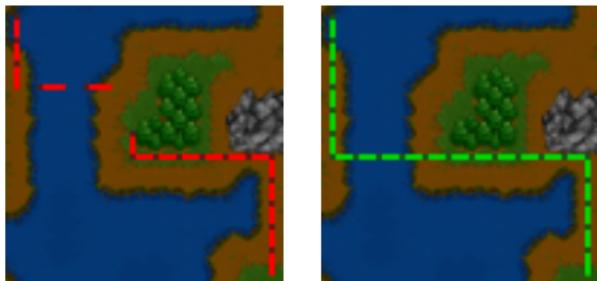


**Constrained multi-label
prediction (w/ guarantees)**

[Ahmed et al. 2022]

why circuits? (2/3)

they enable neuro-symbolic AI

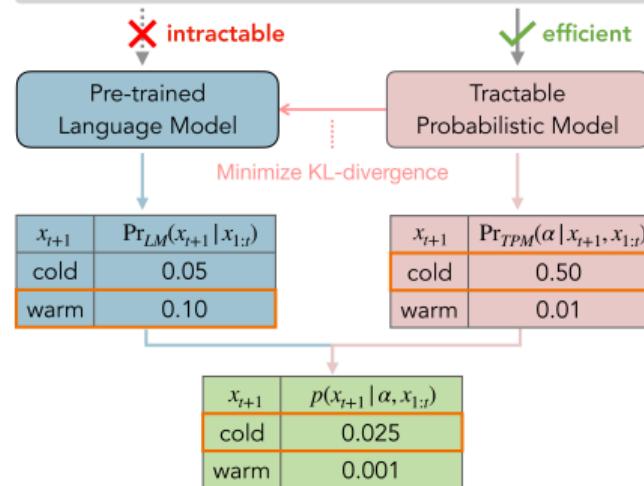


Constrained multi-label
prediction (w/ guarantees)

[Ahmed et al. 2022]

Lexical Constraint α : sentence contains keyword "winter"

Constrained Generation: $\Pr(x_{t+1} | \alpha, x_{1:t} = \text{"the weather is"})$

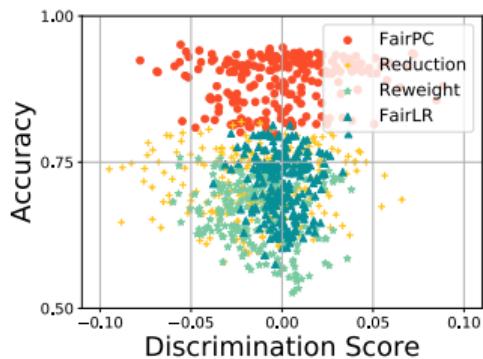


Constrained text generation

[Zhang et al. 2023]

why circuits? (3/3)

they are reliable and interpretable

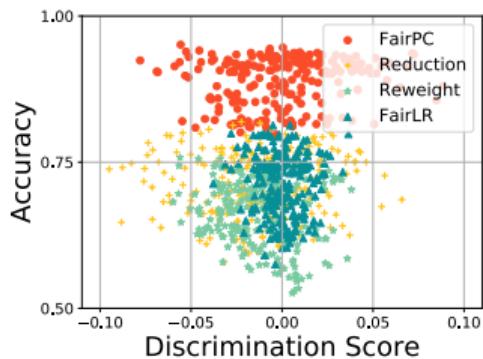


Encode group fairness

[Choi, Dang, and Van den Broeck 2020]

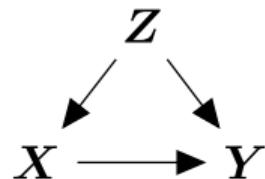
why circuits? (3/3)

they are reliable and interpretable



Encode group fairness

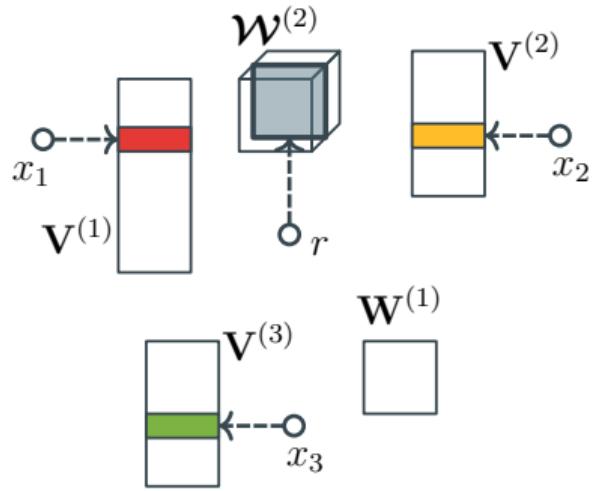
[Choi, Dang, and Van den Broeck 2020]



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

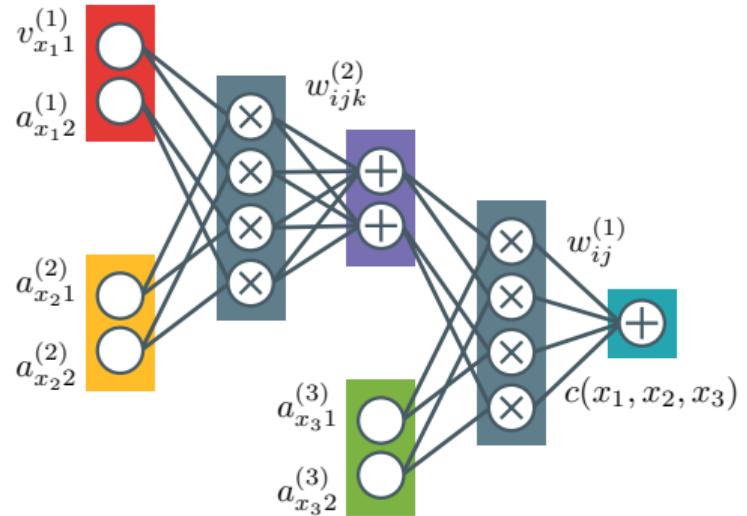
Tractable causal inference

[Wang and Kwiatkowska 2023]



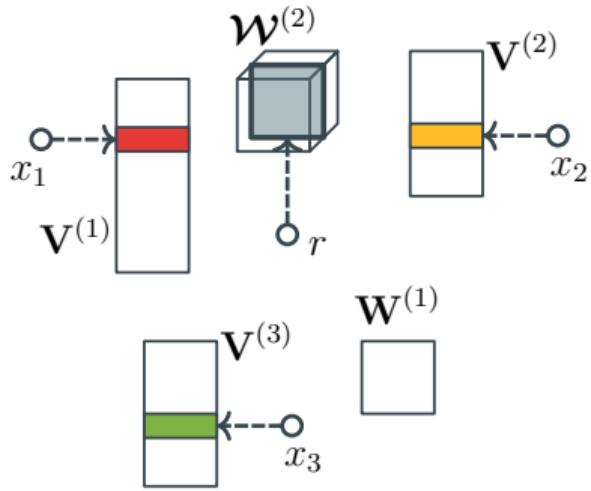
tensor factorizations

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

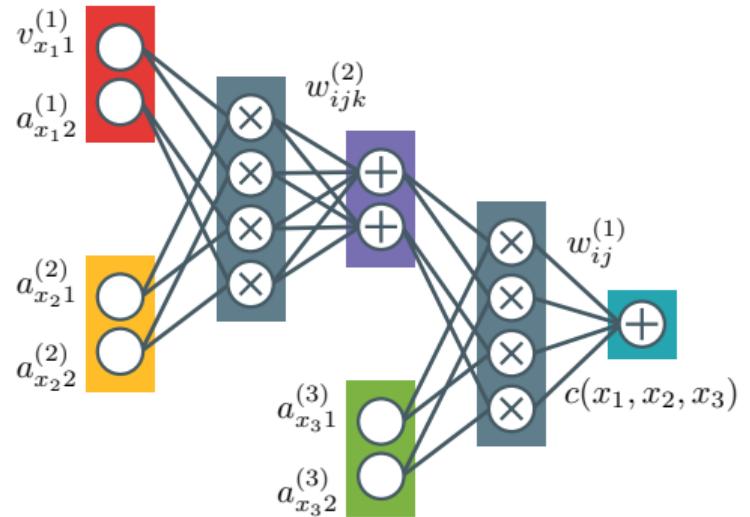


circuits

*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

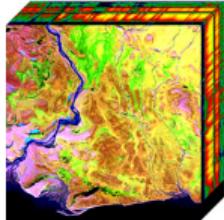
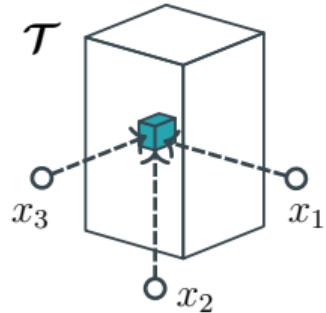
1

connecting *tensor factorizations* and *circuits*

tl;dr

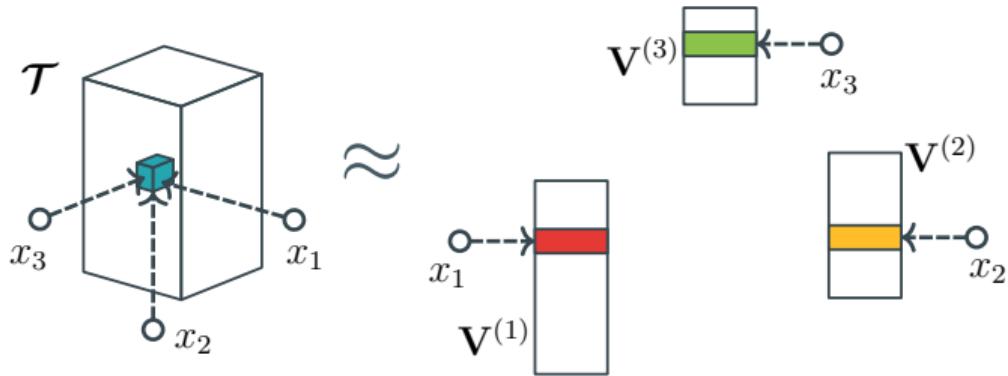
***“Understand when and how
a tensor factorization can be
exactly encoded as a circuit representation”***

Canonical polyadic (CP)



(N. M. Short)

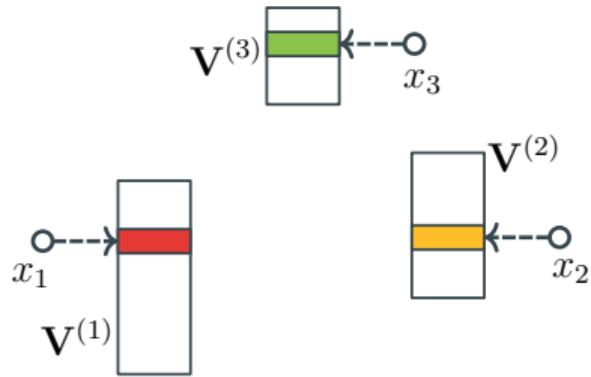
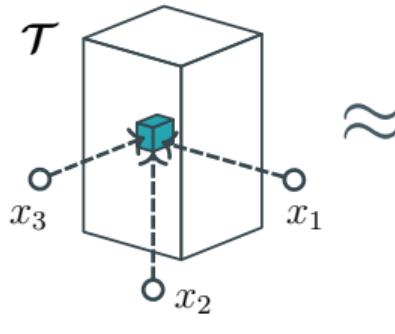
Canonical polyadic (CP)



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

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

Canonical polyadic (**CP**)



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

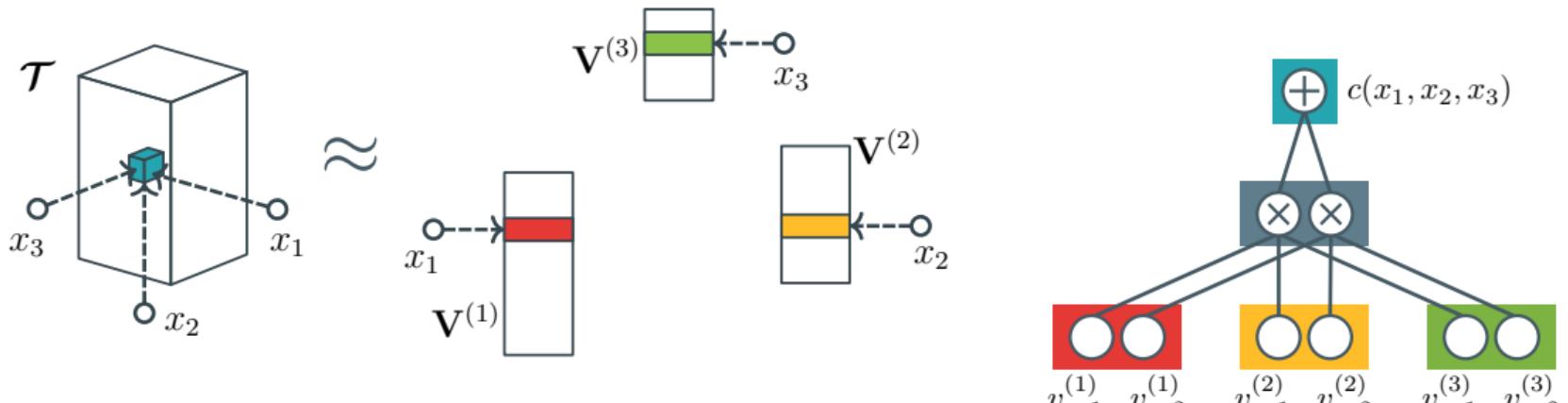
$$\mathbf{V}^{(1)} \in \mathbb{R}^{I_1 \times R}, \mathbf{V}^{(2)} \in \mathbb{R}^{I_2 \times R}, \mathbf{V}^{(3)} \in \mathbb{R}^{I_3 \times R}$$

$$v_{x_1 1}^{(1)} \quad v_{x_1 2}^{(1)}$$

$$v_{x_2 1}^{(2)} \quad v_{x_2 2}^{(2)}$$

$$v_{x_3 1}^{(3)} \quad v_{x_3 2}^{(3)}$$

Canonical polyadic (**CP**)



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

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

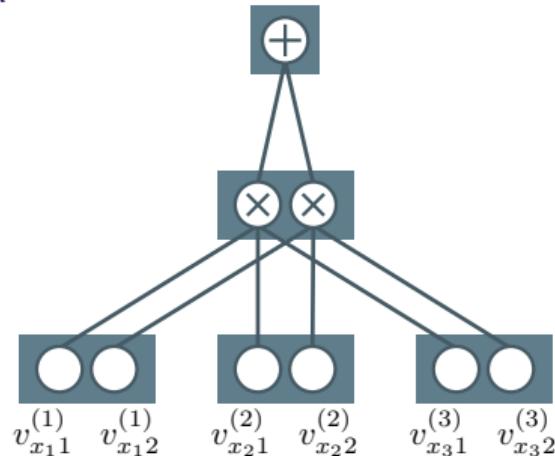
How to evaluate a circuit?

A circuit computes a tensor entry at some index

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

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

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



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

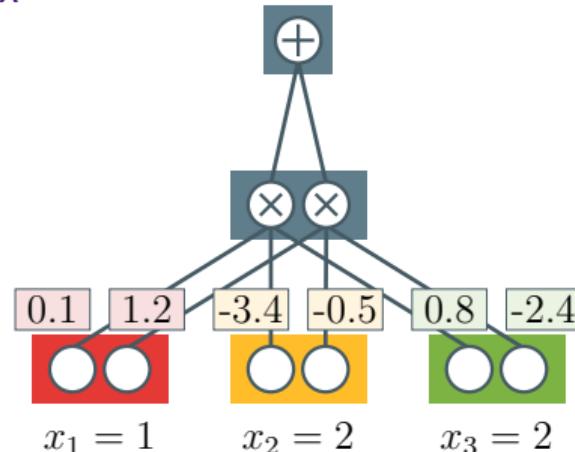
How to evaluate a circuit?

A circuit computes a tensor entry at some index

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

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

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



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

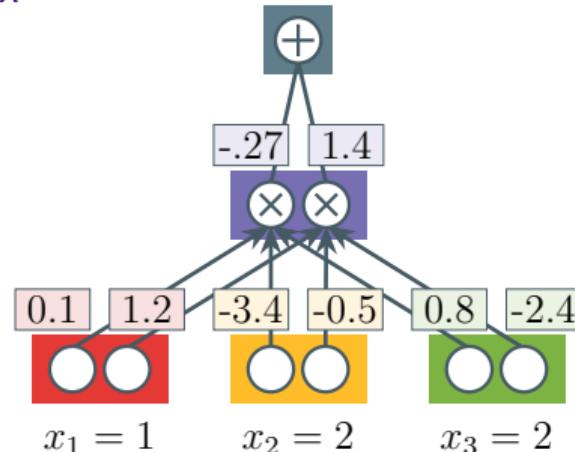
How to evaluate a circuit?

A circuit computes a tensor entry at some index

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

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

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



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

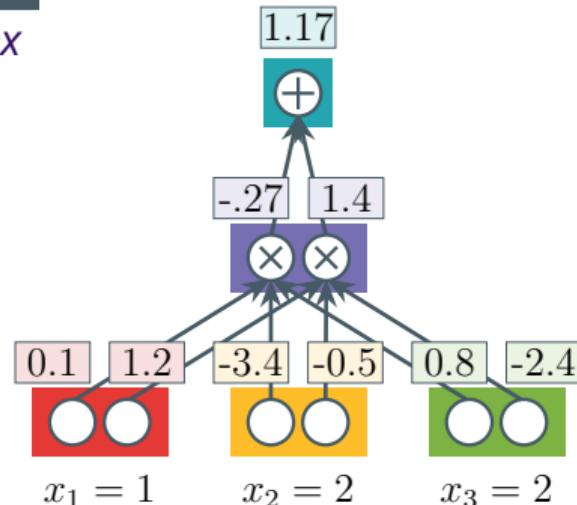
How to evaluate a circuit?

A circuit computes a tensor entry at some index

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

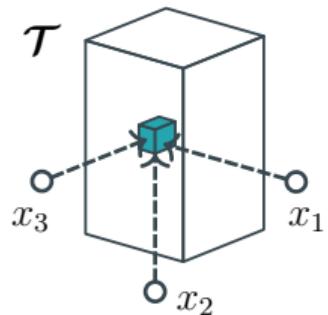
$$\mathbf{V}^{(2)} = \begin{bmatrix} 2.5 & 0.0 \\ -3.4 & -0.5 \\ -0.1 & 2.2 \end{bmatrix}$$

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

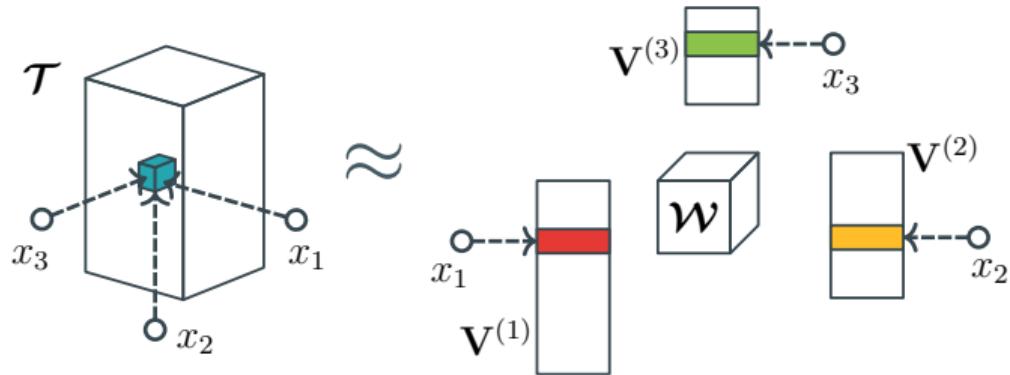


$$t_{122} \approx c(1, 2, 2) = \sum_{r=1}^R v_{1r}^{(1)} v_{2r}^{(2)} v_{2r}^{(3)}$$

Tucker



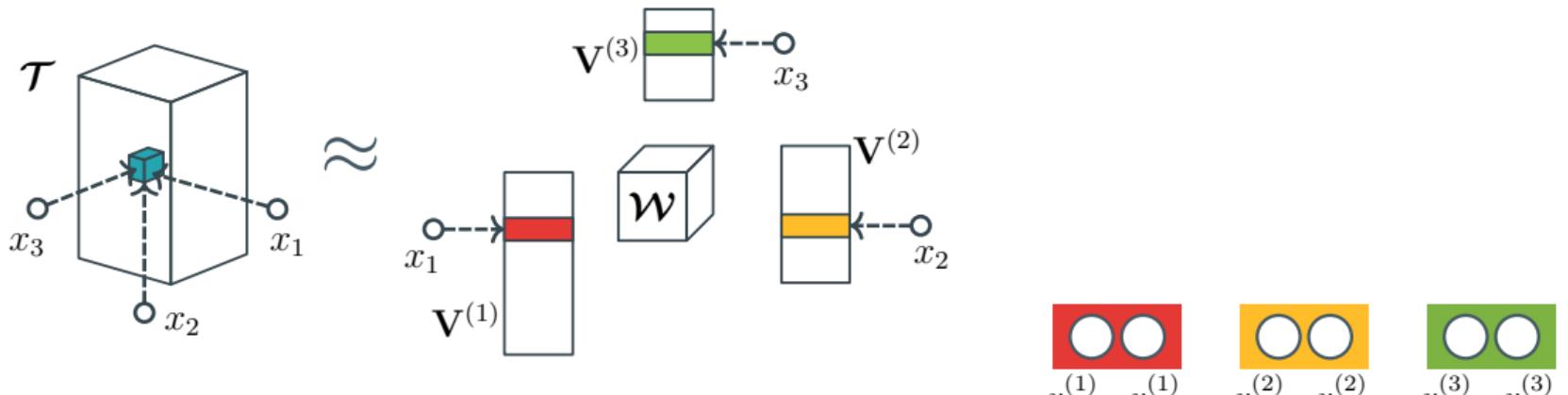
Tucker



$$t_{x_1 x_2 x_3} \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} w_{r_1 r_2 r_3} v_{x_1 r_1}^{(1)} v_{x_2 r_2}^{(2)} v_{x_3 r_3}^{(3)}$$

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

Tucker

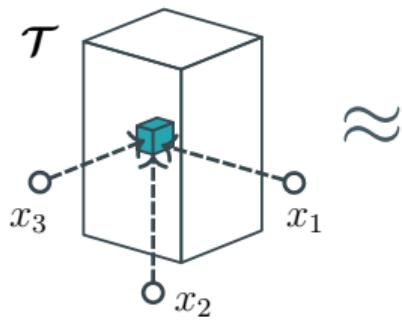
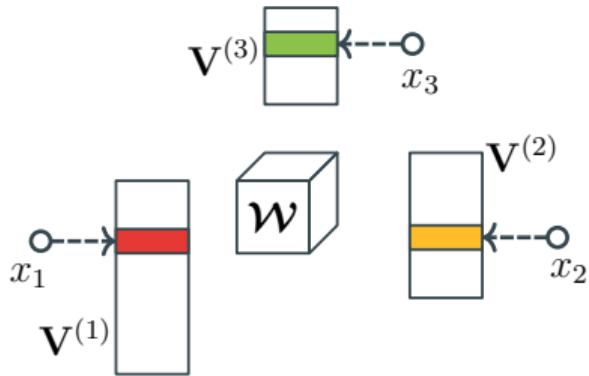


$$t_{x_1 x_2 x_3} \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} w_{r_1 r_2 r_3} v_{x_1 r_1}^{(1)} v_{x_2 r_2}^{(2)} v_{x_3 r_3}^{(3)}$$

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

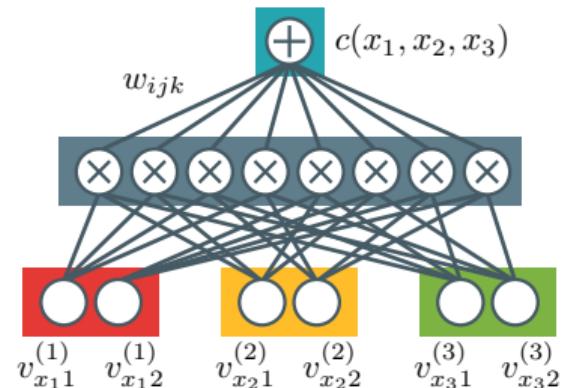
$v_{x_1 1}^{(1)}$	$v_{x_1 2}^{(1)}$	$v_{x_2 1}^{(2)}$	$v_{x_2 2}^{(2)}$	$v_{x_3 1}^{(3)}$	$v_{x_3 2}^{(3)}$
-------------------	-------------------	-------------------	-------------------	-------------------	-------------------

