# *probabilistic circuits*
# *from inference to learning*

**antonio vergari** (he/him) **robert peharz**

*@tetraduzione*

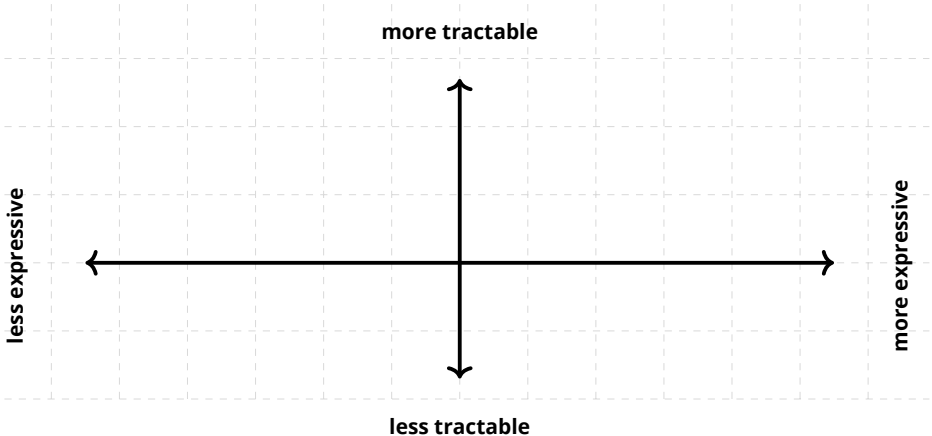
*16th July 2024* - **ESSAI 2024** - *Athens*