Tucker


 \approx


$$t_{x_1 x_2 x_3} \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} w_{r_1 r_2 r_3} v_{x_1 r_1}^{(1)} v_{x_2 r_2}^{(2)} v_{x_3 r_3}^{(3)}$$

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



The layer-wise circuit definition

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



The layer-wise circuit definition

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

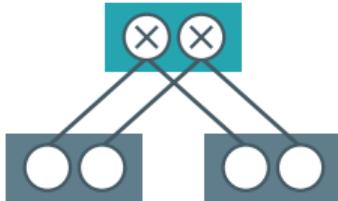


The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$



The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$



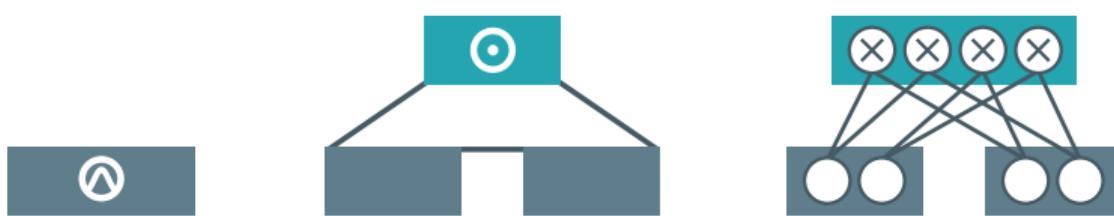
The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$



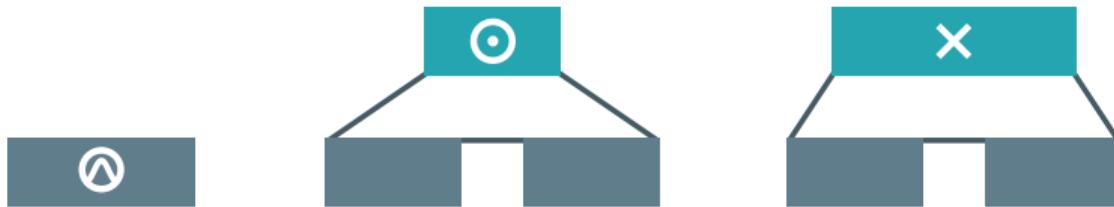
The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$



The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$

A linear projection of a layer is a layer

$$\ell(\mathbf{x}) = \mathbf{W}\ell_i(\mathbf{x})$$



The layer-wise circuit definition

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

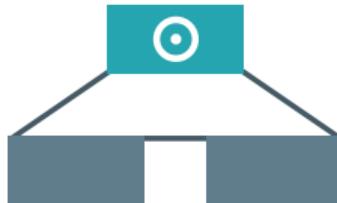
The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$

A linear projection of a layer is a layer

$$\ell(\mathbf{x}) = \mathbf{W}\ell_i(\mathbf{x})$$



The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$

A linear projection of two layers is a layer

$$\ell(\mathbf{x}) = \mathbf{W}\text{concat}(\ell_i(\mathbf{x}), \ell_{ii}(\mathbf{x}))$$



The layer-wise circuit definition

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

The product of two layers is a layer

$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \odot \ell_{ii}(\mathbf{x}) \quad (\text{Hadamard})$$

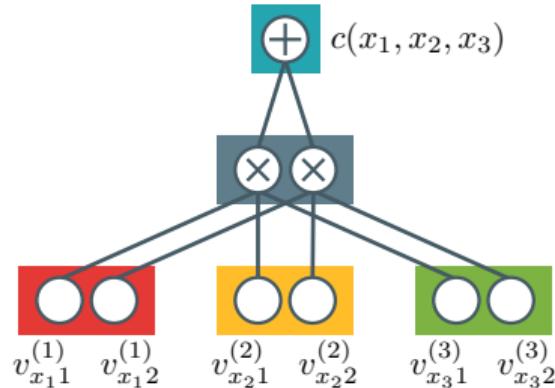
$$\ell(\mathbf{x}) = \ell_i(\mathbf{x}) \otimes \ell_{ii}(\mathbf{x}) \quad (\text{Kronecker})$$

A linear projection of two layers is a layer

$$\ell(\mathbf{x}) = \mathbf{W}\text{concat}(\ell_i(\mathbf{x}), \ell_{ii}(\mathbf{x}))$$

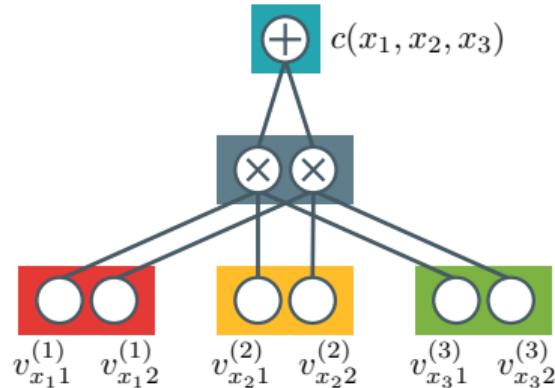


The layer-wise circuit definition

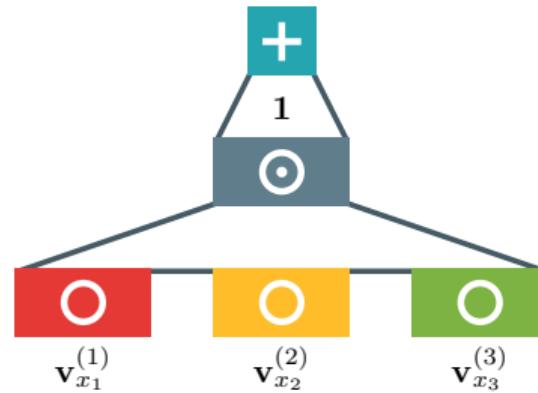


CP circuit

The layer-wise circuit definition

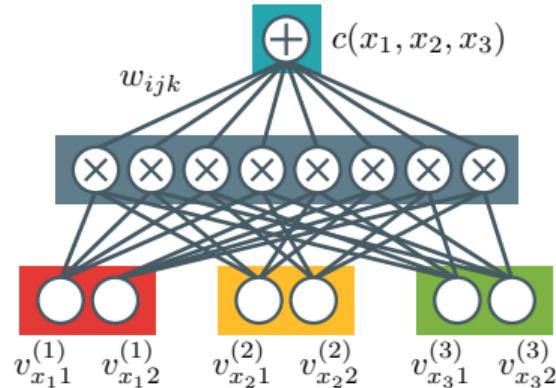


CP circuit



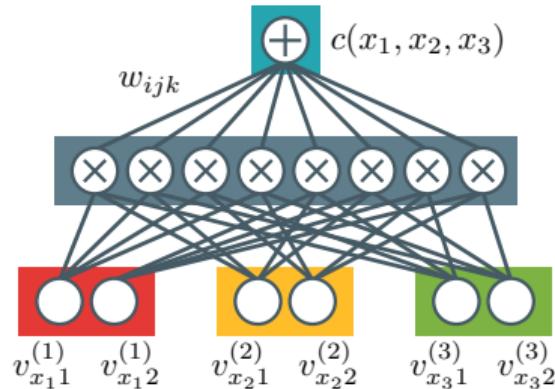
CP circuit

The layer-wise circuit definition

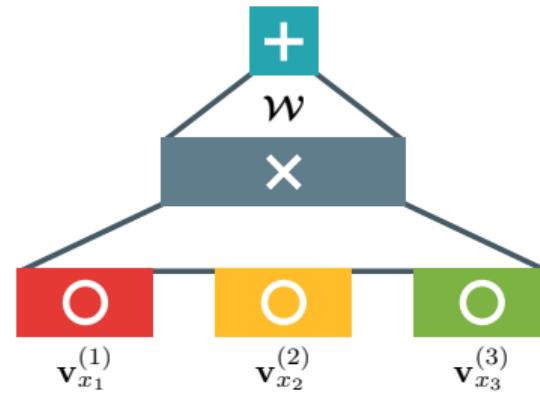


Tucker circuit

The layer-wise circuit definition



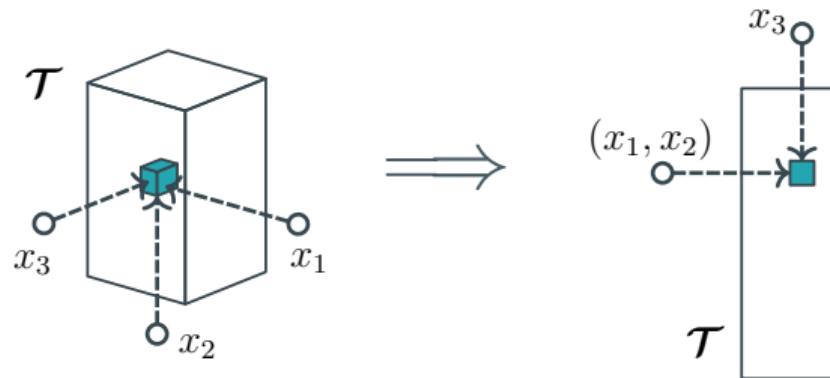
Tucker circuit



Tucker circuit

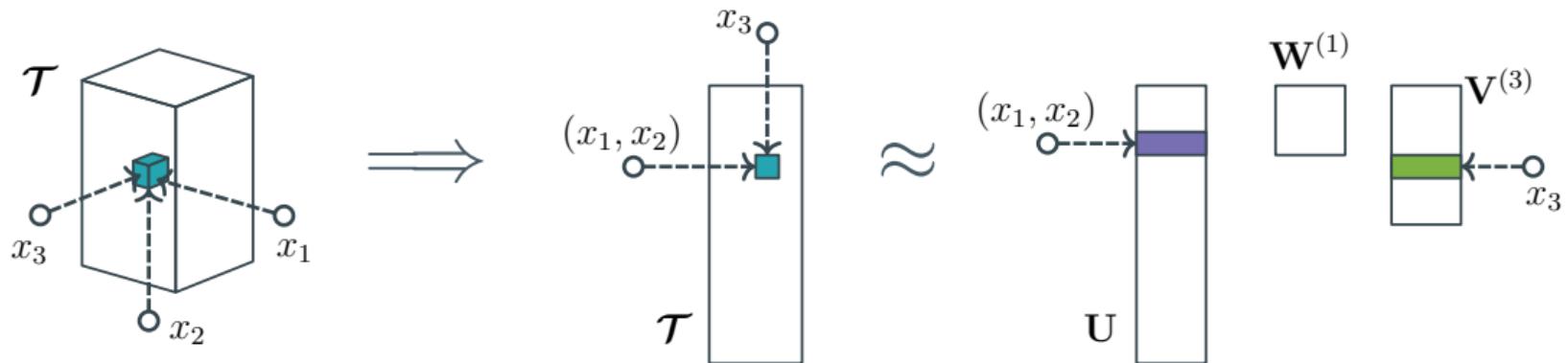
going deeper: hierarchical Tucker

level-one factorization



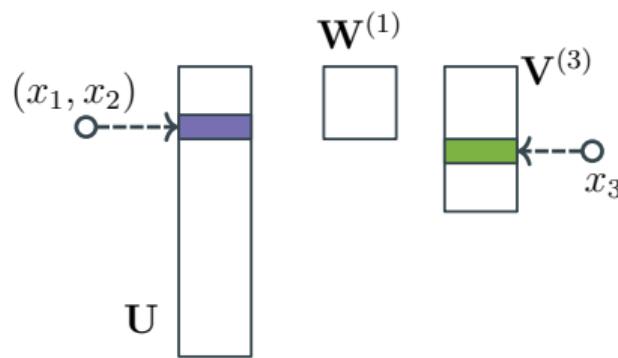
going deeper: hierarchical Tucker

level-one factorization



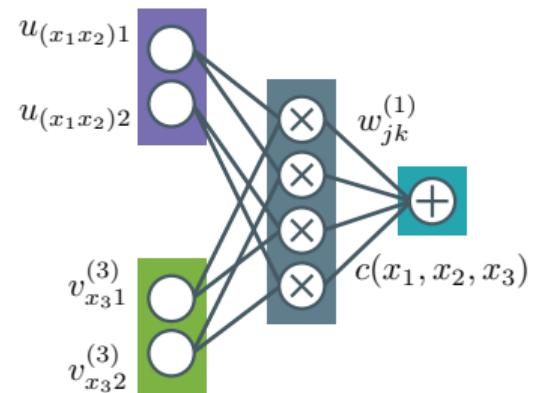
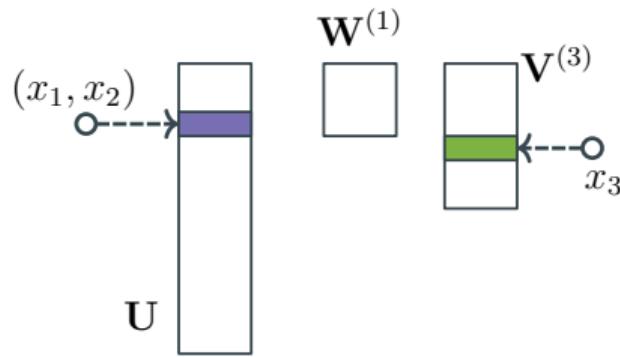
going deeper: hierarchical Tucker

level-one factorization as a circuit



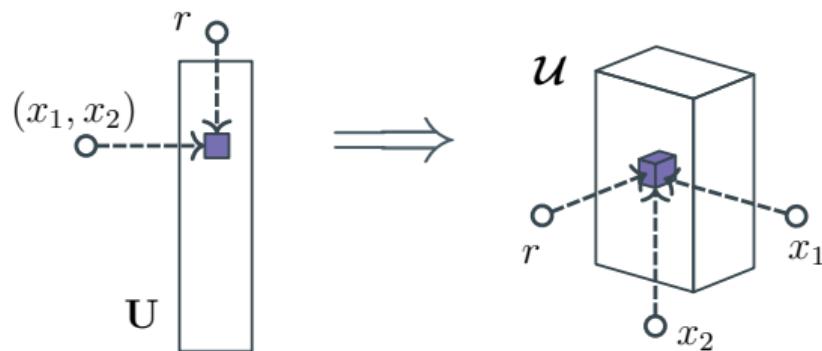
going deeper: hierarchical Tucker

level-one factorization as a circuit



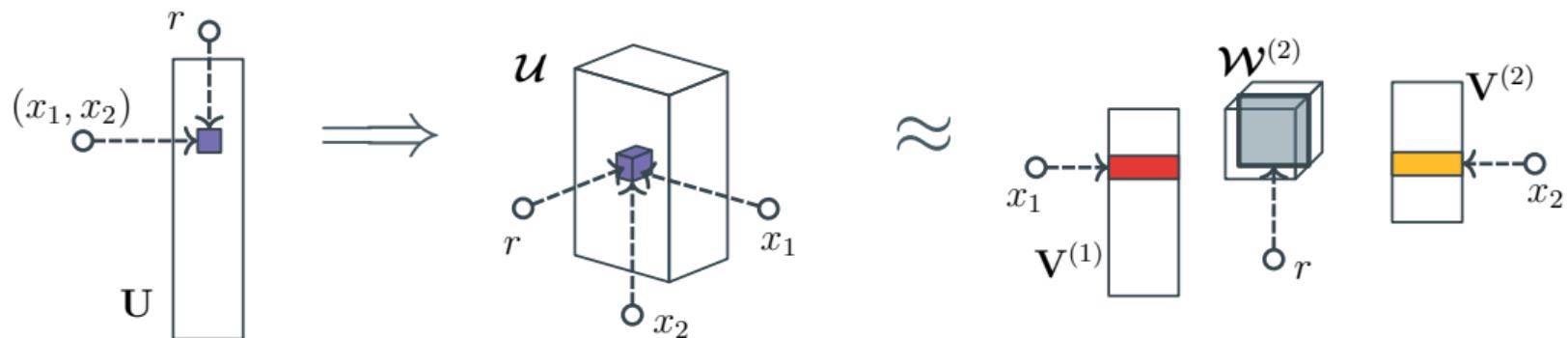
going deeper: hierarchical Tucker

level-two factorization



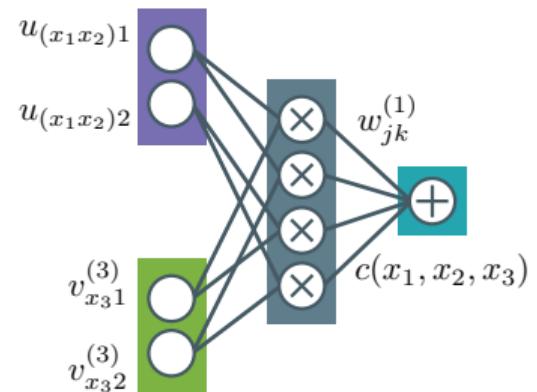
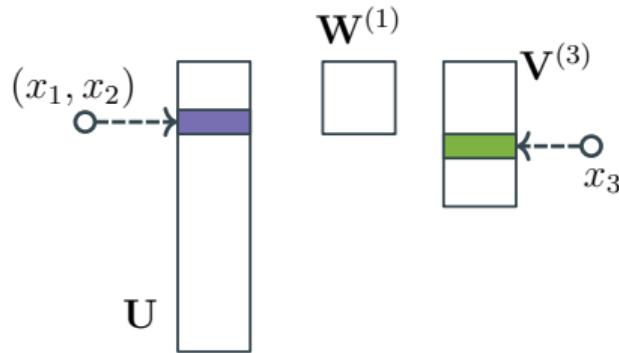
going deeper: hierarchical Tucker

level-two factorization



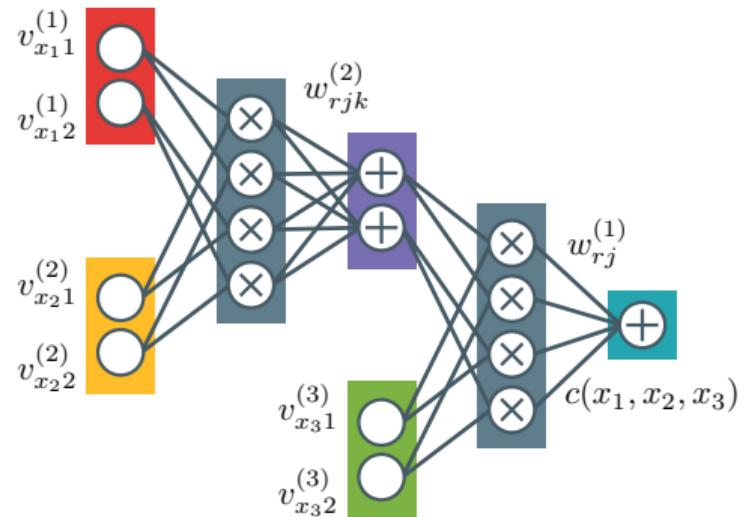
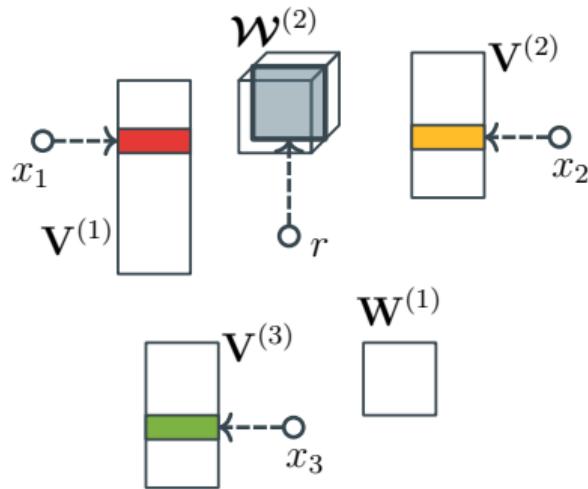
going deeper: hierarchical Tucker

nested factorizations are deep circuits



going deeper: hierarchical Tucker

nested factorizations are deep circuits



Tensor networks

the Penrose graphical notation



scalar

Tensor networks

the Penrose graphical notation



scalar

vector

Tensor networks

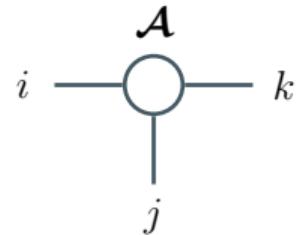
the Penrose graphical notation



vector

Tensor networks

the Penrose graphical notation



scalar

vector

matrix

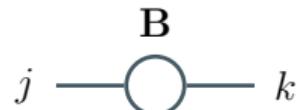
tensor

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

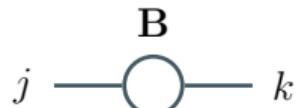
Orús, "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States", 2013

Tensor networks

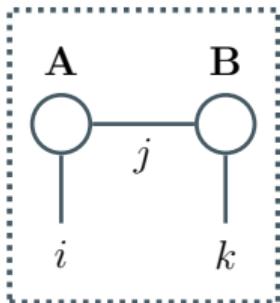
matrix factorization & contraction



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



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



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

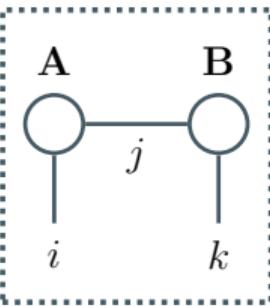
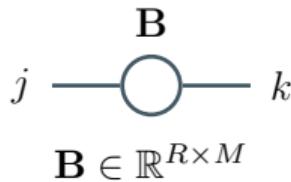
$$c_{ik} = \sum_{j=1}^R a_{ij} b_{jk}$$

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

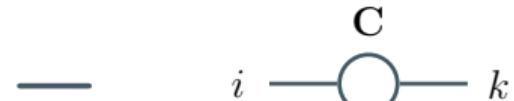
Tensor networks

matrix factorization & contraction



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

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

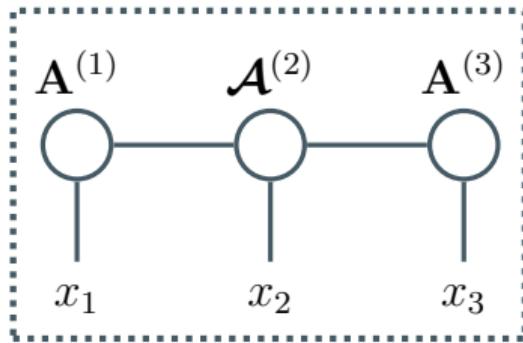


Biamonte and Bergholm, "Tensor Networks in a Nutshell", 2017

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

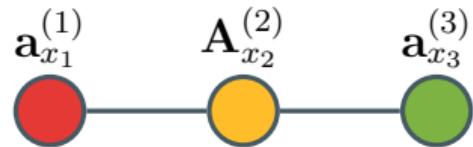


$$\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$$

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

Tensor trains are circuits

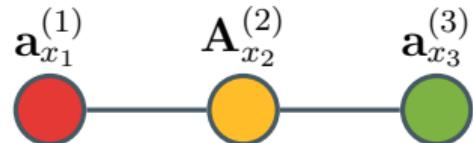
also called matrix-product states



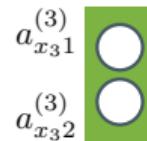
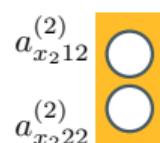
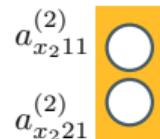
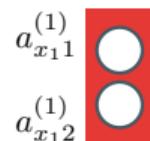
$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$