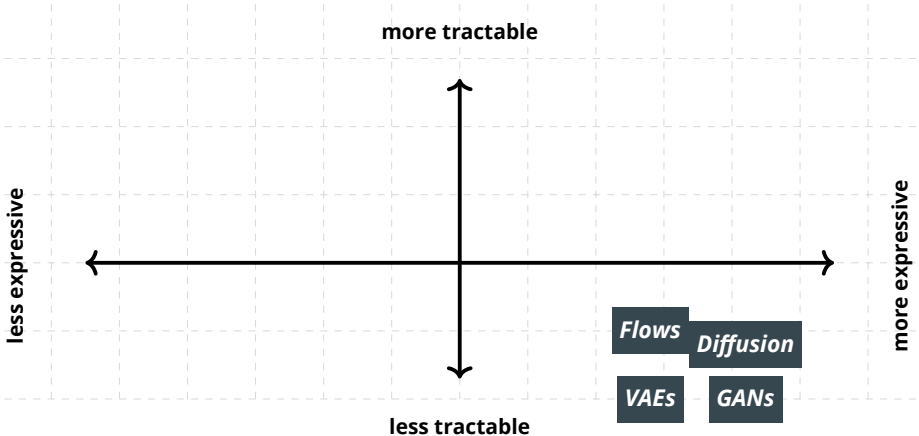
slides available at

***deep generative models***

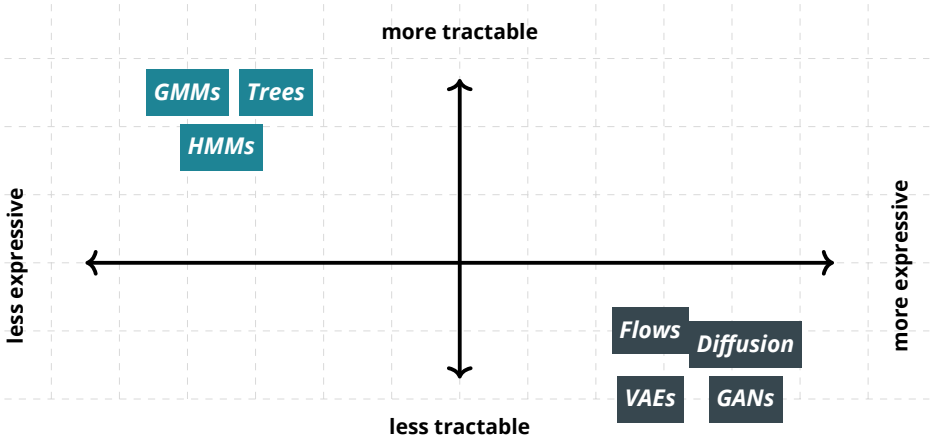
**+**

***flexible and reliable***
***logic & probabilistic reasoning?***

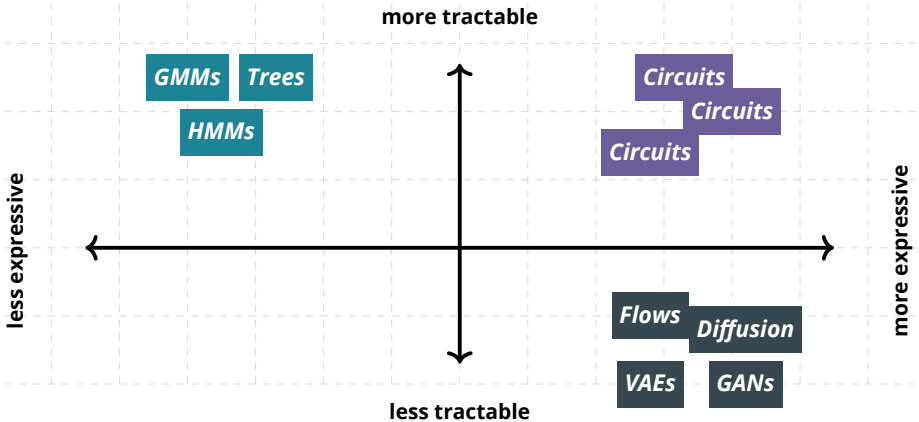

more tractable
less expressive
more expressive
less tractable



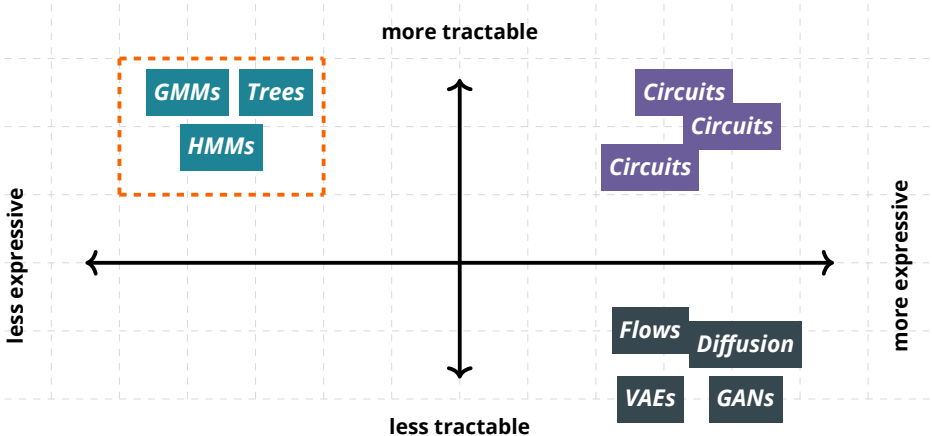

***expressive models are not much tractable...***



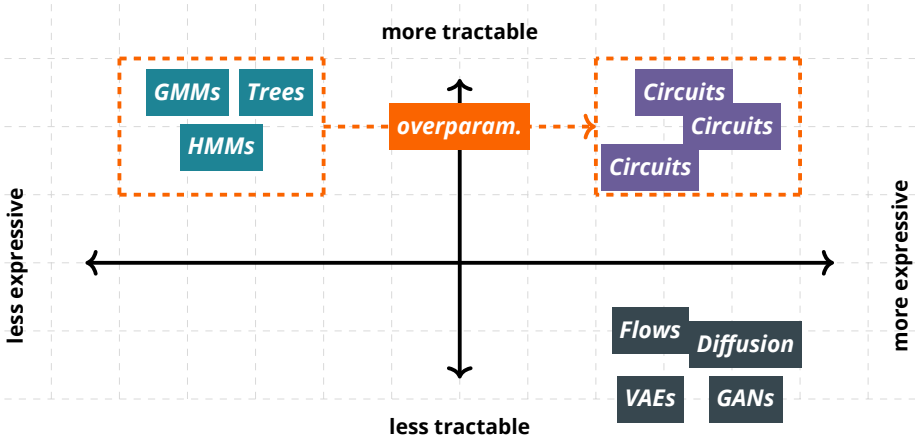

***tractable models are not that expressive...***



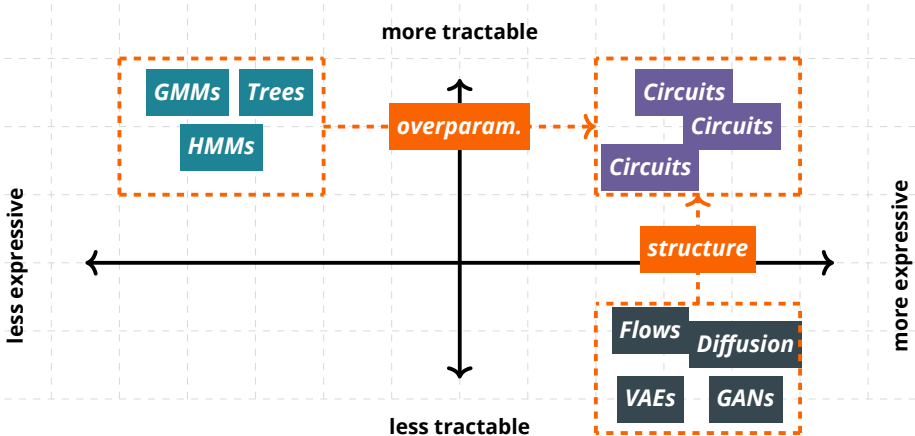

***circuits can be both expressive and tractable!***




***start simple...***




***then make it more expressive!***




***impose structure!***

# Probabilistic Circuits (PCs)

*A grammar for tractable computational graphs*

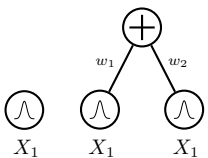
I. *A simple tractable function is a circuit*

# Probabilistic Circuits (PCs)

*A grammar for tractable computational graphs*

I. *A simple tractable function is a circuit*

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



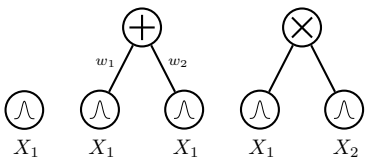
# Probabilistic Circuits (PCs)

*A grammar for tractable computational graphs*

I. *A simple tractable function is a circuit*

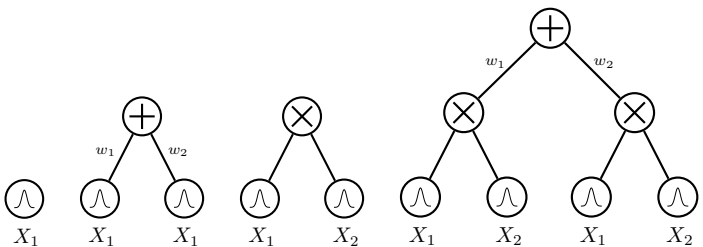
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