Tensor trains are circuits

also called matrix-product states

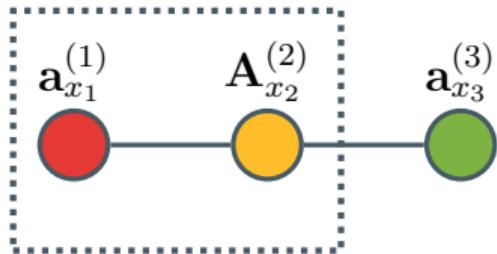


$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$

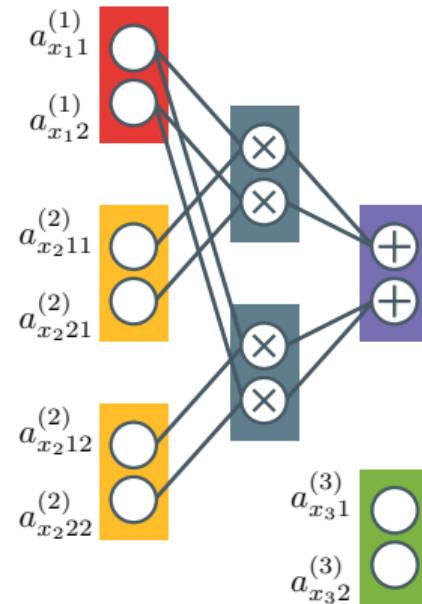


Tensor trains are circuits

also called matrix-product states

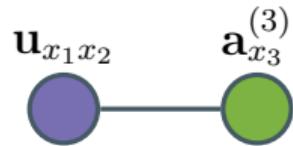


$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$

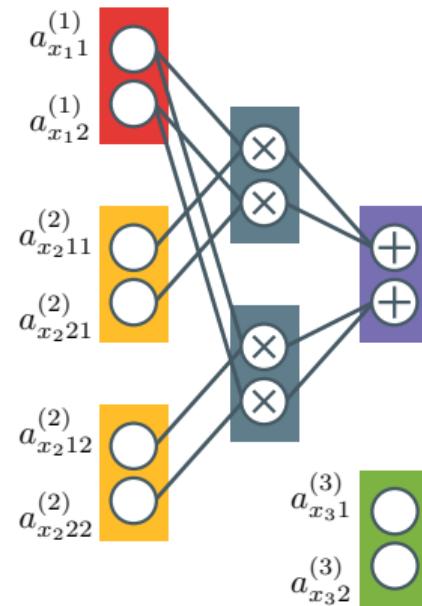


Tensor trains are circuits

also called matrix-product states

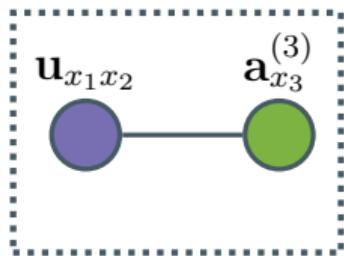


$$t_{x_1 x_2 x_3} = \sum_{r_2=1}^R u_{x_1 x_2 r_2} a_{r_2 x_3}^{(3)}$$

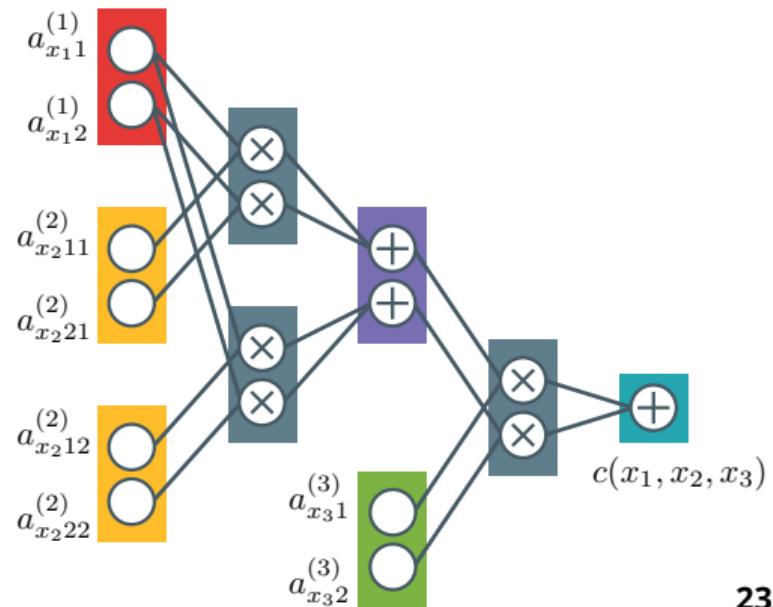


Tensor trains are circuits

also called matrix-product states



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



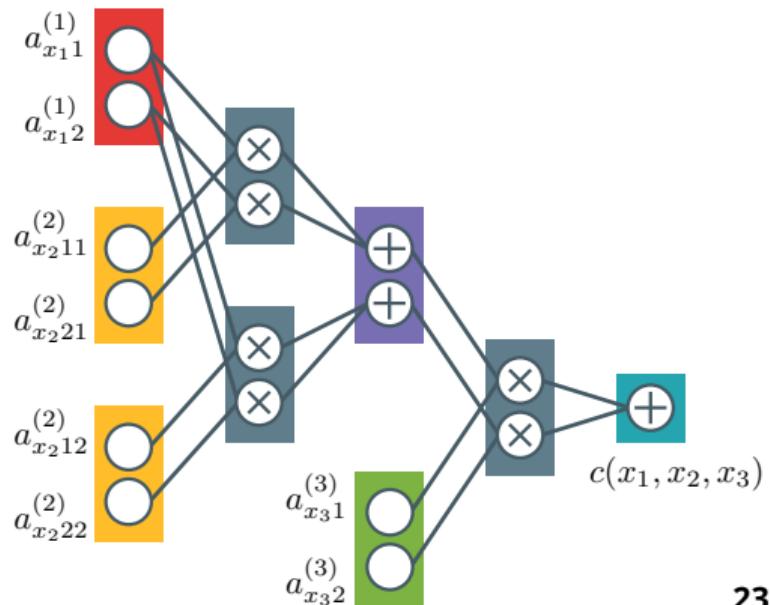
Tensor trains are circuits

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

:

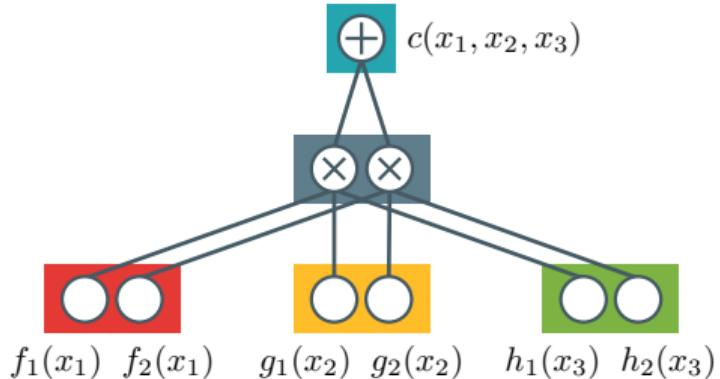
“What do we gain from circuits?”

More input functions with circuits

Input unit functions $f(x)$ compute:

- an entry of matrix (or tensor):

$$f(x) = v_{xr} \quad \text{or} \quad f(x) = v_{r_1 x r_2}$$



More input functions with circuits

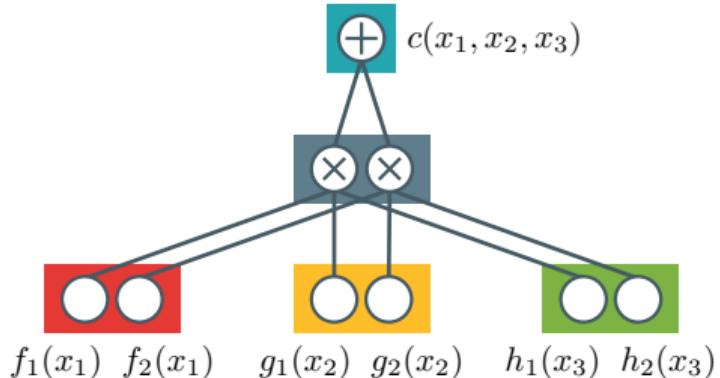
Input unit functions $f(x)$ compute:

- an entry of matrix (or tensor):

$$f(x) = v_{xr} \quad \text{or} \quad f(x) = v_{r_1 x r_2}$$

- probability mass functions:

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



More input functions with circuits

Input unit functions $f(x)$ compute:

- an entry of matrix (or tensor):

$$f(x) = v_{xr} \quad \text{or} \quad f(x) = v_{r_1 x r_2}$$

- probability mass functions:

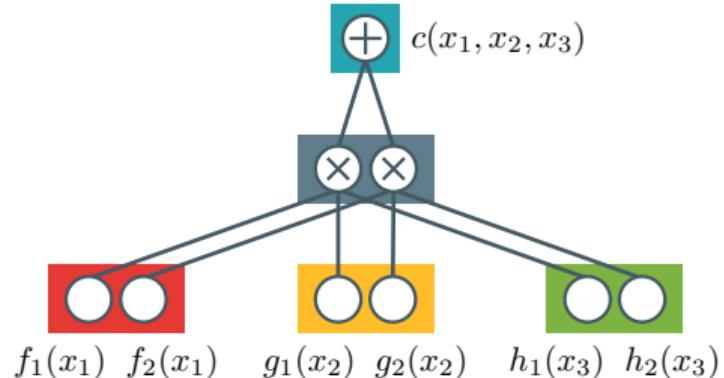
$$f(x) = \text{Binomial}(x; n, p) \quad (\text{more compact!})$$

- continuous functions:

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

$$f(x) = \text{Normal}(x; \mu, \sigma^2)$$

⇒ infinite-dimensional tensors (or functions)



Probabilistic circuits (PCs)

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

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

Probabilistic circuits (PCs)

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

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

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

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

Probabilistic circuits (PCs)

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

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

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

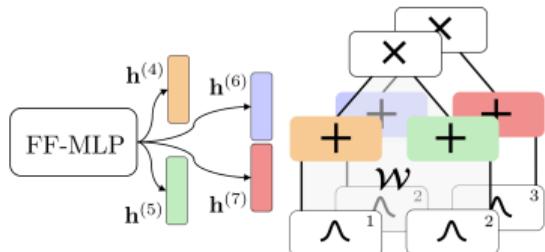
where $Z = \sum_{\mathbf{x}} c(\mathbf{x})$ (PMF) or $Z = \int c(\mathbf{x}) d\mathbf{x}$ (PDF)

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

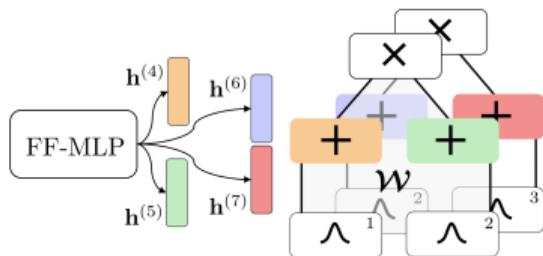
Gala et al., "Scaling Continuous Latent Variable Models as Probabilistic Integral Circuits", 2024

Shao et al., "Conditional sum-product networks: Imposing structure on deep probabilistic architectures", 2020

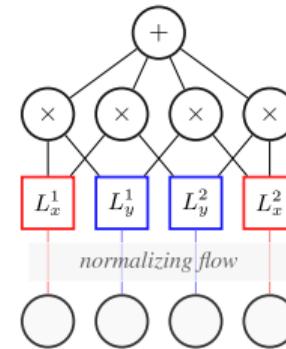
Sidheekh, Kersting, and Natarajan, "Probabilistic Flow Circuits: Towards Unified Deep Models for Tractable Probabilistic Inference", 2023

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



learning & reasoning with circuits in pytorch

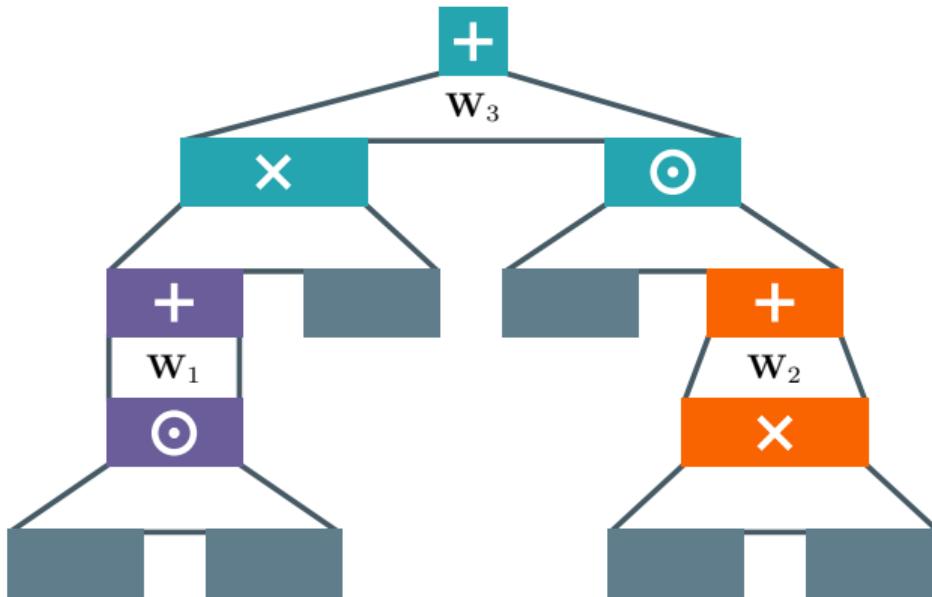
<https://github.com/april-tools/cirkit>

```
1 from cirkit.symbolic.layers import (
2     EmbeddingLayer, SumLayer, KroneckerLayer,
3     Scope
4 )
5
6 # Tensor shape and rank
7 shape = (3, 1280, 720)
8 rank = 42
9
10 # Construct the Tucker factorization layers
11 v1 = EmbeddingLayer(Scope([0]), rank, num_states=shape[0])
12 v2 = EmbeddingLayer(Scope([1]), rank, num_states=shape[1])
13 v3 = EmbeddingLayer(Scope([2]), rank, num_states=shape[2])
14 kron = KroneckerLayer(rank, arity=3)
15 tucker = SumLayer(rank ** 3, num_output_units=1)
```

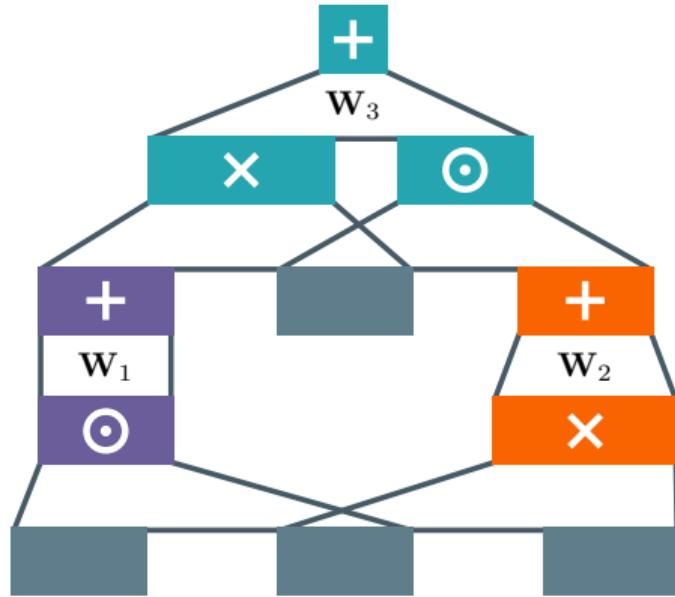
```
17  
18 # Construct the Tucker circuit  
19 circuit = Circuit(  
20     layers=[v1, v2, v3, kron, tucker],  
21     in_layers={ # The layers input connections  
22         kron: [v1, v2, v3],  
23         tucker: [kron]  
24     },  
25     outputs=[tucker]  
26 )
```

```
1 # Compile the circuit to PyTorch code
2 from cirkit.pipeline import compile
3 pth_circuit = compile(circuit)
4
5 print(pth_circuit) # Tucker factorization
6 # TorchCircuit(
7 #   (0): TorchEmbeddingLayer(...)
8 #   (1): TorchEmbeddingLayer(...)
9 #   (2): TorchEmbeddingLayer(...)
10 #  (3): TorchKroneckerLayer(...)
11 #  (4): TorchSumLayer(...)
12 #)
13
14 # Compute one entry of the encoded tensor
15 x = torch.tensor([[1, 500, 300]])
16 t_x = pth_circuit(x)
```

```
1 from cirkit.templates import tensor_factorizations  
2  
3 shape = (3, 1280, 720)  
4  
5 # CP factorization  
6 circuit = tensor_factorizations.cp(shape, rank=42)  
7  
8 # Tucker factorization  
9 circuit = tensor_factorizations.tucker(shape, rank=42)  
10  
11 # Tensor-train / matrix-product state  
12 circuit = tensor_factorizations.tensor_train(shape, rank=42)
```



Stack layers to build a deep factorization!



Save computation by sharing sub-factorizations!

```
1 from cirkit.symbolic.layers import (
2     EmbeddingLayer, SumLayer, KroneckerLayer,
3     HadamardLayer, Scope)
4
5 # Tensor shape and ranks
6 shape = (17, 3, 1280, 720)
7 rank1, rank2 = 2, 4
8
9 # Construct the layers
10 v1 = EmbeddingLayer(Scope([0]), rank1, num_states=shape[0])
11 v2 = EmbeddingLayer(Scope([1]), rank1, num_states=shape[1])
12 v3 = EmbeddingLayer(Scope([2]), rank1, num_states=shape[2])
13 v4 = EmbeddingLayer(Scope([3]), rank1, num_states=shape[3])
14 kron1 = KroneckerLayer(rank1, arity=2)
15 kron2 = KroneckerLayer(rank1, arity=2)
16 hada1 = HadamardLayer(rank1, arity=2)
```

```
17 hada2 = HadamardLayer(rank1, arity=2)
18 sum1 = SumLayer(rank1, num_output_units=rank1)
19 sum2 = SumLayer(rank1 ** 2, num_output_units=rank1)
20 sum3 = SumLayer(rank1 + rank2, num_output_units=1, arity=2)
21
22 # Construct the "Frankenstein" circuit
23 circuit = Circuit(
24     layers=[v1, v2, v3, v4, kron1, kron2, ...],
25     in_layers={ # The layers input connections
26         hada1: [v1, v2],
27         kron1: [v2, v3],
28         sum1: [hada1],
29         sum2: [kron1],
30         kron2: [sum1, v4],
31         ...
32     }
33 )
34
35 circuit
```

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)*

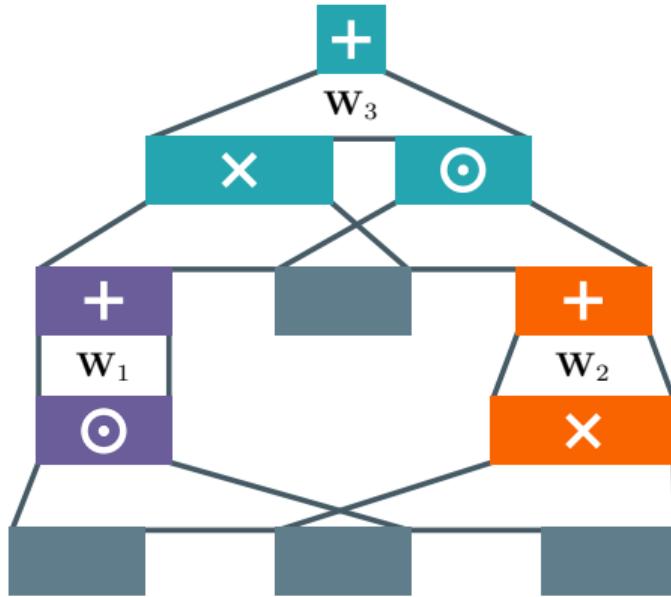
Takeaways

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



...but is it always useful?

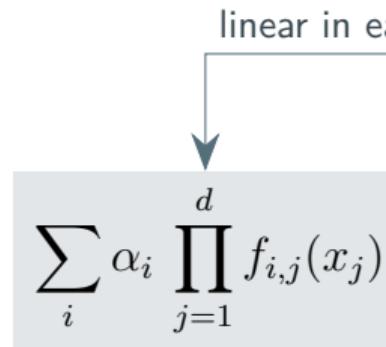
tl;dr

***“Understand when and how
we can build a deep circuit
that is a deep factorization”***

Multilinear forms

tensor factorizations are typically multilinear

linear in each univariate basis/input function


$$\sum_i \alpha_i \prod_{j=1}^d f_{i,j}(x_j) \quad \left\{ \begin{array}{l} t_{x_1 \dots x_n} = \sum_{r=1}^R \prod_{j=1}^d v_{x_j r}^{(j)} \quad (\text{CP}) \\ t_{x_1 \dots x_n} = \sum_{r_1, \dots, r_d=1}^{R_1, \dots, R_d} w_{r_1 \dots r_d} \prod_{j=1}^d v_{x_j r_j}^{(j)} \quad (\text{Tucker}) \end{array} \right.$$

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

Structural properties

smoothness

decomposability

compatibility

determinism

Structural properties

property A

property B

property C

structural properties

smoothness

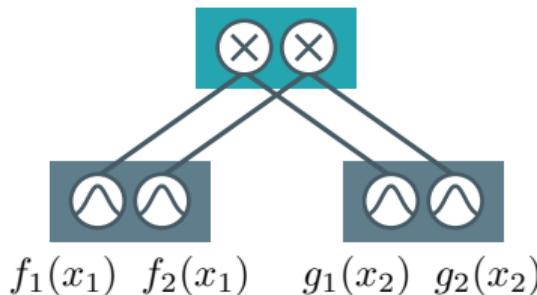
decomposability

property C

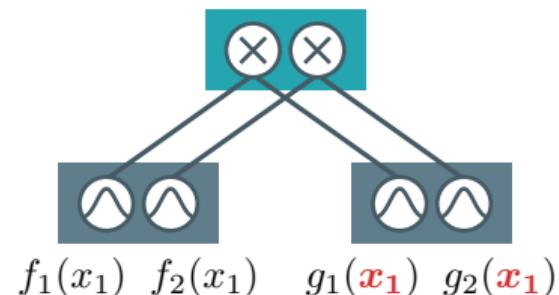
smoothness \wedge decomposability
 \implies multilinearity

Multilinearity in circuits

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



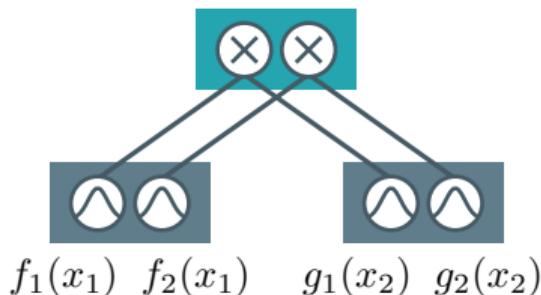
✓ multilinear



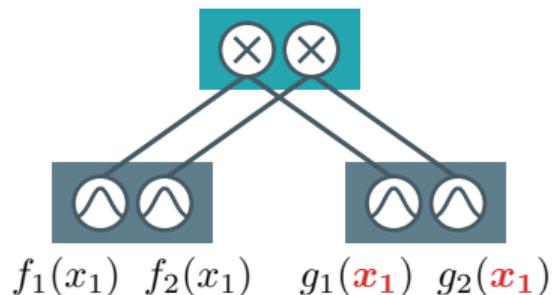
✗ not multilinear

Multilinearity in circuits

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



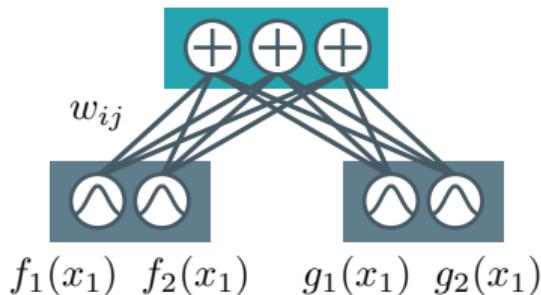
decomposable circuit



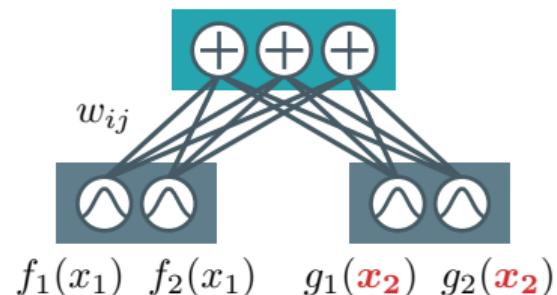
non-decomposable circuit

Multilinearity in circuits

the inputs of sum units are defined over the same variables



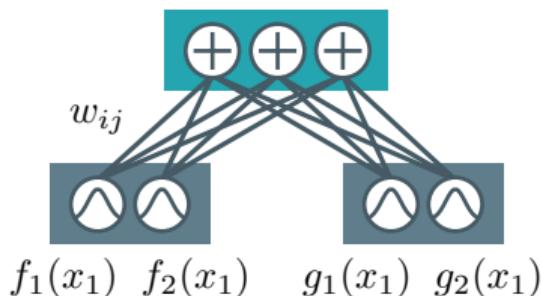
✓ multilinear



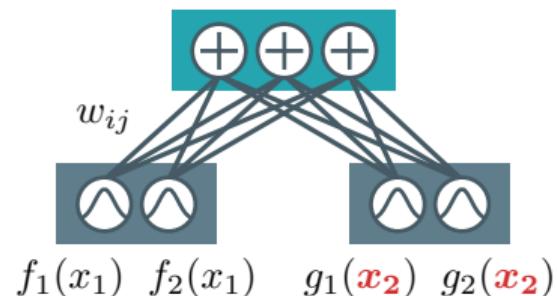
✗ not multilinear

Multilinearity in circuits

the inputs of sum units are defined over the same variables



smooth circuit



non-smooth circuit

Structural properties

smoothness

decomposability

property C

smoothness \wedge decomposability
 \implies multilinearity

Structural properties

smoothness

tractable computation of arbitrary integrals
in probabilistic circuits

decomposability

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

property C

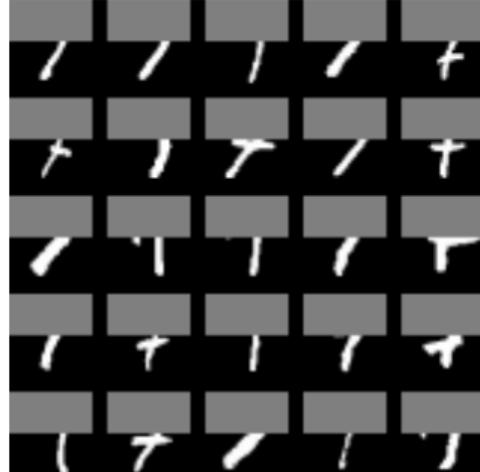
⇒ tractable partition function
⇒ also any conditional is tractable

tractable marginals on PCs

Original

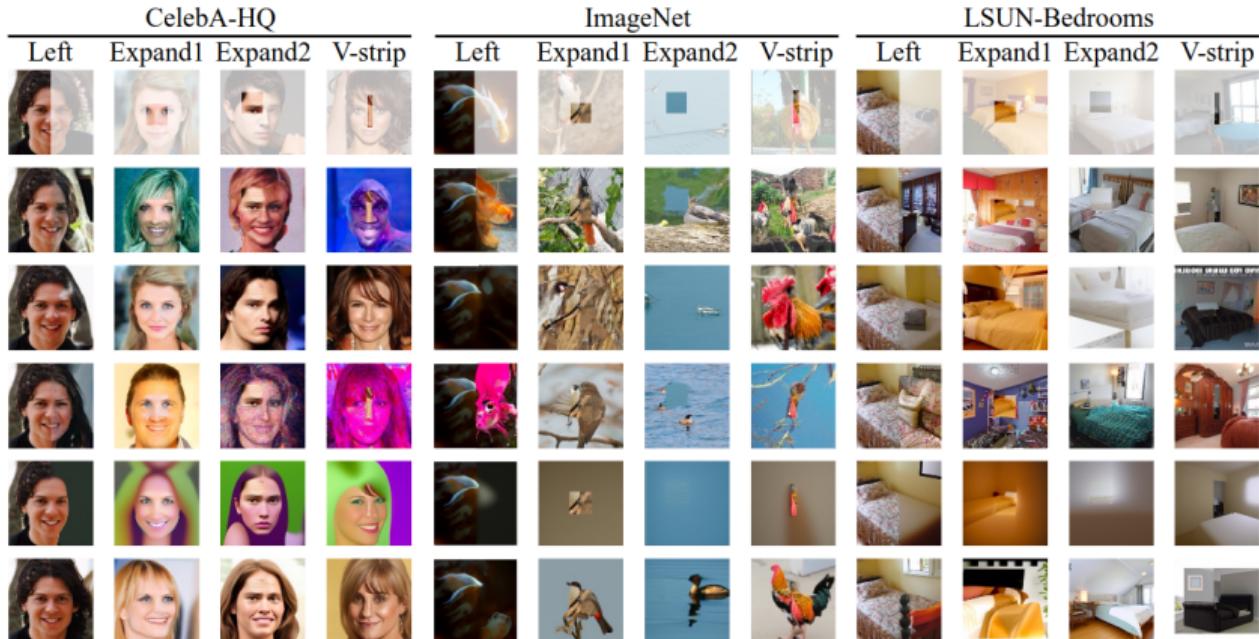


Missing



Conditional sample





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

compute arbitrary summations (or integrals)
 \implies linear time in circuit size!

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

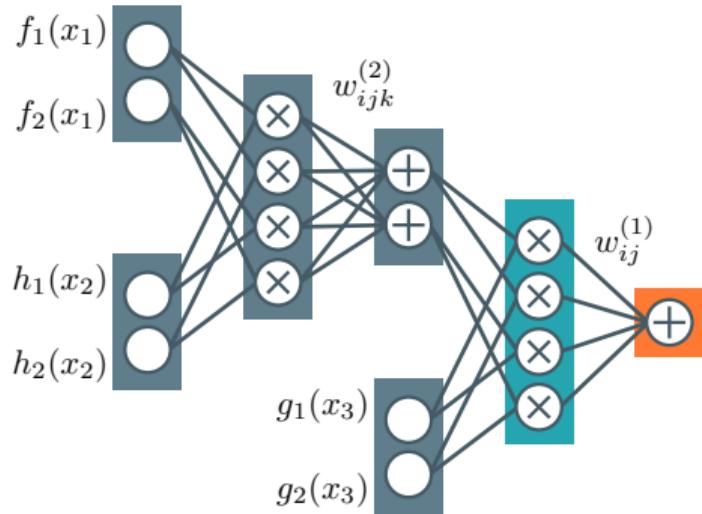
or $\int c(\mathbf{x}) d\mathbf{x}$ in the continuous case

smooth + decomposable circuits = ...

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

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

⇒ integrals are “pushed down” to the inputs



smooth + decomposable circuits = ...

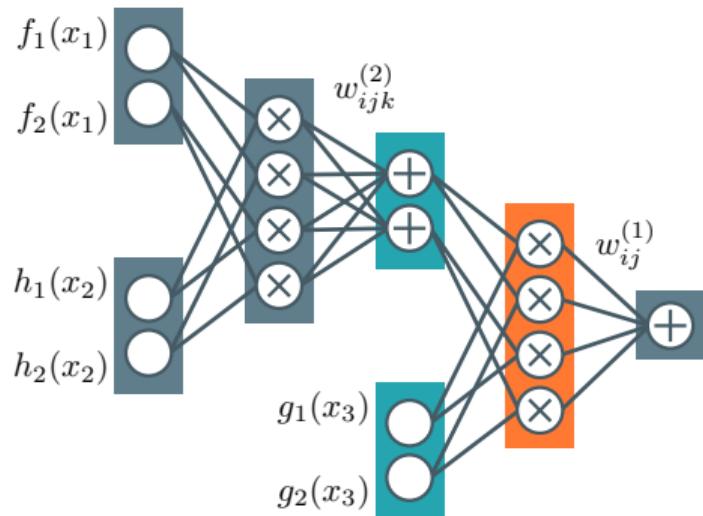
If $c(\mathbf{x}) = c_1(\mathbf{y}) \quad c_2(\mathbf{z})$

(decomposability):

$$\int c(\mathbf{x}) \, d\mathbf{x} = \int \int c_1(\mathbf{y}) \, c_2(\mathbf{z}) \, dy \, dz$$

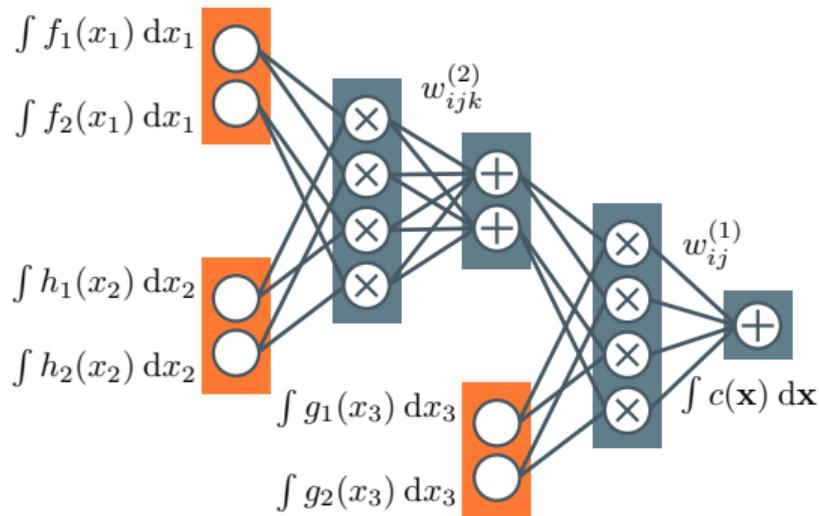
$$= \left(\int c_1(\mathbf{y}) \, dy \right) \left(\int c_2(\mathbf{z}) \, dz \right)$$

⇒ integrals “decompose” into easier ones



smooth + decomposable circuits = ...

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



“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 $_{\mathbb{R} \geq 0}$ (Glasser et al., 2019)

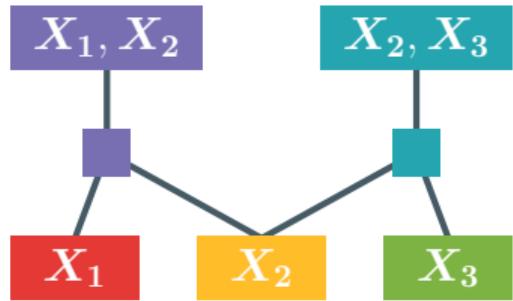
BM (Han et al., 2018)

TTDE (Novikov et al., 2021)

NPC² (Loconte et al., 2024)

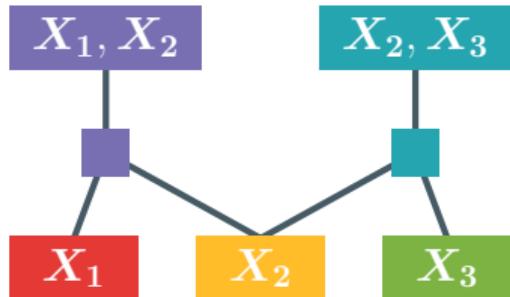
TTN (Cheng et al., 2019)

Building smooth & decomposable circuits

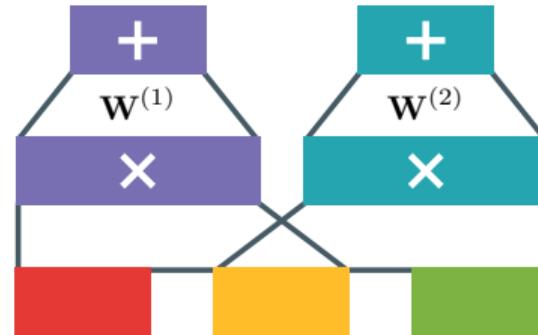


1) choose a template

Building smooth & decomposable circuits



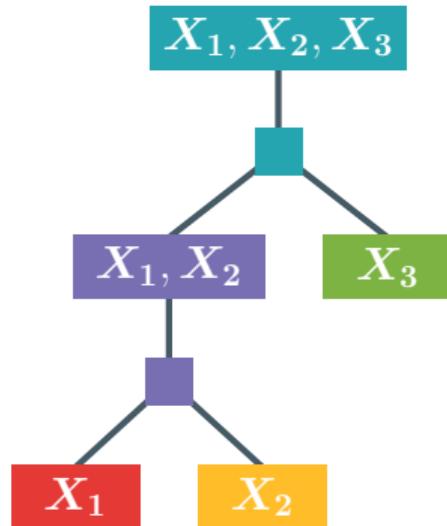
1) choose a template



2) pick a layer to parameterize
the chosen template

Region graphs

A bipartite graph to build smooth and decomposable circuits:

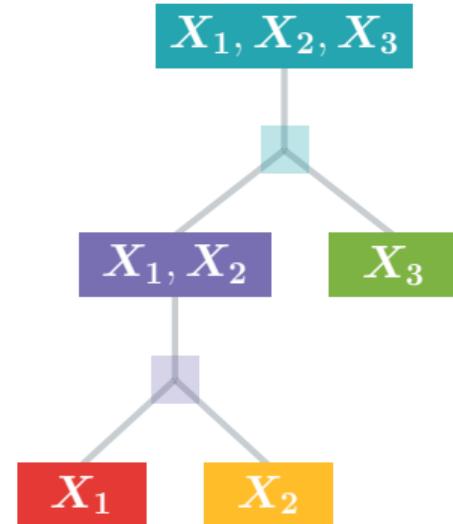


Region graphs

A bipartite graph to build smooth and decomposable circuits:

Region node:

set of variables (or dimensions)



Region graphs

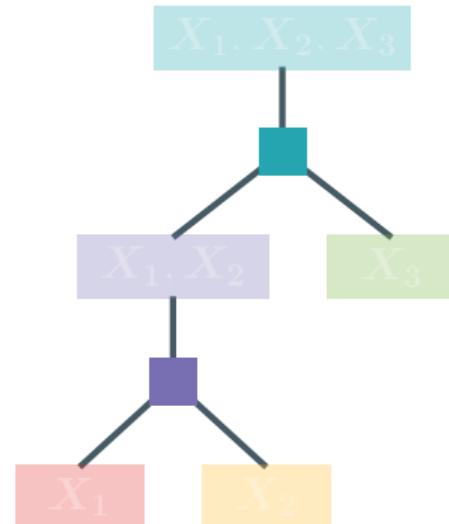
A bipartite graph to build smooth and decomposable circuits:

Region node:

set of variables (or dimensions)

Partition node:

decomposition of a region



Region graphs

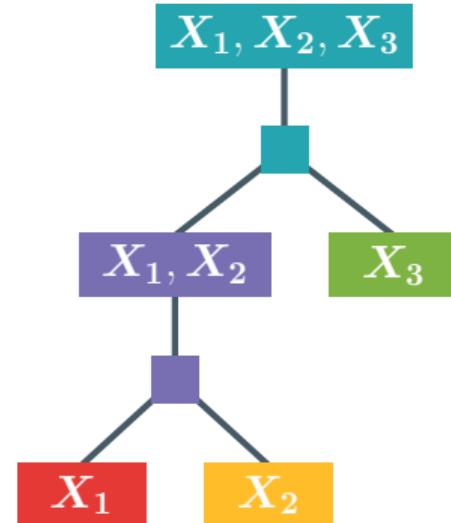
A bipartite graph to build smooth and decomposable circuits:

Region node:

set of variables (or dimensions)

Partition node:

decomposition of a region



Region graphs

A bipartite graph to build smooth and decomposable circuits:

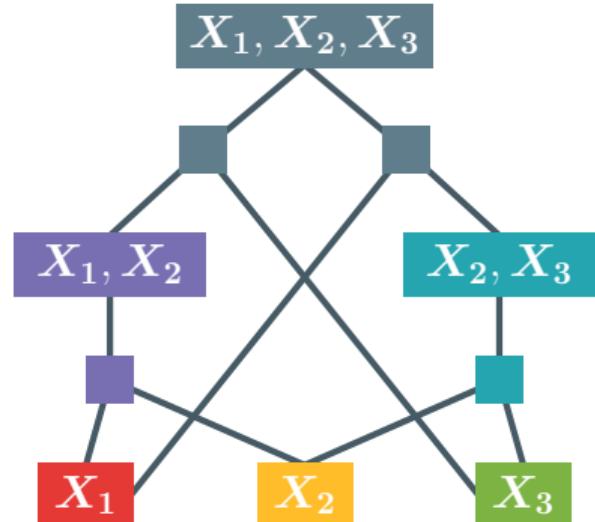
Region node:

set of variables (or dimensions)

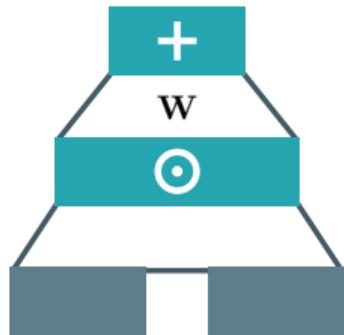
Partition node:

decomposition of a region

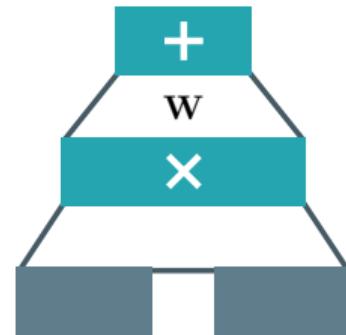
⇒ generalizes mode cluster trees



Circuit sum-product layers as factorizations

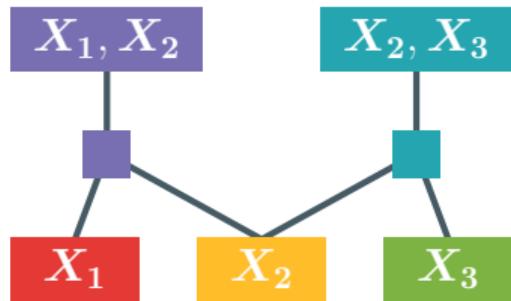


CP layer



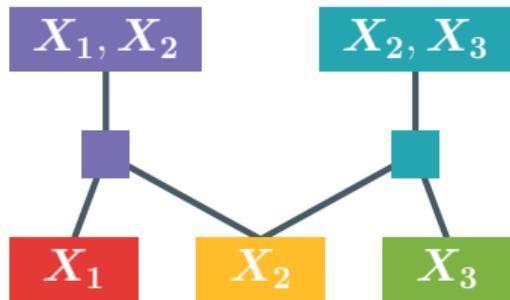
Tucker layer

From region graphs to circuits

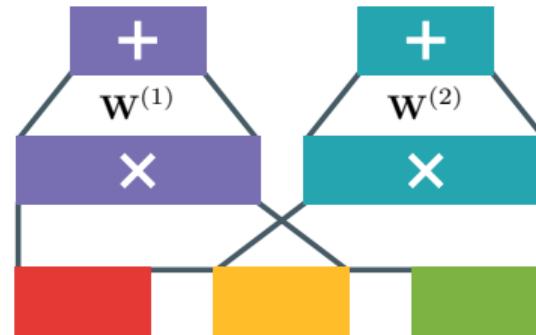


1) choose a region graph

From region graphs to circuits

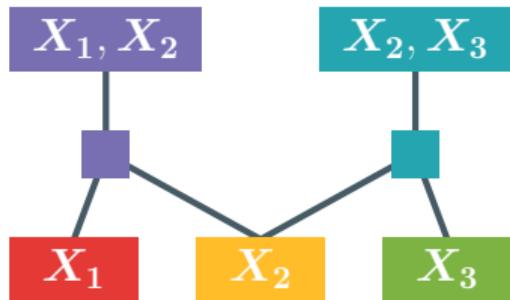


1) choose a region graph

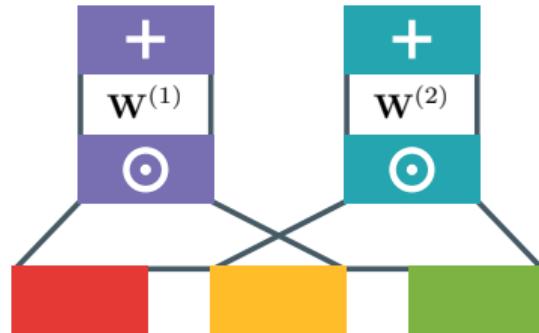


2) pick layer and number of units
(e.g., Tucker layer)

From region graphs to circuits



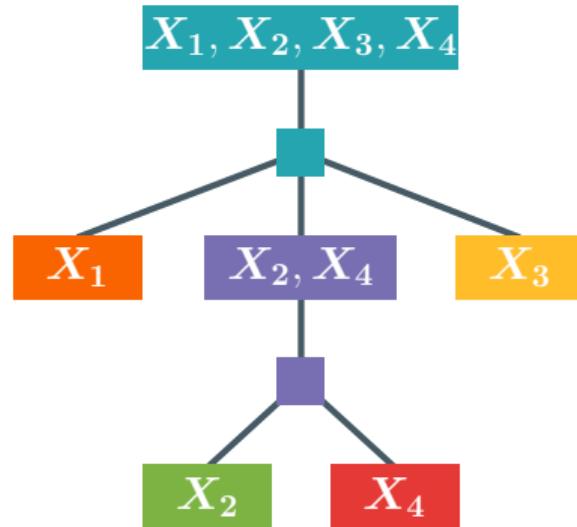
1) choose a region graph



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

Which region graph?

learned or randomized trees

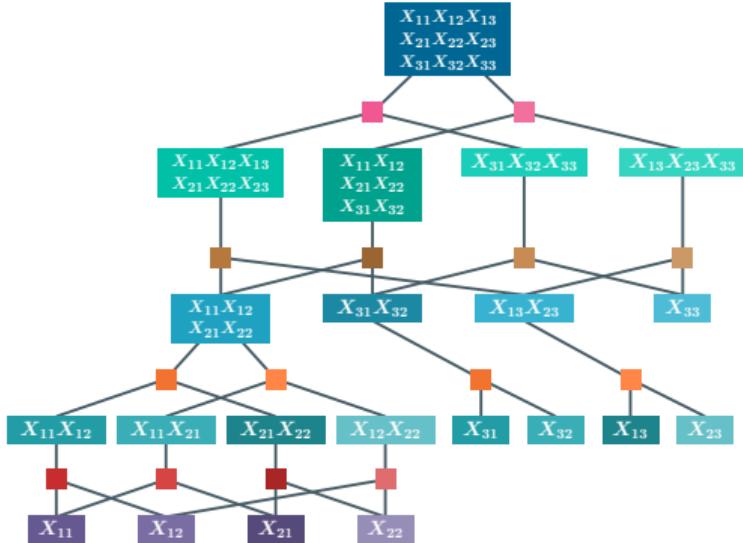


Peharz et al., "Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning", 2020

Liu and Broeck, "Tractable Regularization of Probabilistic Circuits", 2021

Which region graph?