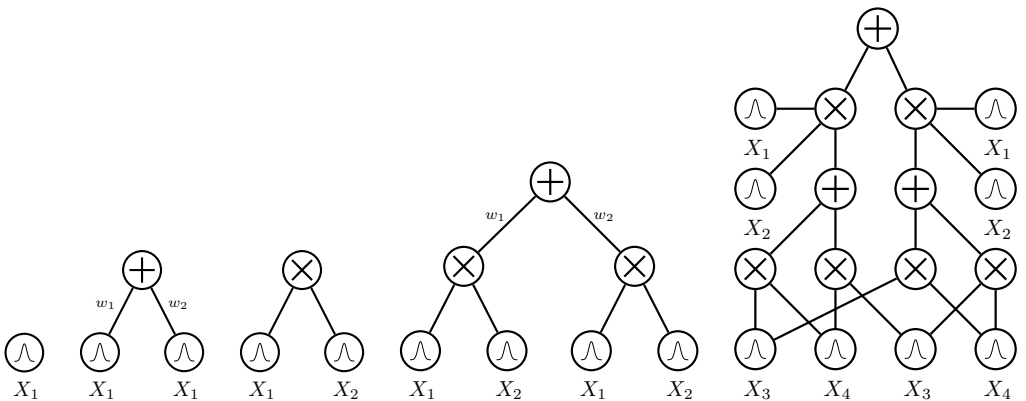
# Probabilistic Circuits (PCs)

*A grammar for tractable computational graphs*



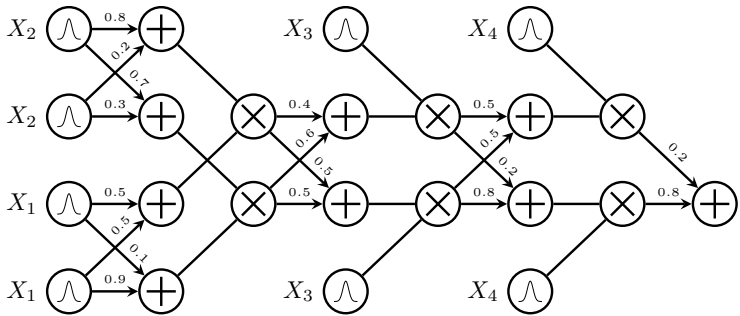
# Probabilistic Circuits (PCs)

*A grammar for tractable computational graphs*



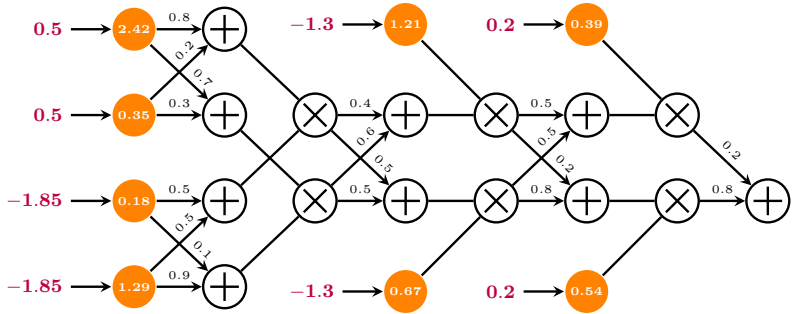
**Probabilistic queries** = **feedforward** evaluation

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



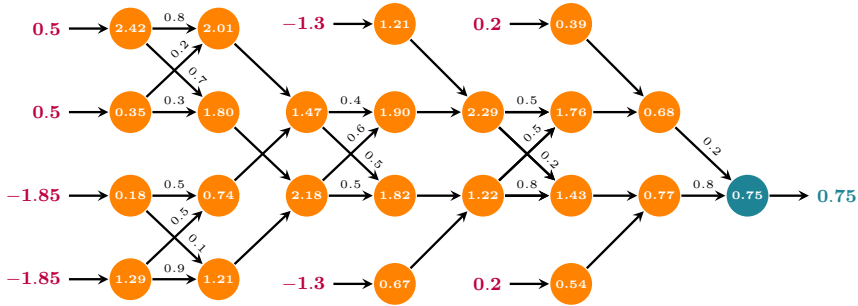
**Probabilistic queries** = **feedforward** evaluation