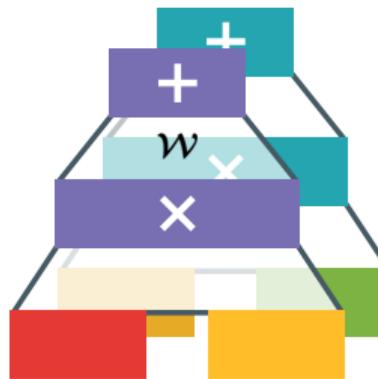
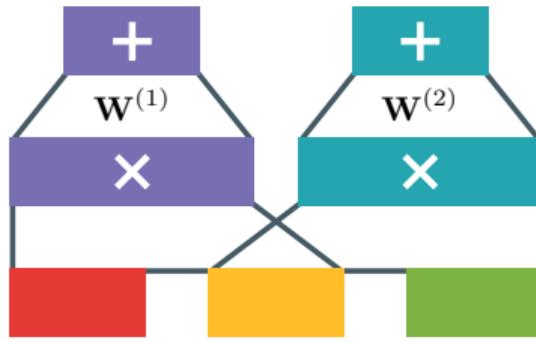
image-tailored graphs



folding

to speed-up inference

parallelize layers that can be evaluated independently



A unifying circuit construction pipeline

PC ARCHITECTURE	REGION GRAPH	SUM-PRODUCT LAYER	FOLD
Poon&Domingos (Poon & Domingos, 2011)	PD	CP^\top	✗
RAT-SPN (Peharz et al., 2020c)	RND	Tucker	✗
EiNet (Peharz et al., 2020a)	{ RND, PD }	Tucker	✓
HCLT (Liu & Van den Broeck, 2021b)	CL	CP^\top	✓
HMM/MPS $_{\mathbb{R} \geq 0}$ (Glasser et al., 2019)	LT	CP^\top	✗
BM (Han et al., 2018)	LT	CP^\top	✗
TTDE (Novikov et al., 2021)	LT	CP^\top	✗
NPC ² (Loconte et al., 2024)	{ LT, RND }	{ CP^\top , Tucker }	✓
TTN (Cheng et al., 2019)	QT-2	Tucker	✗
Mix & Match (our pipeline)	$\left\{ \begin{array}{l} \text{RND}, \text{PD}, \text{LT}, \\ \text{CL}, \text{QG}, \text{QT-2}, \text{QT-4} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{Tucker}, \text{CP}, \text{CP}^\top \\ \text{CP}^S, \text{CP}^{XS} \mid \text{FOLD } \checkmark \end{array} \right\} \cup \left\{ \begin{array}{l} \text{CP}^S, \text{CP}^{XS} \mid \text{FOLD } \checkmark \end{array} \right\}$	$\times \{ \text{✗, ✓} \}$

enrich the pipeline

with new layers

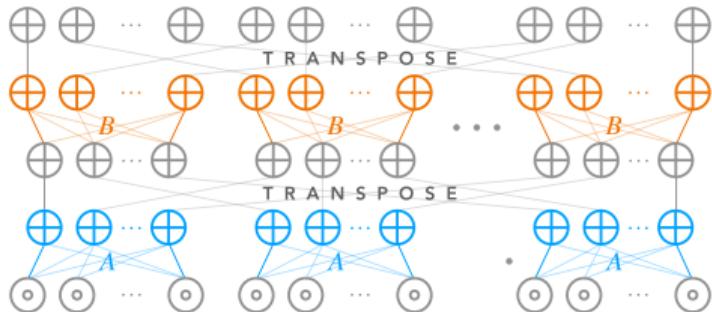
Monarch matrix factorization:

$$\mathbf{W}_M = \mathbf{P}_L \mathbf{L} \mathbf{P}_R \mathbf{R}$$

Monarch circuit layer:

$$\ell(\mathbf{x}) = \mathbf{W}_M \ell_i(\mathbf{x})$$

**More tensor factorizations
with new layers!**



```
1 # Construct a region graph
2 from cirkit.templates.region_graph import (
3     RandomBinaryTree) # or QuadGraph, LinearTree, ...
4 region_graph = RandomBinaryTree(num_variables=10)
5
6 # Build the circuit from the region graph
7 from cirkit.symbolic.layers import EmbeddingLayer
8 circuit = region_graph.build_circuit(
9     sum_product='tucker',           # or 'cp'
10    input_factory=EmbeddingLayer  # or GaussianLayer, ...
11    num_sum_units=32,
12    num_input_units=32)
13
14 # Compile the circuit to PyTorch
15 from cirkit.pipeline import compile
16 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



q₁

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

q₂

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

q₃

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

Reasoning about ML models



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

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

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

...in the language of probabilities

Reasoning about ML models



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

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

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

hard to compute in general!

Reasoning about ML models



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

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

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

it is crucial we compute them exactly and in polytime!

Which structural properties

for complex reasoning



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

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

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

smooth + decomposable

Which structural properties

for complex reasoning



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

smooth + decomposable

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

???????

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

???????

Which properties for expectations?

smoothness

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

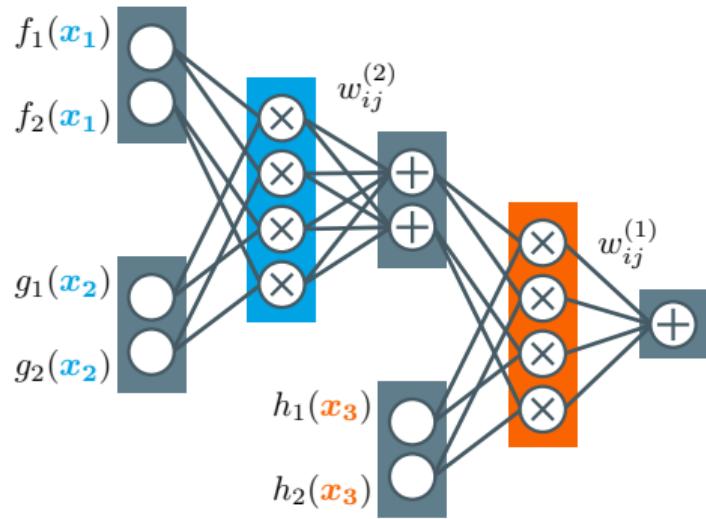
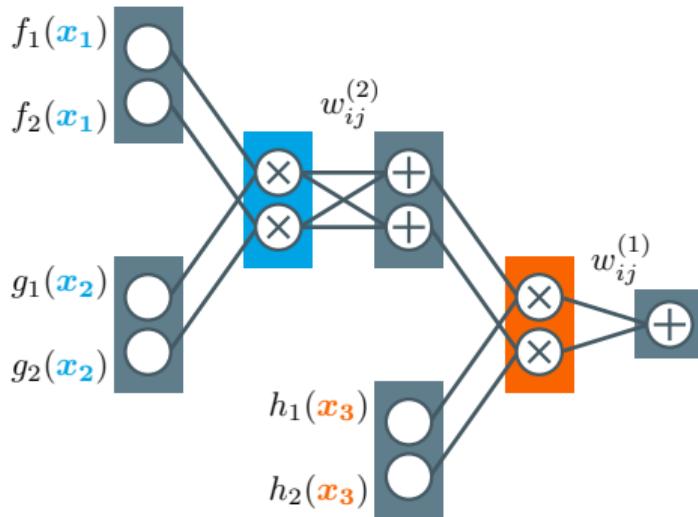
decomposability

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

compatibility

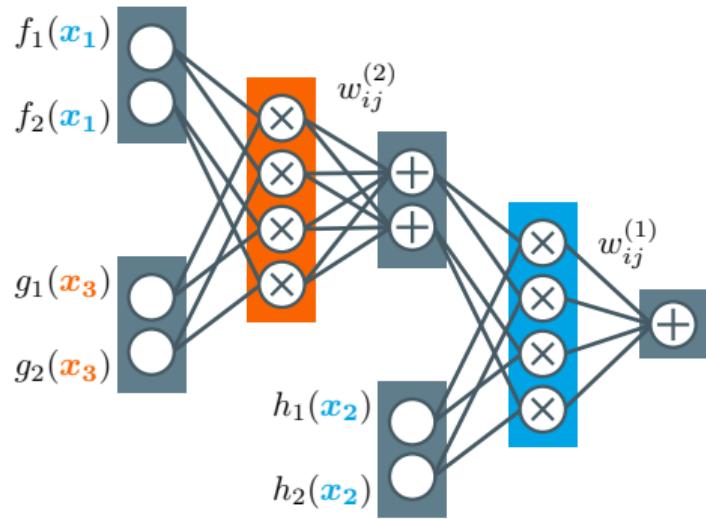
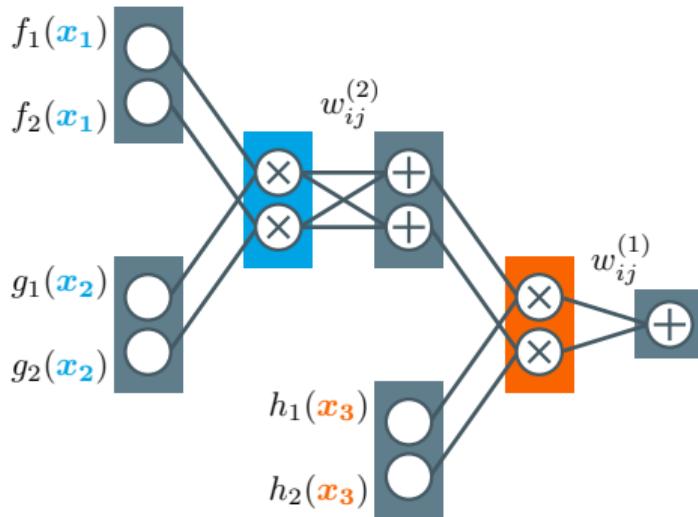
when both $p(\mathbf{x})$ and $f(\mathbf{x})$ are circuits

Compatibility



Compatible circuits

Compatibility



non-compatible circuits

Structural properties

smoothness

compatibility

decomposability

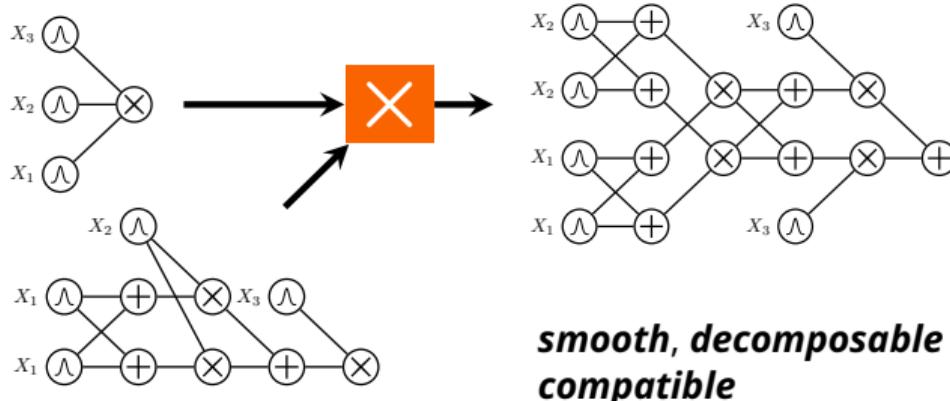


smoothness \wedge decomposability

compatibility

compatibility \Rightarrow tractable expectations

Tractable products



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

```
1 from cirkit.symbolic.circuit import Circuit
2 from cirkit.symbolic.functional import (
3     integrate, multiply)
4
5 # Circuits expectation  $\int [p(x) f(x)] dx$ 
6 def expectation(p: Circuit, f: Circuit) -> Circuit:
7     i = multiply(p, f)
8     return integrate(i)
9
10 # Squared loss  $\int [p(x)-q(x)]^2 dx = E_p[p] + E_q[q] - 2E_p[q]$ 
11 #           =  $\int p^2(x) dx + \int q^2(x) dx - 2\int p(x)q(x) dx$ 
12 def squared_loss(p: Circuit, q: Circuit) -> Circuit:
13     p2 = multiply(p, p)
14     q2 = multiply(q, q)
15     pq = multiply(p, q)
16     return integrate(p2) + integrate(q2) - 2 * integrate(pq)
```

Which structural properties

for complex reasoning



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

smooth + decomposability

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

compatibility

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

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)



q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(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**:

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

⇒ can be slow for large dimensions, requires **inverting the CDF**

approximate inference

e.g., via sampling

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

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

⇒ can be slow for large dimensions, requires **inverting the CDF**

can we do better?

approximate inference

e.g., via sampling

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

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

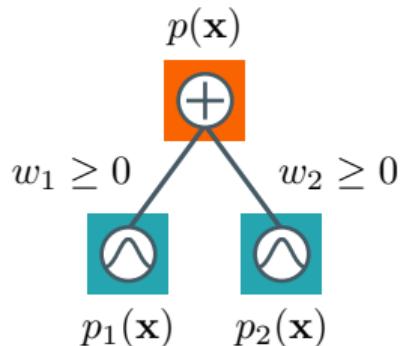
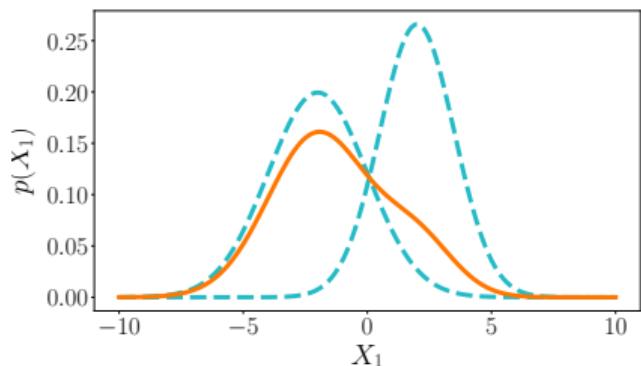
⇒ can be slow for large dimensions, requires **inverting the CDF**

can we do better?

⇒ yes, for non-negative factorizations/monotonic PCs

How to sample?

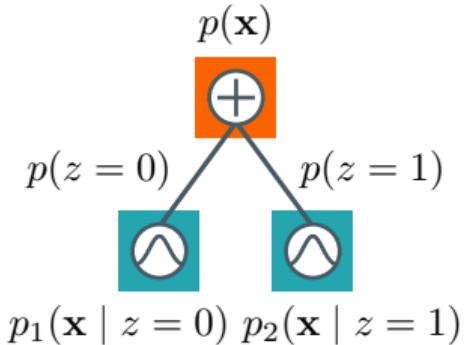
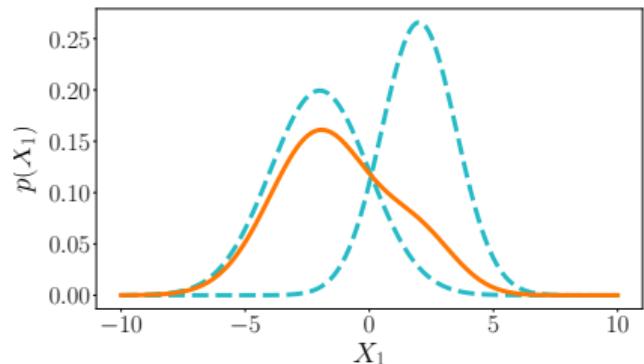
non-negative factorizations as latent-variable models



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

How to sample?

non-negative factorizations as latent-variable models



$$\begin{aligned} p(\mathbf{x}) &= p(z = 0) \ p_1(\mathbf{x} \mid z = 0) \\ &\quad + p(z = 1) \ p_2(\mathbf{x} \mid z = 1) \end{aligned}$$

Structural properties

smoothness

decomposability

compatibility

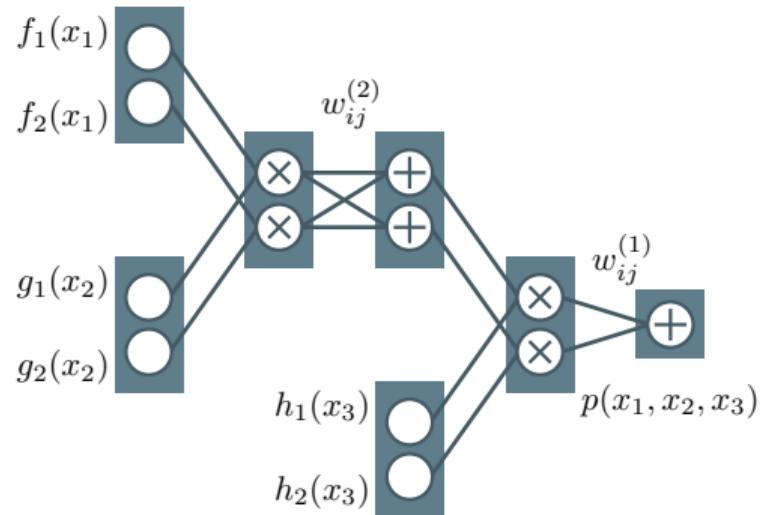
sampling in a single backward pass

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

⇒ exact sampling method

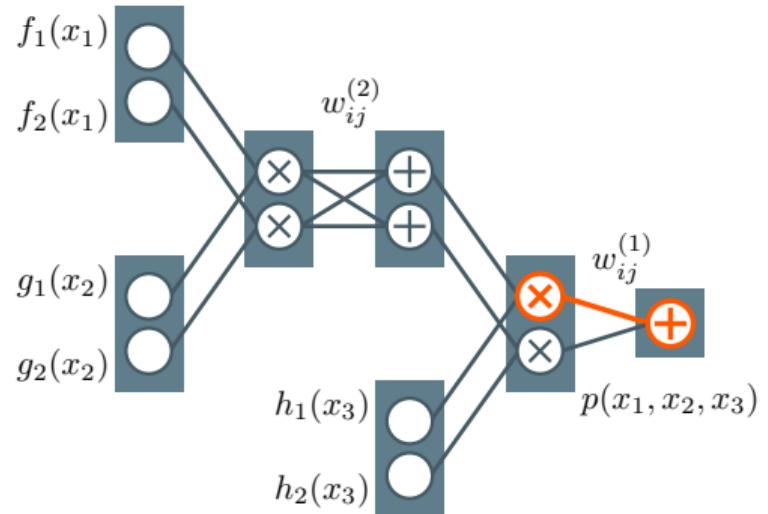
smooth + decomposable circuits = ...

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



smooth + decomposable circuits = ...

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

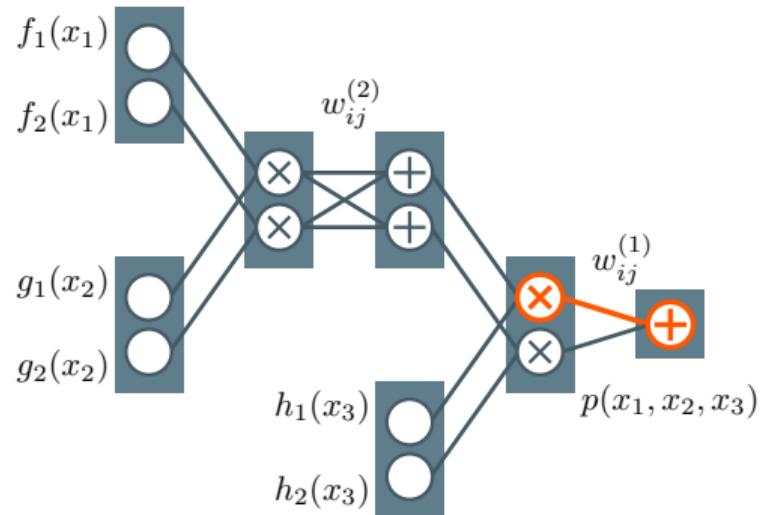


***smooth* + *decomposable* circuits = ...**

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 $p_i(\mathbf{x} \mid z = i)$

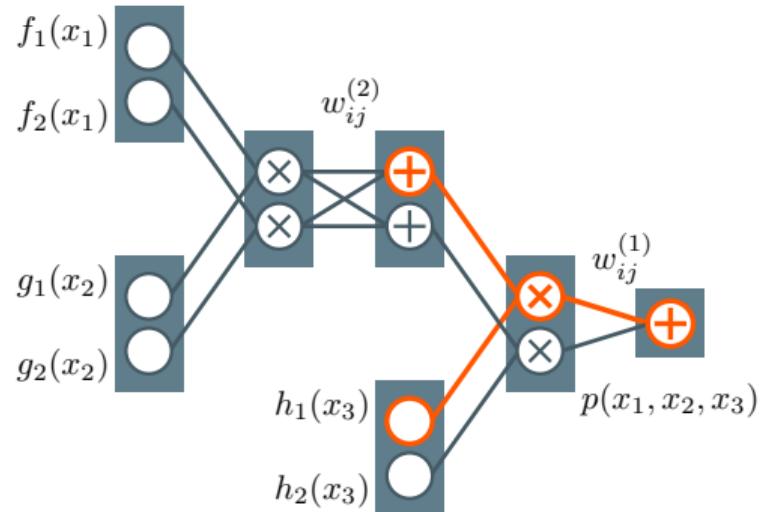


smooth + decomposable circuits = ...

If $p(\mathbf{x}) = p_1(\mathbf{y}) \ p_2(\mathbf{z})$

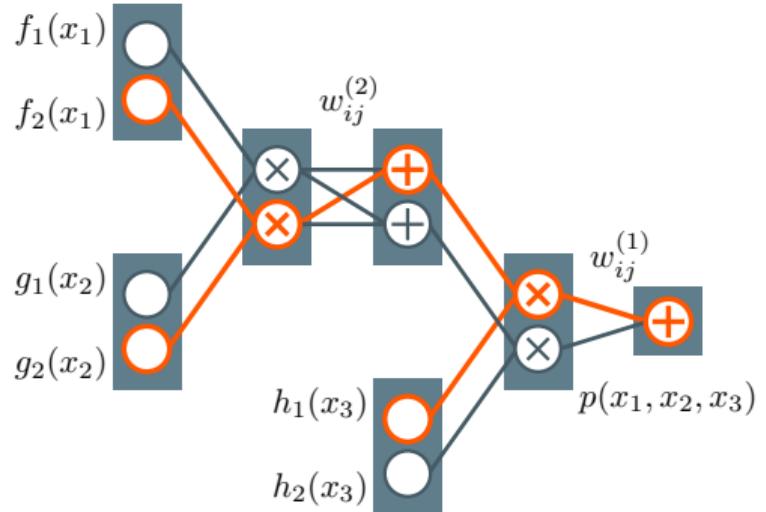
(decomposability):

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



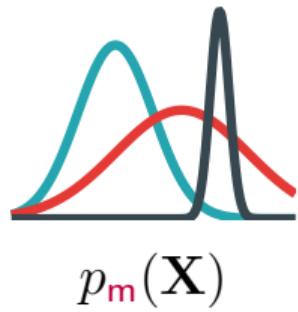
smooth + decomposable circuits = ...

Sample from simple input distributions:
⇒ easy for Categorical, Gaussian, ...





$q_1(m) ?$
 $q_2(m) ?$
...
 $q_k(m) ?$



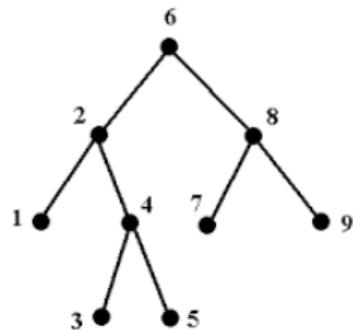
\approx

	X^1	X^2	X^3	X^4	X^5
x_8					
x_7					
x_6					
x_5					
x_4					
x_3					
x_2					
x_1					

generative models that can reason probabilistically

...but some events are certain!

When?



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

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

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

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

hierarchical multi-label classification

When?



Ground Truth

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

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

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!



Ground Truth



ResNet-18



Semantic Loss



circuits

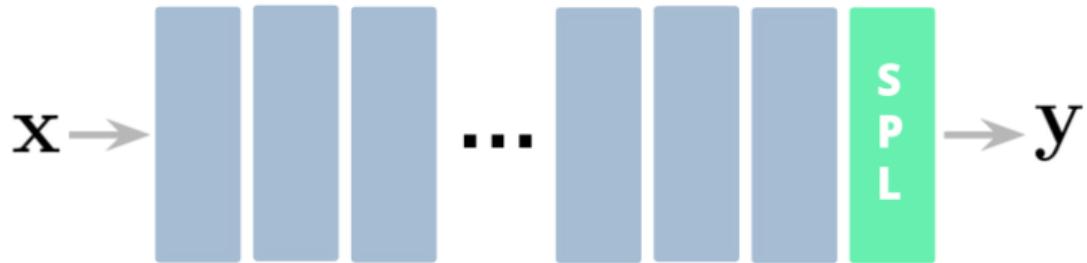
you can predict valid paths 100% of the time!

How?

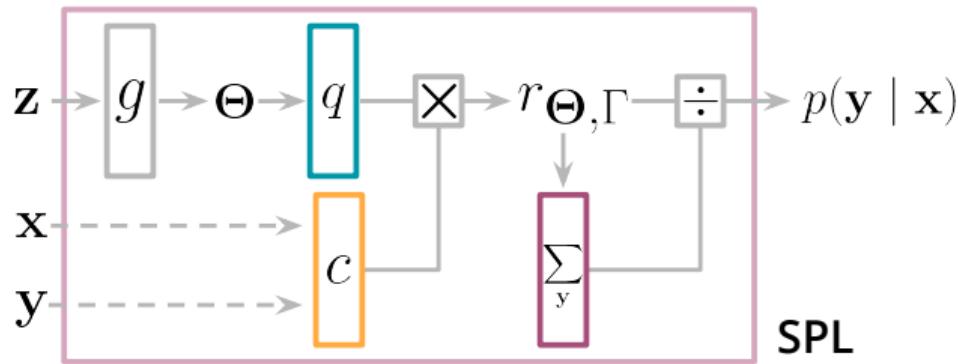


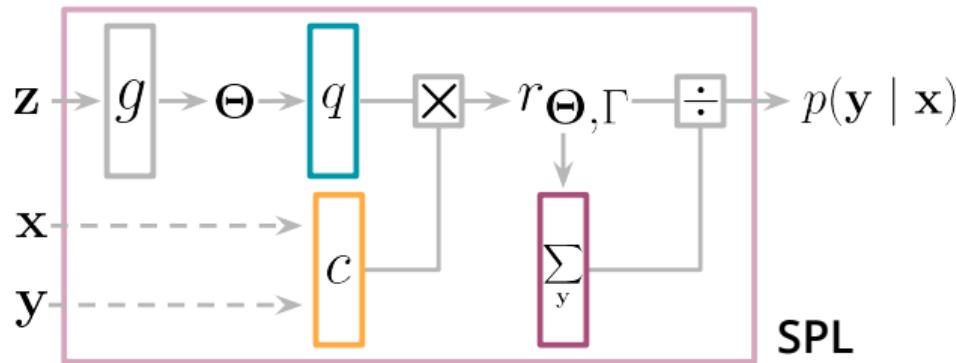
take an unreliable neural network architecture...

How?



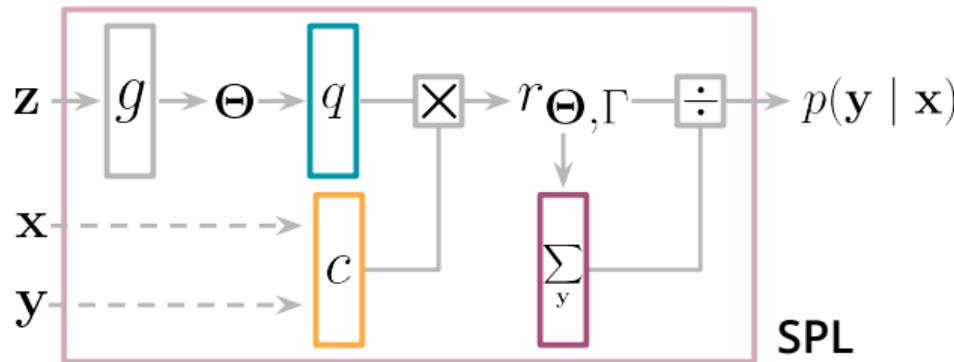
*.....and replace the last layer with
a semantic probabilistic layer (SPL)*





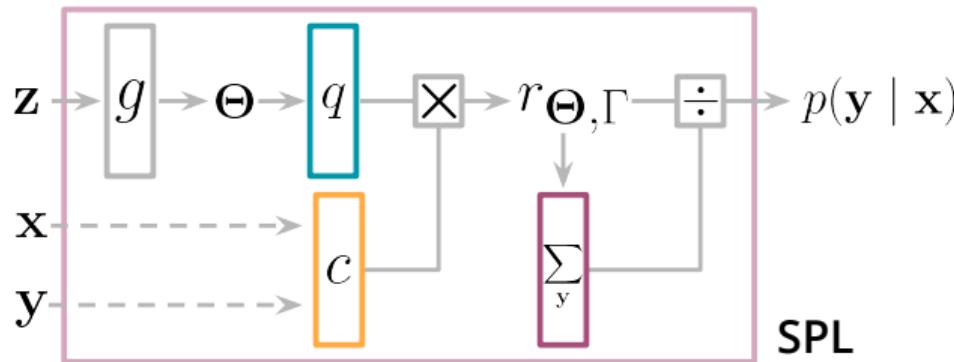
$$p(\mathbf{y} \mid \mathbf{x}) = \mathbf{q}_{\Theta}(\mathbf{y} \mid g(\mathbf{z}))$$

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



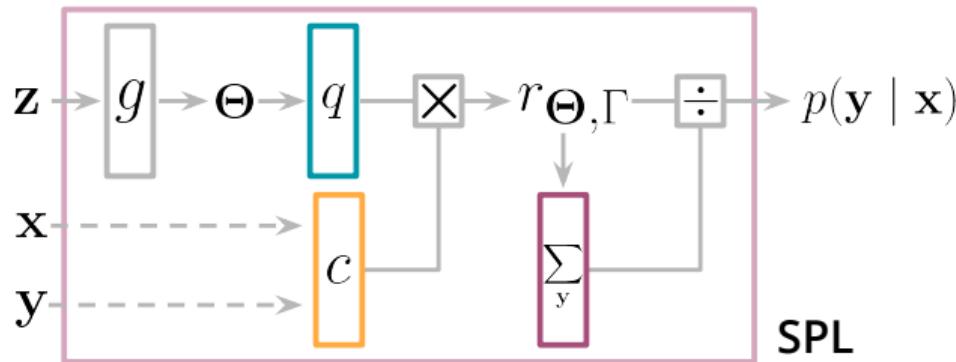
$$p(\mathbf{y} \mid \mathbf{x}) = \mathbf{q}_{\Theta}(\mathbf{y} \mid g(\mathbf{z})) \cdot \mathbf{c}_K(\mathbf{x}, \mathbf{y})$$

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



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

a product of experts : (



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

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

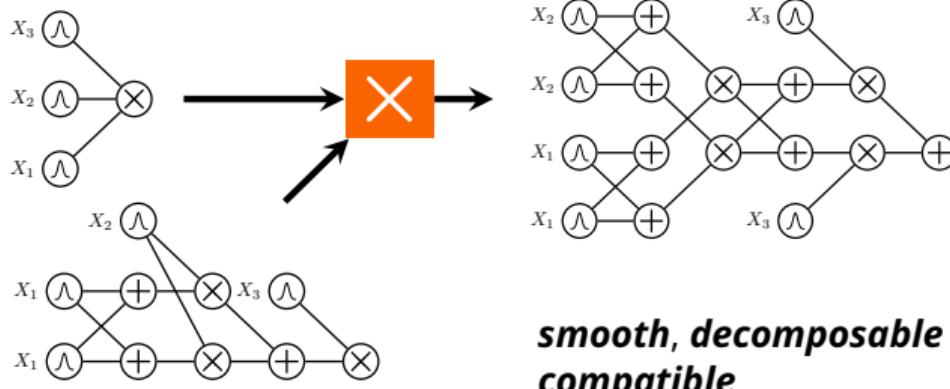
Goal

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

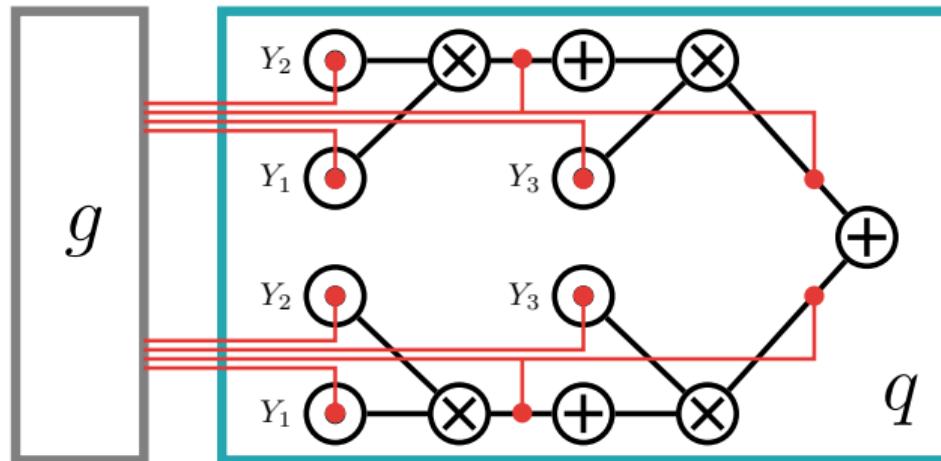
Goal

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

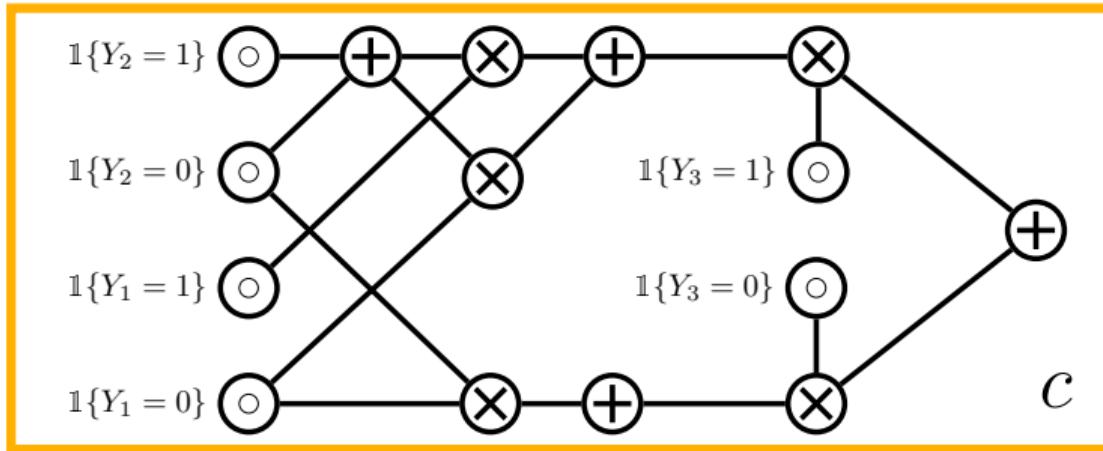
Tractable products



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



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



and a logical circuit $c(y, x)$ encoding K

knowledge compilation

(as a Boolean tensor factorization)

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

$$\wedge (Y_2 = 1 \implies Y_3 = 1)$$

$$\mathbb{1}\{Y_1 = 0\} \odot$$

$$\mathbb{1}\{Y_1 = 1\} \odot$$