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



# Probabilistic queries = feedforward evaluation

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



# ...why PCs?

***1. A grammar for tractable models***

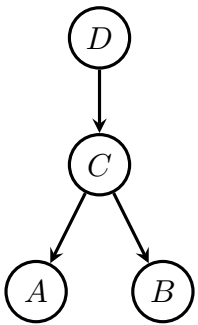
One formalism to represent many probabilistic and logical models

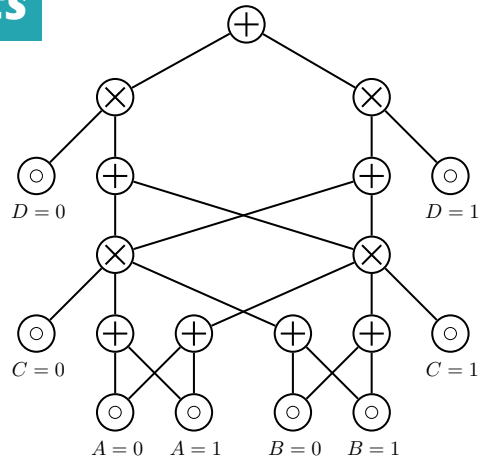
$\Rightarrow$ *#HMMs #Trees #XGBoost, Tensor Networks, ...*

and other *PGMs*...

# *from PGMs to circuits*

*via compilation*

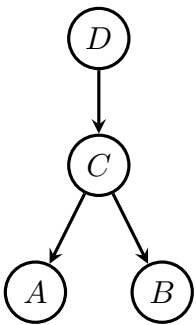


# *from PGMs to circuits*

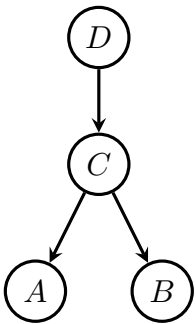
*via compilation*

Bottom-up ***compilation***: starting from leaves...



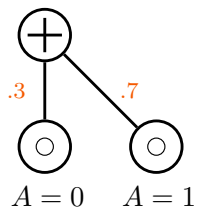
# *from PGMs to circuits*

*via compilation*



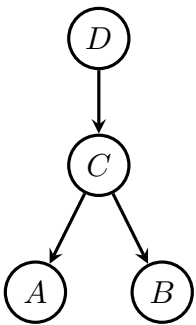

...compile a leaf CPT

$p(A|C=0)$



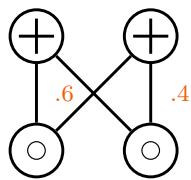
# *from PGMs to circuits*

*via compilation*



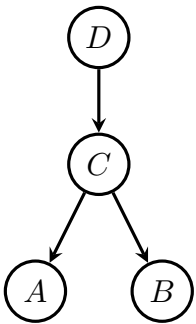

...compile a leaf CPT

$p(A|C=1)$

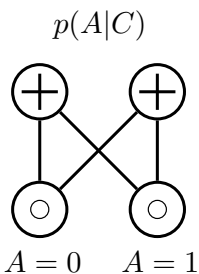


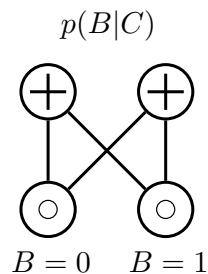
# *from PGMs to circuits*

*via compilation*



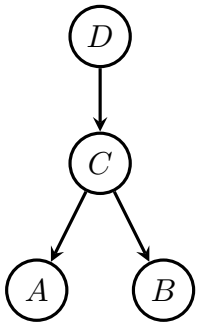

...compile a leaf CPT...for all leaves...