$$\mathbb{1}\{Y_2 = 0\} \odot$$

Boolean tensor: $\mathcal{K} \in \{0, 1\}^4$

$$\mathbb{1}\{Y_2 = 1\} \odot$$

$$\mathbb{1}\{Y_3 = 0\} \odot$$

$$\mathbb{1}\{Y_3 = 1\} \odot$$

$$k_{y_1 y_2 y_3 y_4} = \mathbb{1}\{y_1 y_2 y_3 y_4 \models \mathsf{K}\}$$

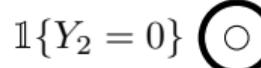
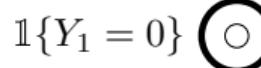
knowledge compilation

(as a Boolean tensor factorization)

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

Boolean tensor: $\mathcal{K} \in \{0, 1\}^4$

$$k_{y_1 y_2 y_3 y_4} = \mathbf{1}\{y_1 y_2 y_3 y_4 \models K\}$$



knowledge compilation

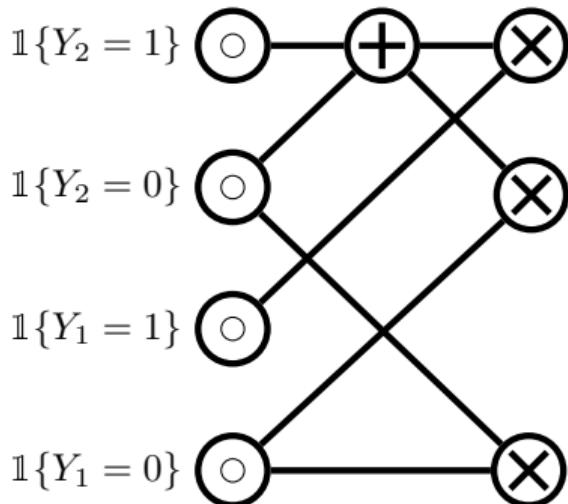
(as a Boolean tensor factorization)

$$K: (Y_1 = 1 \implies Y_3 = 1)$$

$$\wedge (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0, 1\}^4$

$$k_{y_1 y_2 y_3 y_4} = \mathbf{1}\{y_1 y_2 y_3 y_4 \models K\}$$



knowledge compilation

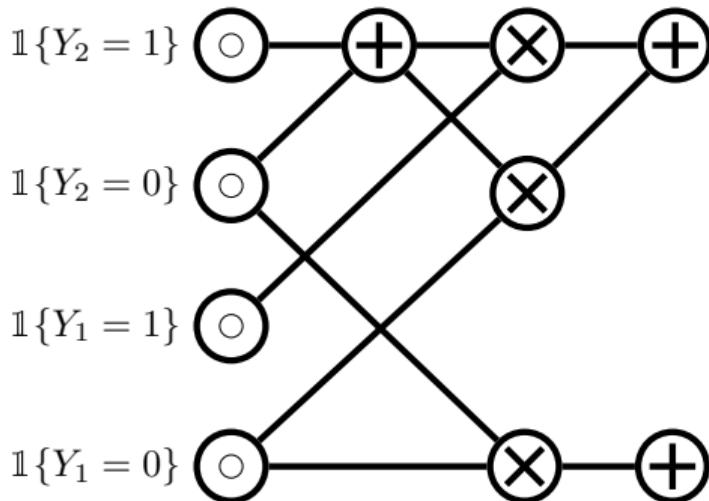
(as a Boolean tensor factorization)

$$K: (Y_1 = 1 \implies Y_3 = 1)$$

$$\wedge (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0, 1\}^4$

$$k_{y_1 y_2 y_3 y_4} = \mathbf{1}\{y_1 y_2 y_3 y_4 \models K\}$$



knowledge compilation

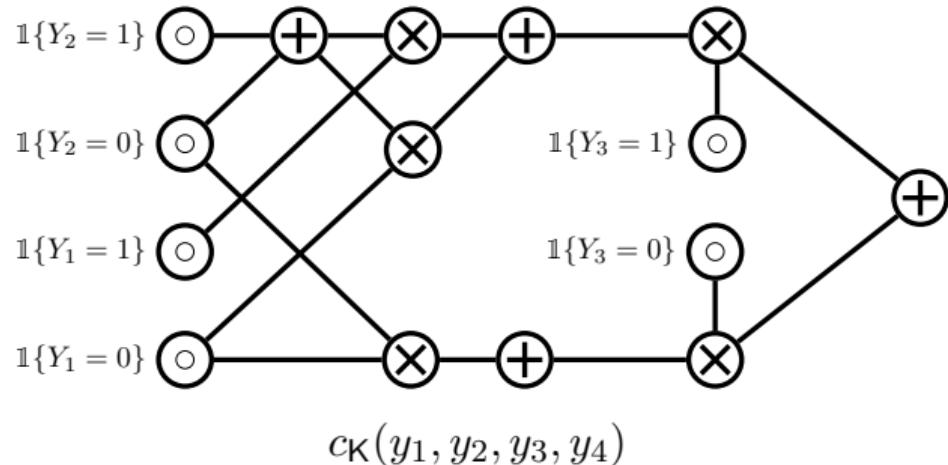
(as a Boolean tensor factorization)

$$K: (Y_1 = 1 \implies Y_3 = 1)$$

$$\wedge (Y_2 = 1 \implies Y_3 = 1)$$

Boolean tensor: $\mathcal{K} \in \{0, 1\}^4$

$$k_{y_1 y_2 y_3 y_4} = \mathbf{1}\{y_1 y_2 y_3 y_4 \models K\}$$



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)

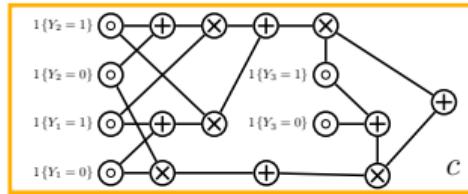
$$\begin{aligned} K : & (Y_1 = 1 \implies Y_3 = 1) \\ \wedge & (Y_2 = 1 \implies Y_3 = 1) \end{aligned}$$

1) Take a
logical constraint

NeSy AI recipe

with circuits (and tensor factorizations)

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



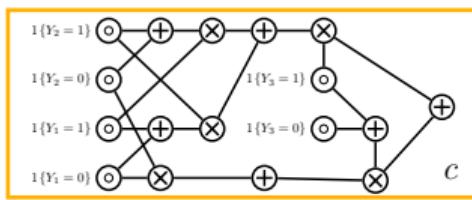
1) Take a
logical constraint

2) Compile it into
a constraint circuit

NeSy AI recipe

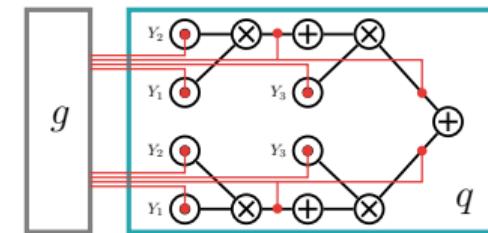
with circuits (and tensor factorizations)

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



1) Take a
logical constraint

2) Compile it into
a constraint circuit

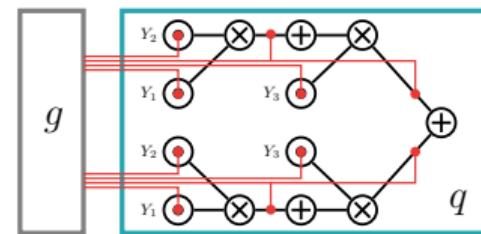
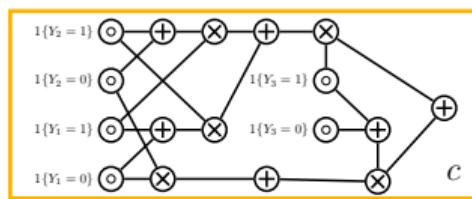


3) Multiply it
by a circuit distribution

NeSy AI recipe

with circuits (and tensor factorizations)

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



1) Take a
logical constraint

2) Compile it into
a constraint circuit

3) Multiply it
by a circuit distribution

4) ***train end-to-end by sgd!***

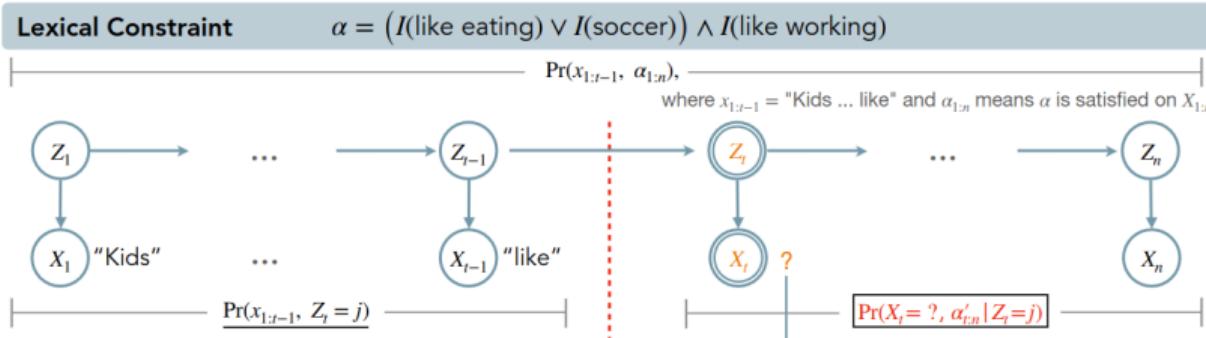
SPLs

(and more circuits)

everywhere

Tractable Control for Autoregressive Language Generation

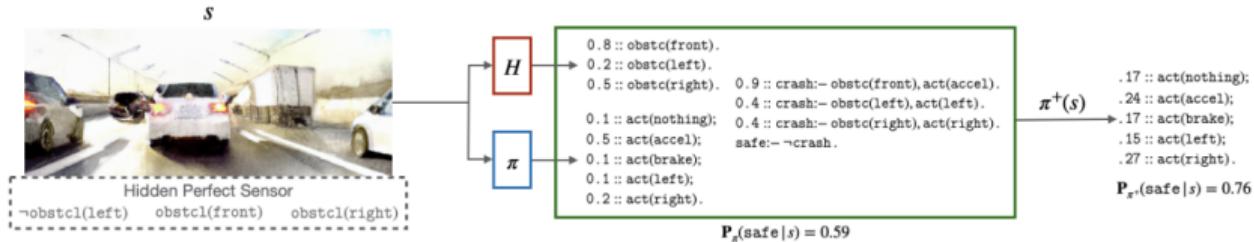
Honghua Zhang^{* 1} Meihua Dang^{* 1} Nanyun Peng¹ Guy Van den Broeck¹



constrained text generation with LLMs (ICML 2023)

Safe Reinforcement Learning via Probabilistic Logic Shields

Wen-Chi Yang¹, Giuseppe Marra¹, 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



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

smooth + decomposability

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

compatibility

q₃ $\mathbb{E}_{\mathbf{e} \sim p_{\text{noise}}(\mathbf{E})} [f(\mathbf{x} + \mathbf{e})]$
(adversarial robust.)

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

`+ , × , pow , log , exp` and `/`

⇒ *many divergences and information-theoretic queries*

A language for queries

Integral expressions that can be formed by composing these operators

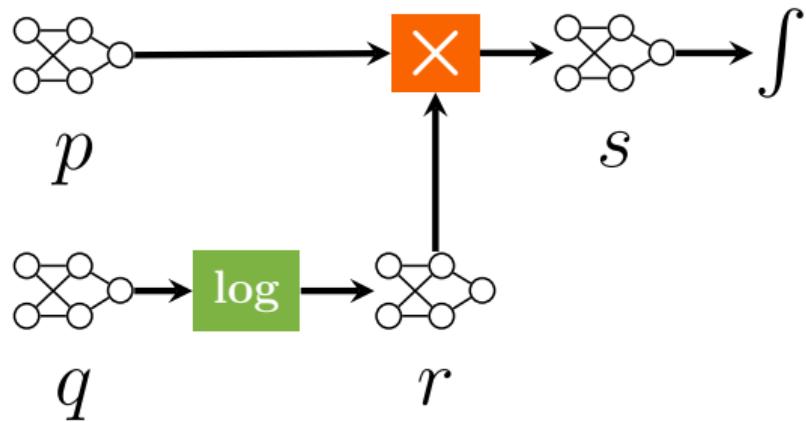
`+ , × , pow , log , exp` and `/`

⇒ *many divergences and information-theoretic queries*

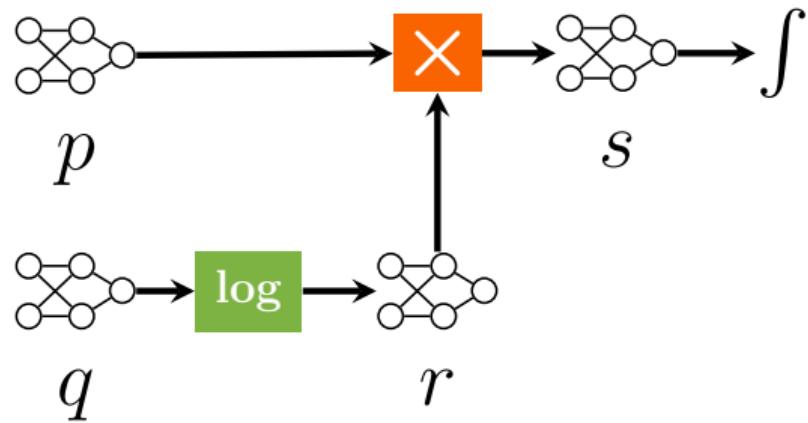
Represented as ***higher-order computational graphs***—pipelines—operating over circuits!

⇒ *re-using intermediate transformations across queries*

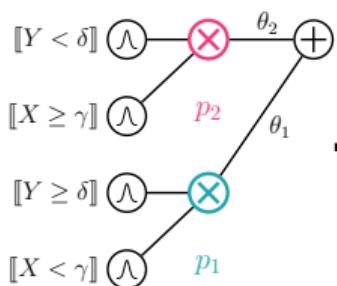
$$\text{XENT}(p \parallel q) = \int p(\mathbf{x}) \times \log q(\mathbf{x}) d\mathbf{X}$$



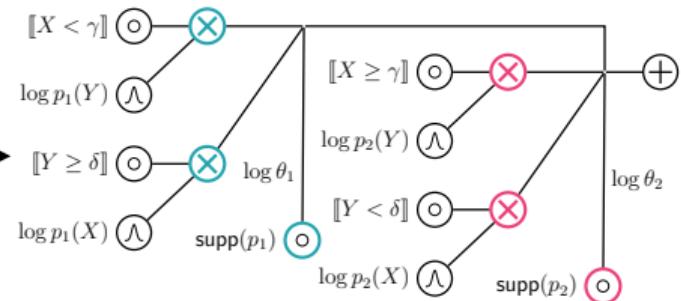
$$\text{XENT}(p \parallel q) = \int p(\mathbf{x}) \times \log q(\mathbf{x}) d\mathbf{X}$$



Tractable operators



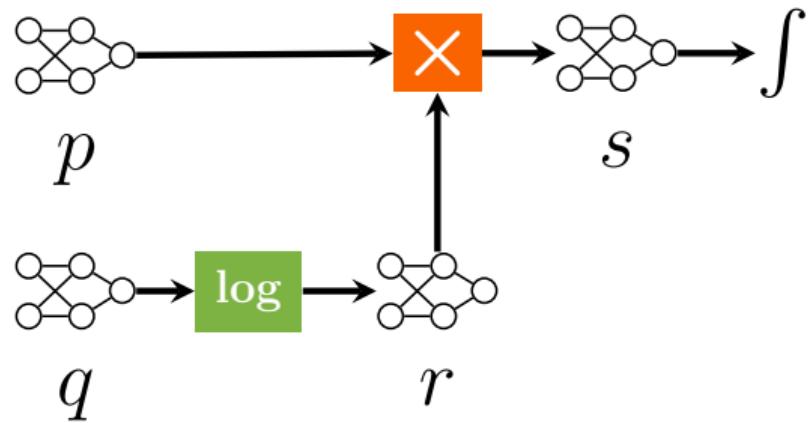
log



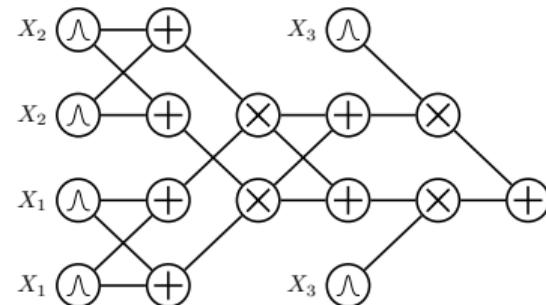
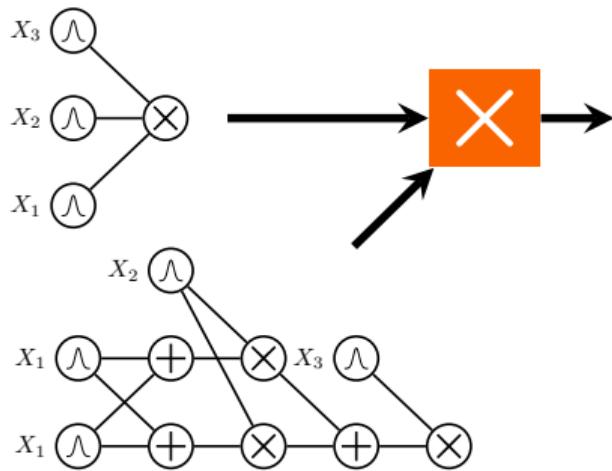
*smooth, decomposable
deterministic*

smooth, decomposable

$$\text{XENT}(p \parallel q) = \int p(\mathbf{x}) \times \log q(\mathbf{x}) d\mathbf{X}$$

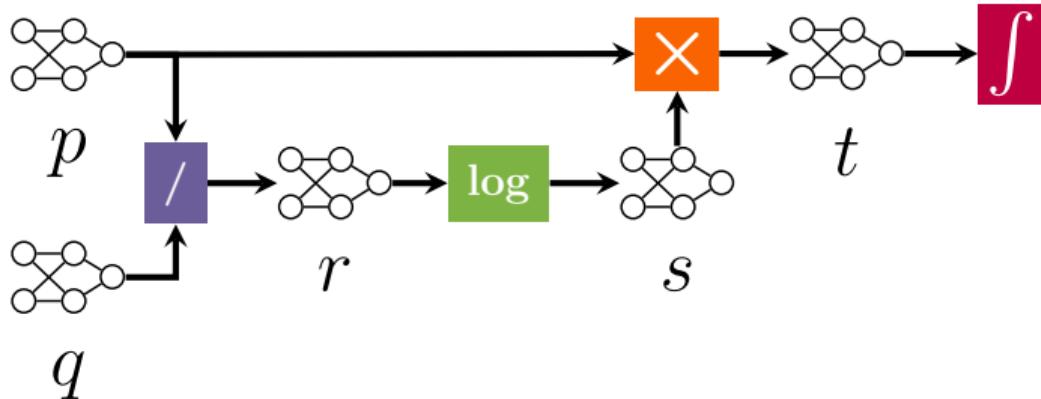


Tractable operators



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

...and compositionally derive many more tractable algorithms

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!"

⇒ storing $\mathcal{T} \in \mathbb{R}^{M \times \dots \times M}$ 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!"

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

One factorization method may require
exponentially smaller rank than others

⇒ it is more **expressive**

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**

⇒ 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**

⇒ 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”*

SUBTRACTIVE MIXTURE MODELS VIA SQUARING: REPRESENTATION AND LEARNING

Lorenzo Loconte^{1*} Aleksanteri M. Sladek² Stefan Mengel³

Martin Trapp² Arno Solin² Nicolas Gillis⁴ Antonio Vergari¹

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**

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**

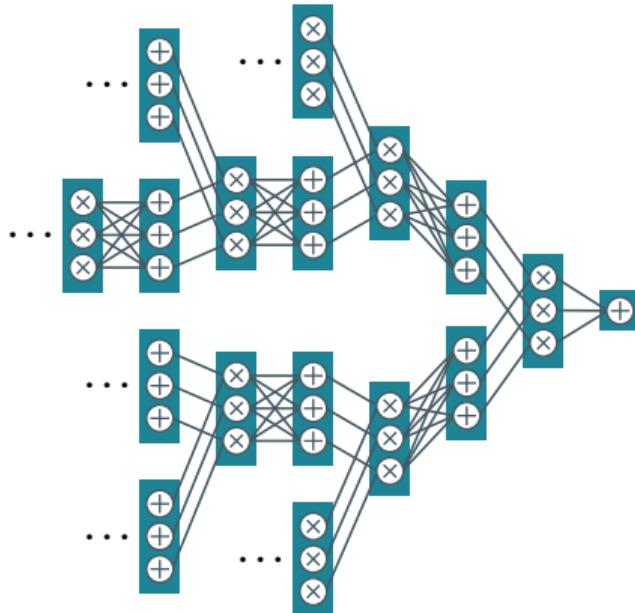
(represent non-negative tensor factorizations [Cichocki and Phan 2009])

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

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

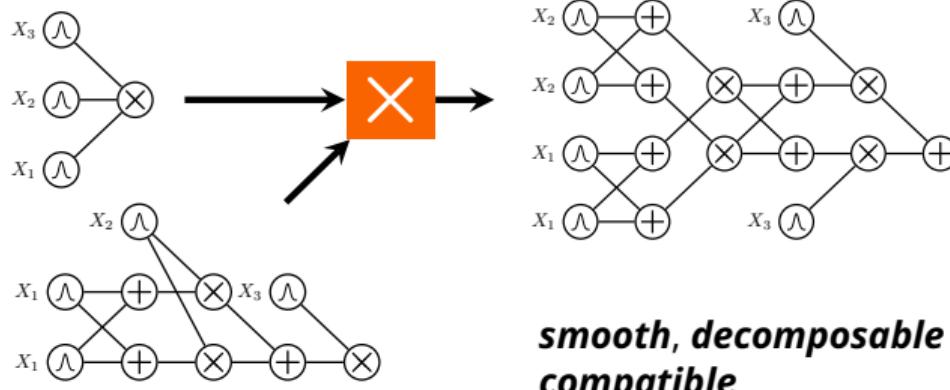
where parameters and input functions can be **negative**

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



Tractable product

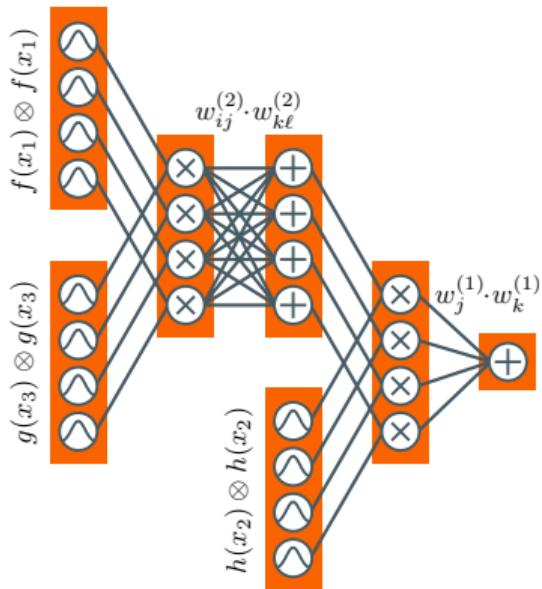
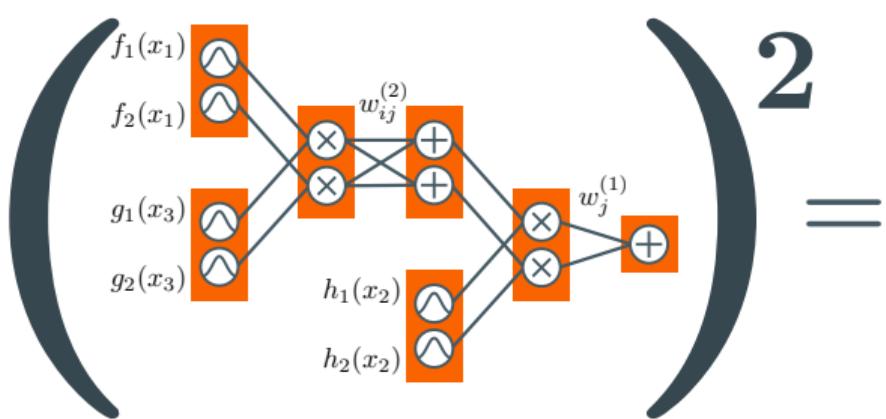
thanks to circuit compatibility

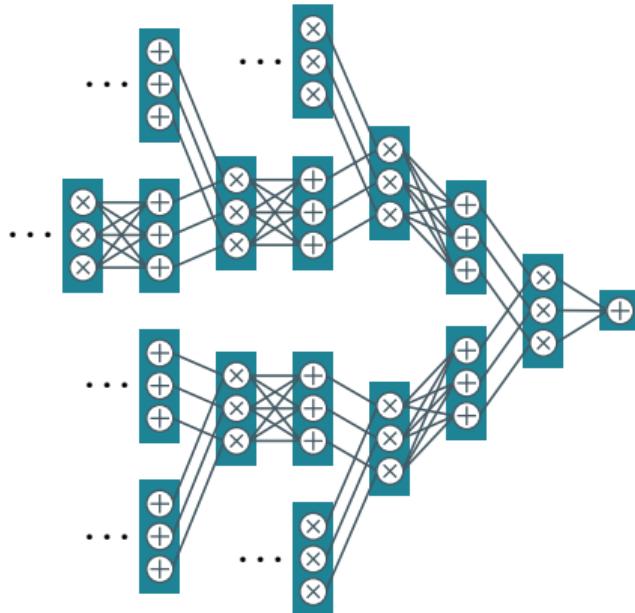


Squared circuits

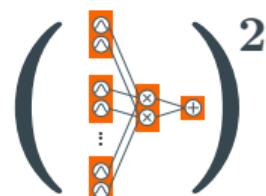
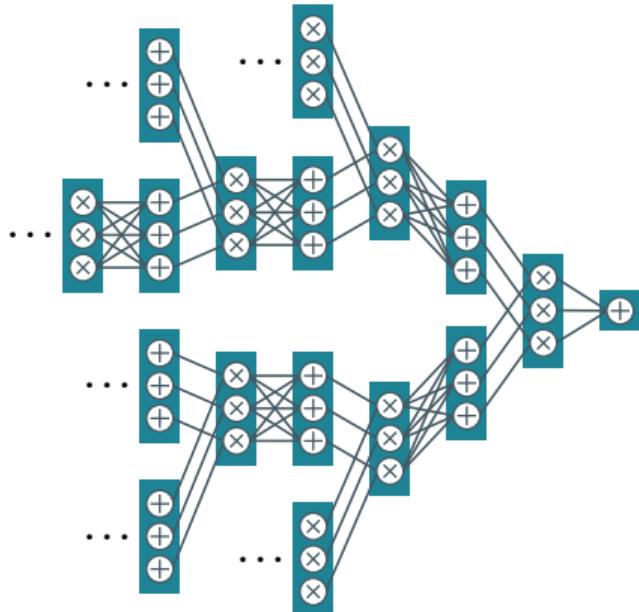
$$p(\mathbf{x}) = \frac{1}{Z} c^2(\mathbf{x}), \quad c(\mathbf{x}) \in \mathbb{R}$$

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

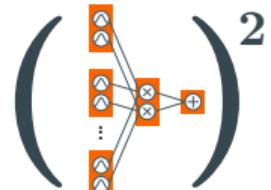
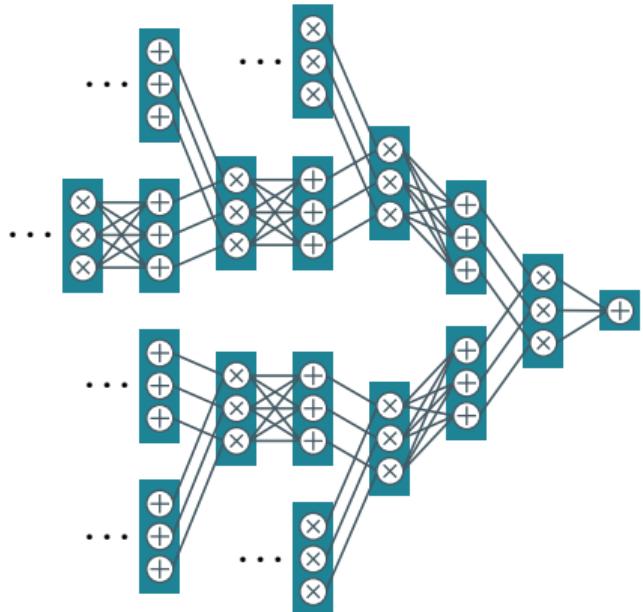




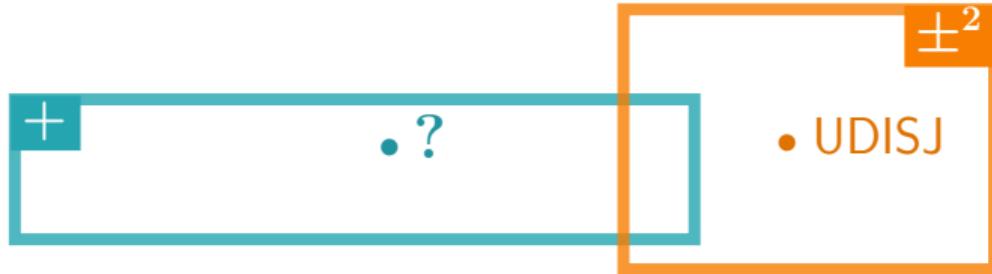
$\exists p$ requiring **exponentially large monotonic circuits...**



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

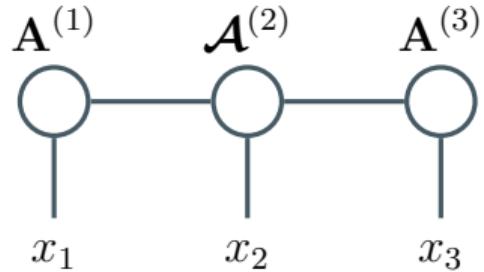


Squared circuits more expressive than **monotonic** ones



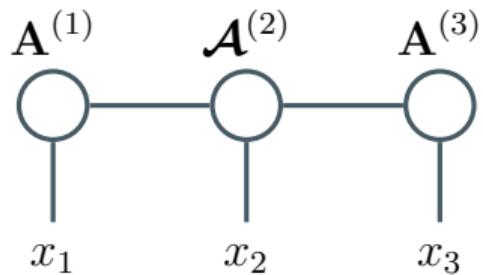
“Can monotonic circuits be more expressive than squared?”

Squared circuits are sparse Born machines

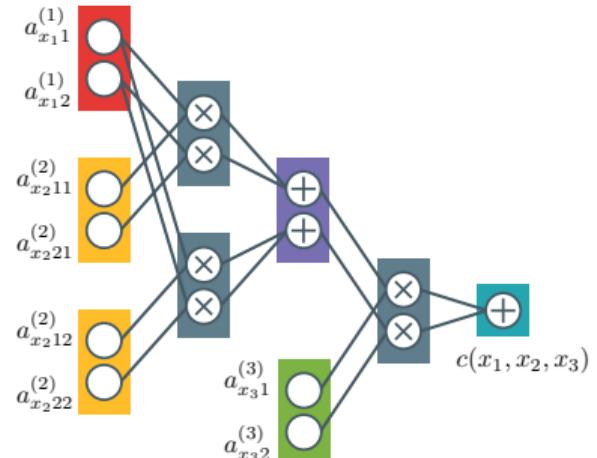


$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$

Squared circuits are sparse Born machines

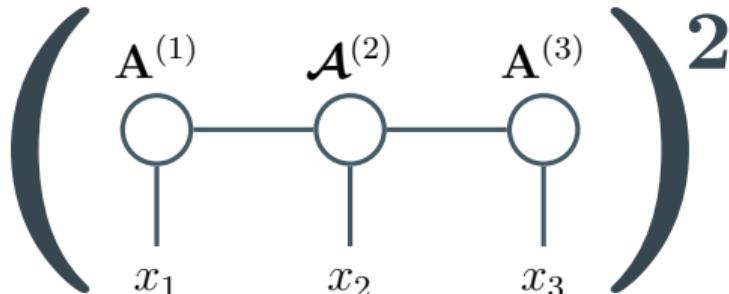


$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$



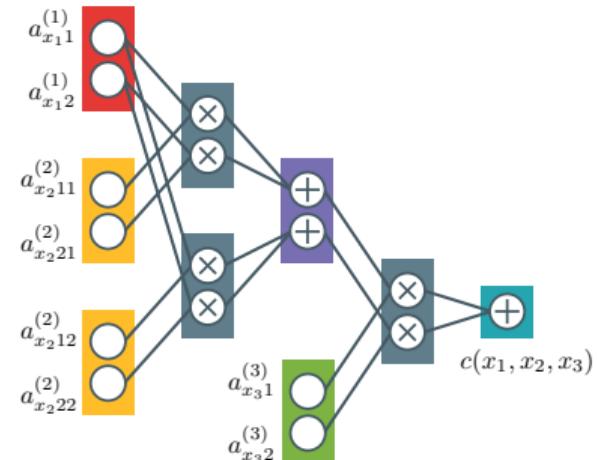
circuit compatible with itself

Squared circuits are sparse Born machines



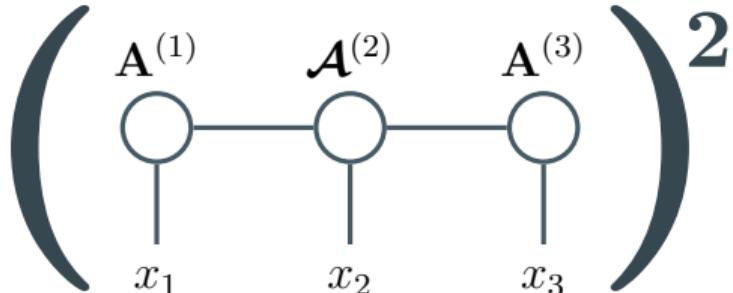
$$t_{x_1 x_2 x_3} = \sum_{r_1=1}^R \sum_{r_2=1}^R a_{x_1 r_1}^{(1)} a_{r_1 x_2 r_2}^{(2)} a_{r_2 x_3}^{(3)}$$

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



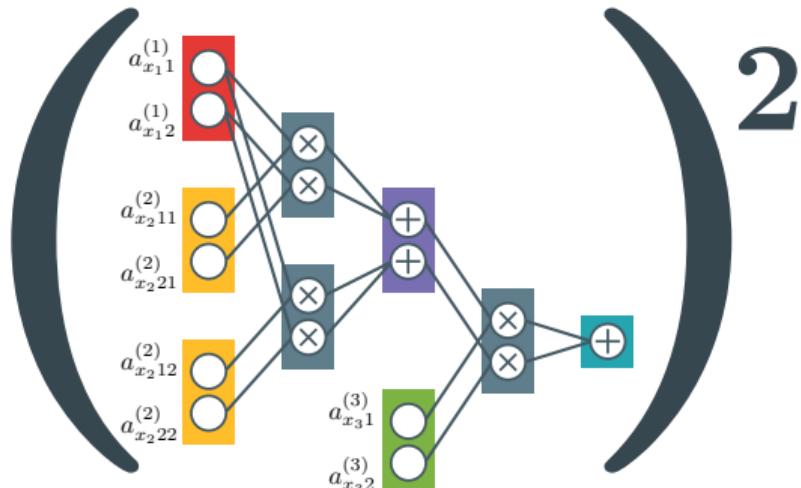
circuit compatible with itself

Squared circuits are sparse Born machines



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

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



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 $2^{\Omega(d)}$.

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?"



"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

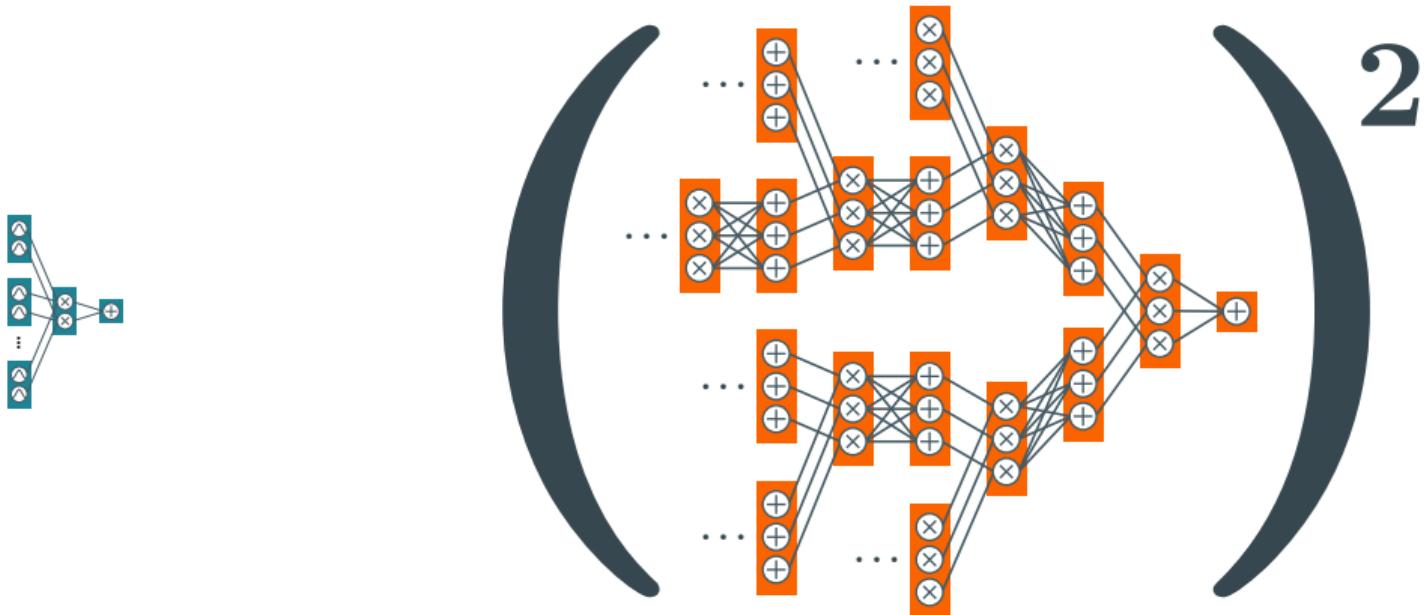
Wang and Van den Broeck

“On the Relationship Between Monotone
and Squared Probabilistic Circuits”

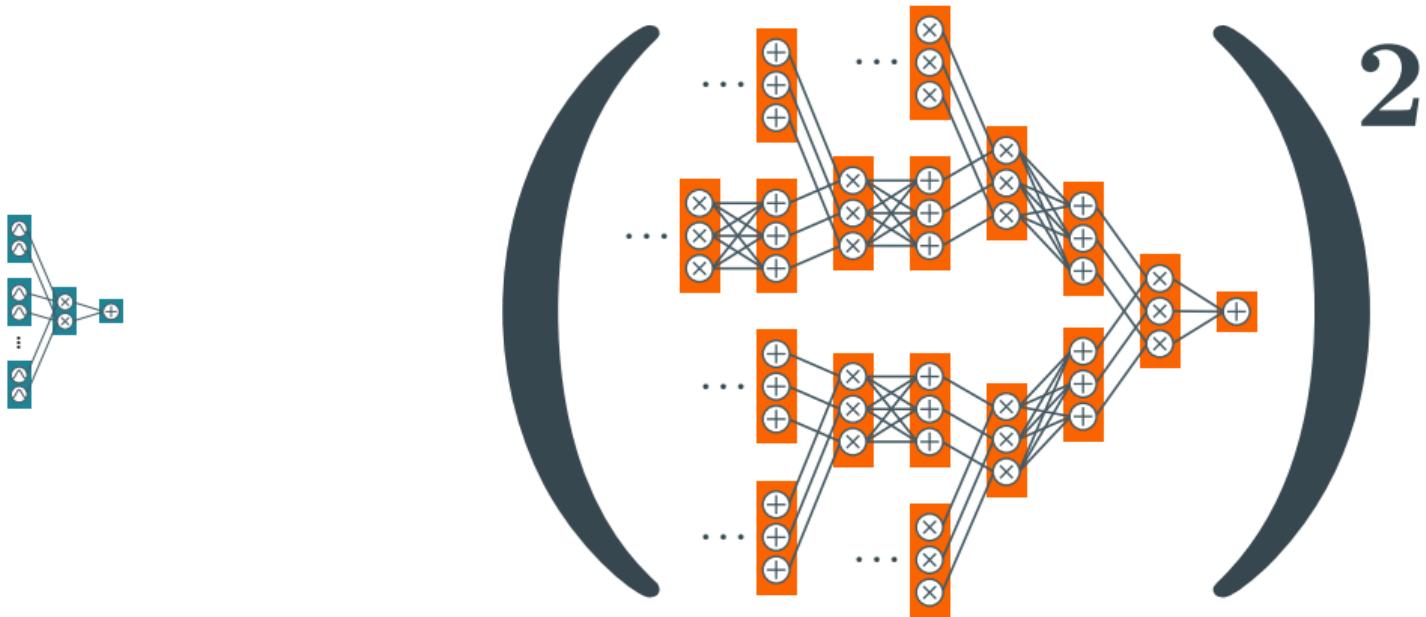
(also at AAAI 2025)



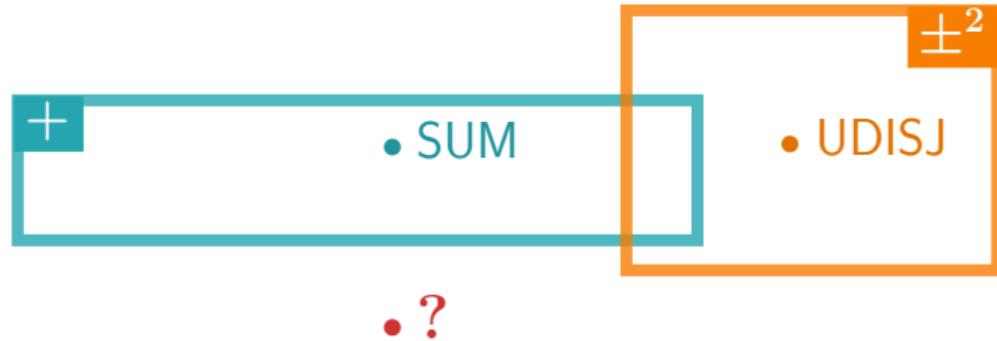
$\exists p$ requiring **polysize monotonic circuits**...



...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 both? ”

Sum of squares (SOS) circuits

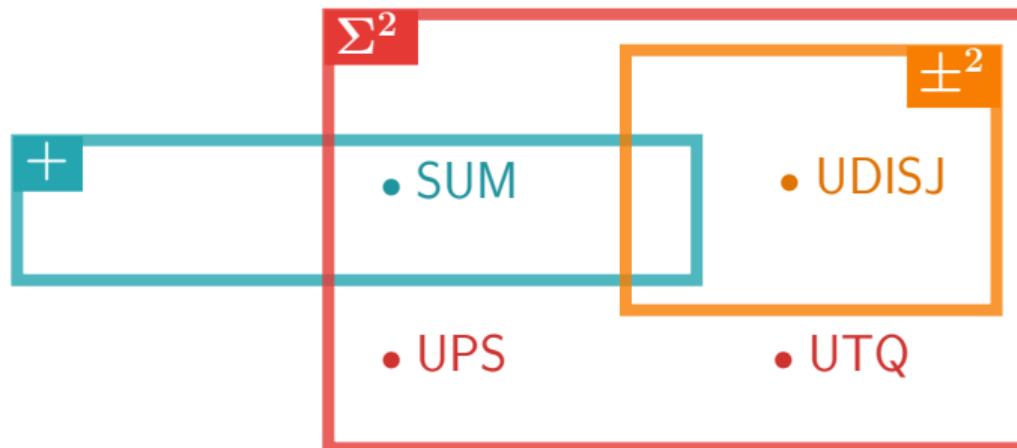
$$p(\mathbf{x}) = \frac{1}{Z} \sum_{i=1}^r c_i^2(\mathbf{x}), \quad c_i(\mathbf{x}) \in \mathbb{R}$$

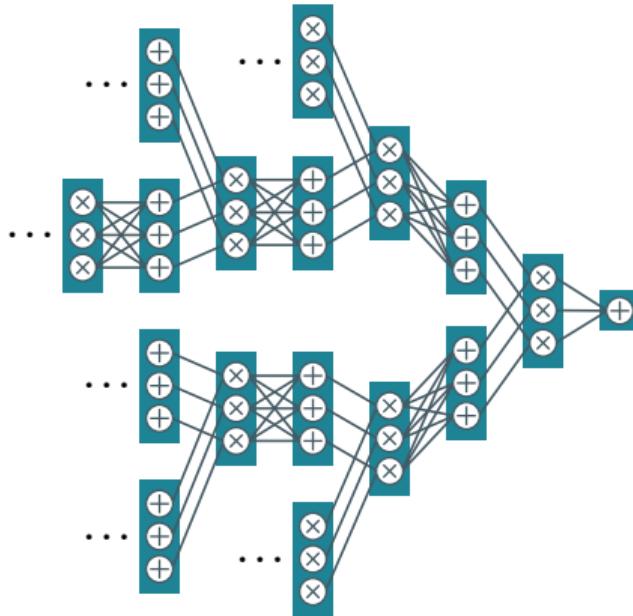
where parameters and input functions can be **negative**

Sum of squares (SOS) circuits

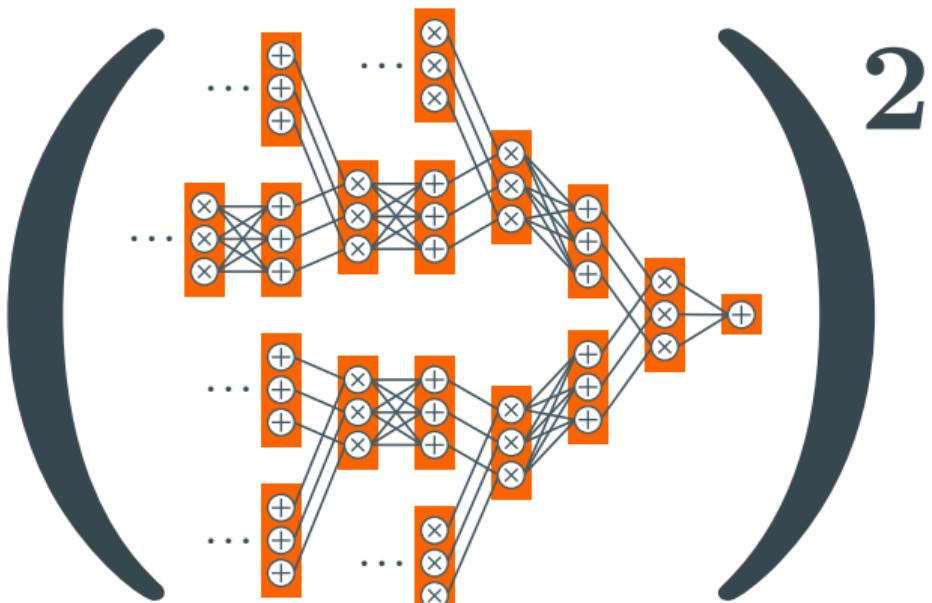
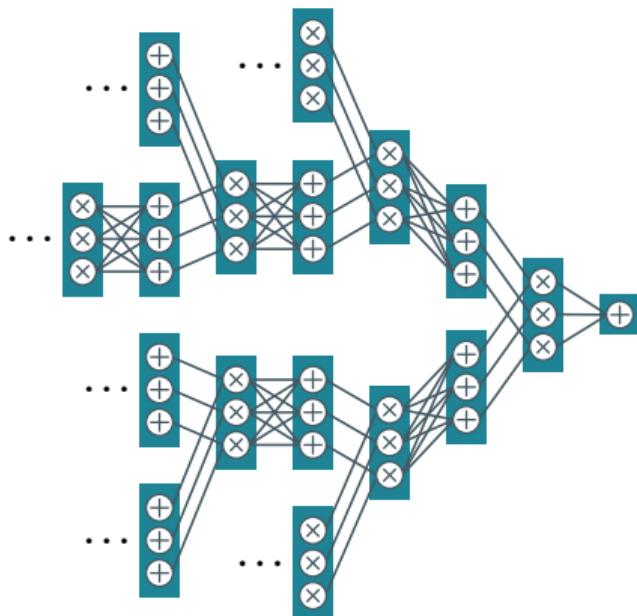
$$p(\mathbf{x}) = \frac{1}{Z} \sum_{i=1}^r c_i^2(\mathbf{x}), \quad c_i(\mathbf{x}) \in \mathbb{R}$$

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

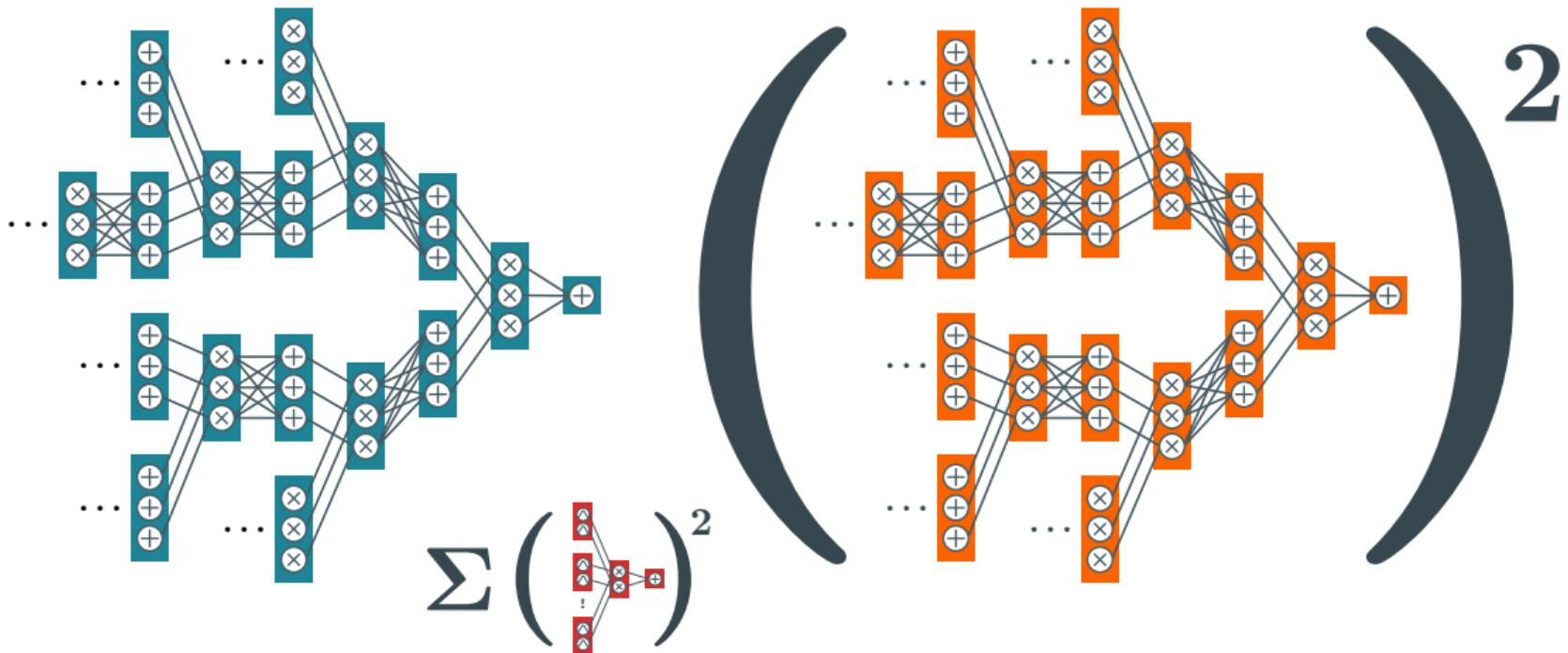




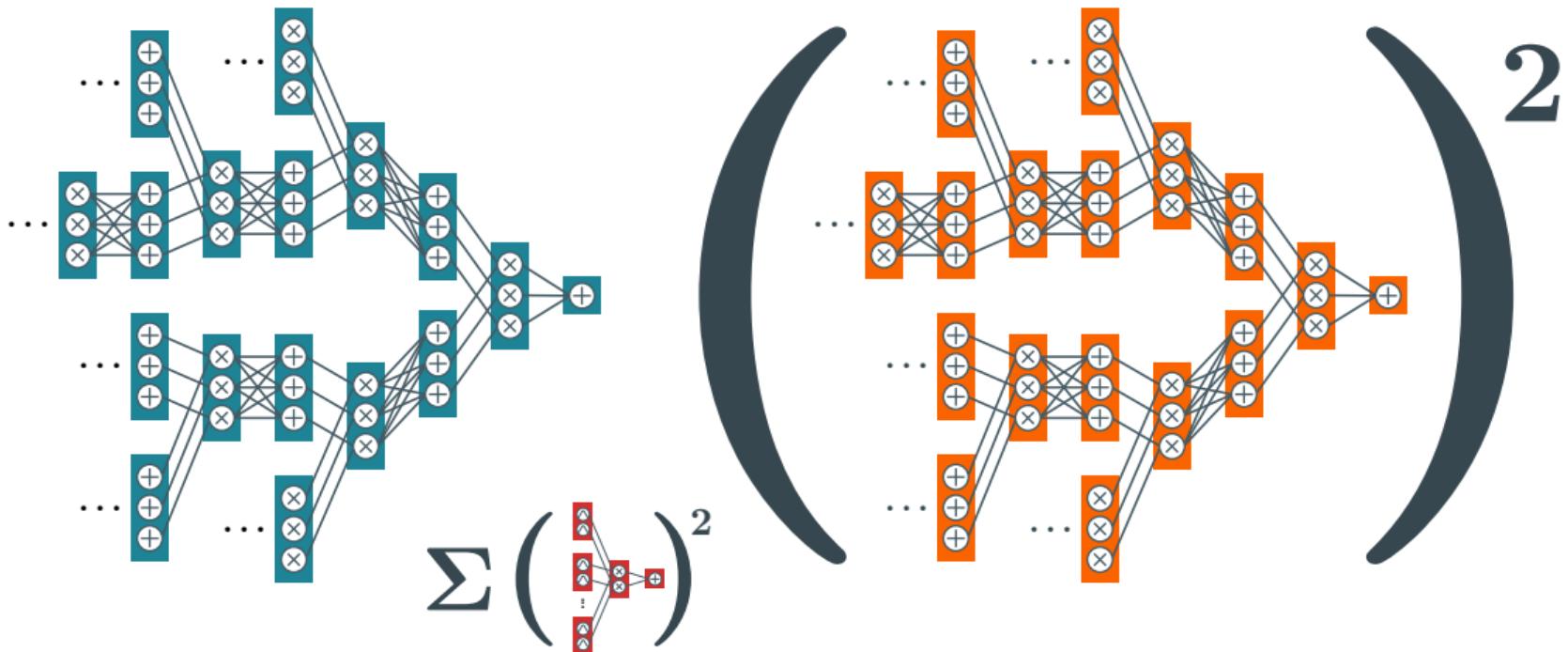
$\exists p$ requiring exponentially large **monotonic circuits**...



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

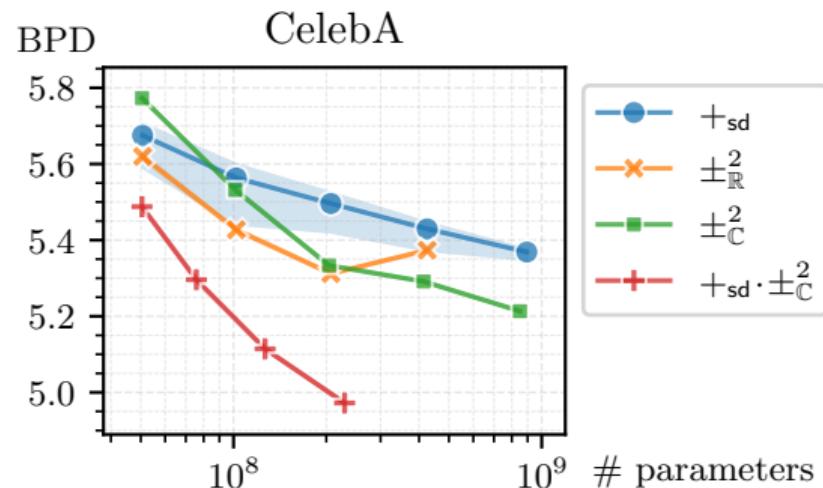
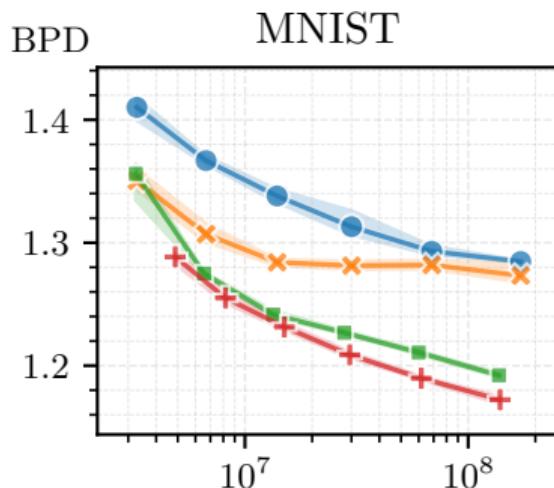


...but a **sum of squares (SOS)** polysize circuits



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?

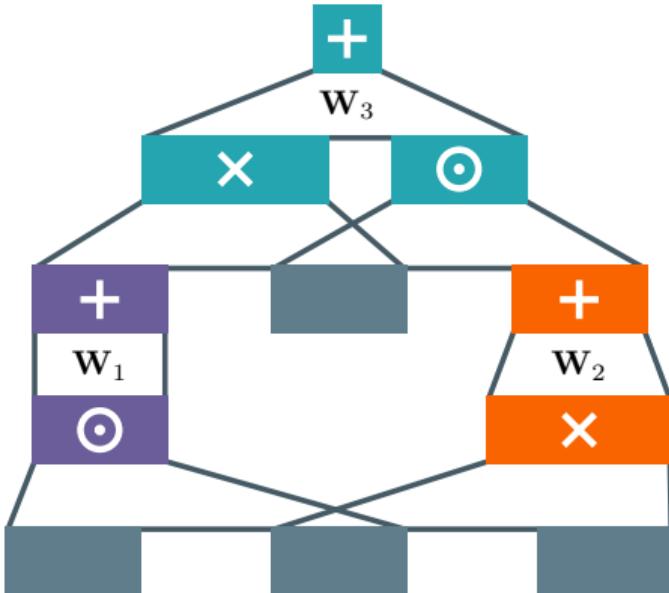
- 1 factorizations and circuits **expressiveness results...**
bridge rank and circuit size
- 2 circuits can help proving **stronger expressiveness results**
e.g., results from Born machines to squared circuits
- 3 **sum of squared circuits** are more expressive (use them!)

wait...!

conclusions...?

PC ARCHITECTURE	REGION GRAPH	SUM-PRODUCT LAYER	FOLD
Poon&Domingos (Poon & Domingos, 2011)	PD	CP^\top	✗
RAT-SPN (Peharz et al., 2020c)	RND	Tucker	✗
EiNet (Peharz et al., 2020a)	{ RND, PD }	Tucker	✓
HCLT (Liu & Van den Broeck, 2021b)	CL	CP^\top	✓
HMM/MPS $_{\mathbb{R} \geq 0}$ (Glasser et al., 2019)	LT	CP^\top	✗
BM (Han et al., 2018)	LT	CP^\top	✗
TTDE (Novikov et al., 2021)	LT	CP^\top	✗
NPC ² (Loconte et al., 2024)	{ LT, RND }	{ CP^\top , Tucker }	✓
TTN (Cheng et al., 2019)	QT-2	Tucker	✗
Mix & Match (our pipeline)	$\left\{ \begin{array}{l} \text{RND}, \text{PD}, \text{LT}, \\ \text{CL}, \text{QG}, \text{QT-2}, \text{QT-4} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{Tucker}, \text{CP}, \text{CP}^\top \\ \text{CP}^S, \text{CP}^{XS} \mid \text{FOLD } \checkmark \end{array} \right\} \cup \left\{ \begin{array}{l} \text{CP}^S, \text{CP}^{XS} \mid \text{FOLD } \checkmark \end{array} \right\}$	$\times \{ \text{✗, ✓} \}$

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

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

Lorenzo Loconte

School of Informatics, University of Edinburgh, UK

l.loconte@sms.ed.ac.uk

Antonio Mari

École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

antonio.mari@epfl.ch

Gennaro Gala

Eindhoven University of Technology, NL

g.gala@tue.nl

Robert Peharz

Graz University of Technology, Austria

robert.peharz@tugraz.at

Cassio de Campos

Eindhoven University of Technology, NL

c.decampos@tue.nl

Erik Quaeghebeur

Eindhoven University of Technology, NL

e.quaeghebeur@tue.nl

Gennaro Vessio

Computer Science Department, University of Bari Aldo Moro, IT

gennaro.vessio@uniba.it

Antonio Vergari

School of Informatics, University of Edinburgh, UK

avergari@ed.ac.uk

accepted at TMLR featured certification



learning & reasoning with circuits in pytorch

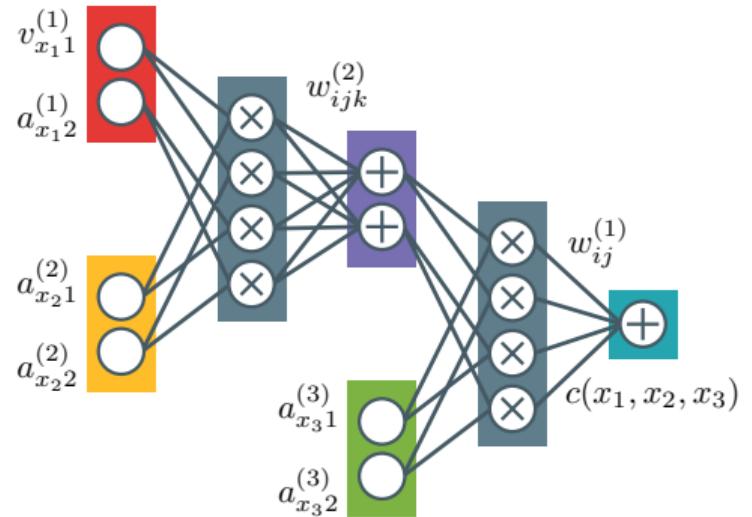
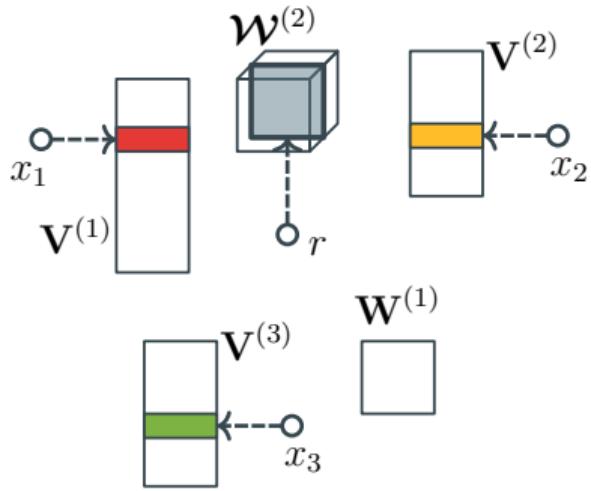
<https://github.com/april-tools/cirkit>

colorai

*connecting
low-rank
representations
in ai*

workshop at AAAI-25, March 4

april-tools.github.io/colorai/



questions?