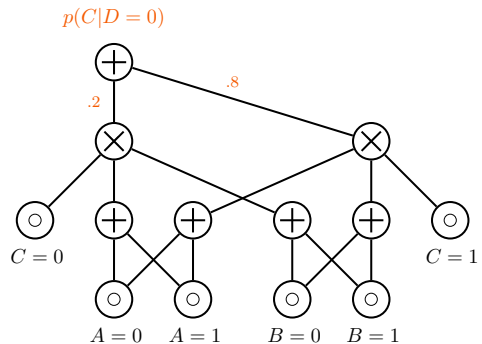



# *from PGMs to circuits*

*via compilation*

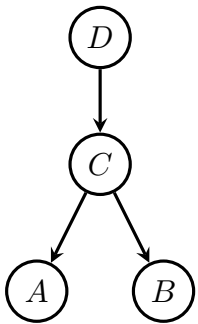



...and recurse over parents...

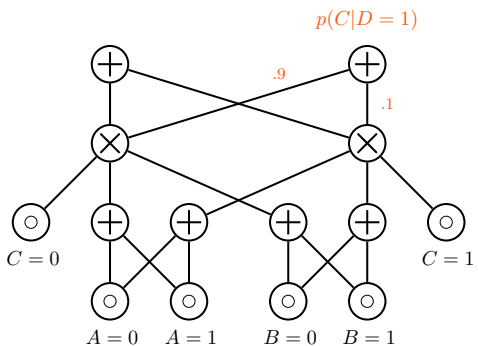


# *from PGMs to circuits*

*via compilation*

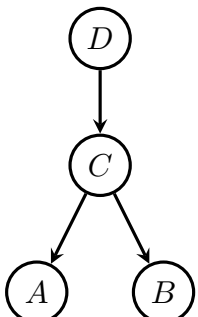


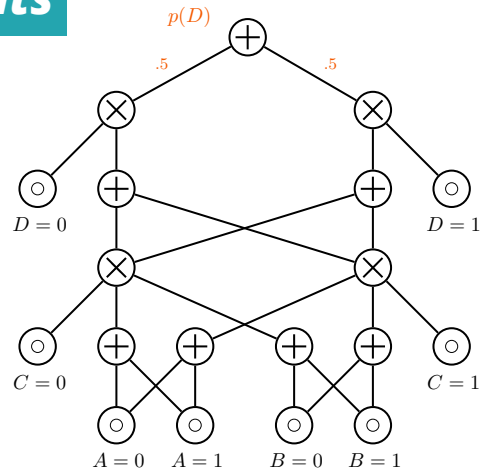

...while reusing previously compiled nodes!...



# from PGMs to circuits

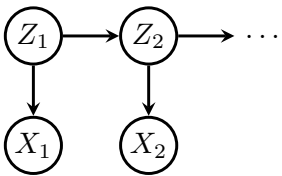
*via compilation*

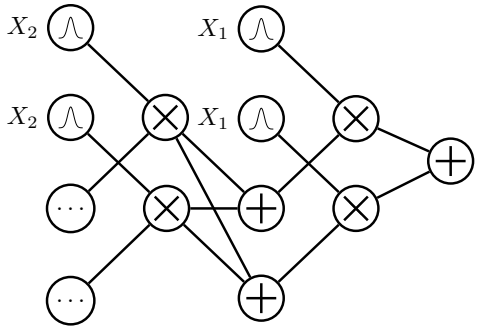


# HMMs

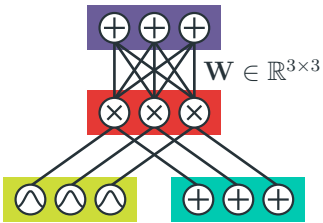
*as computational graphs*

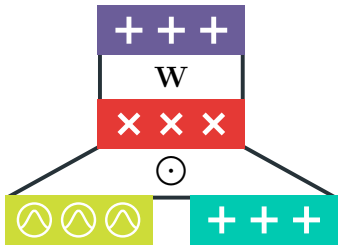


# ***overparameterize***

*and tensorize*

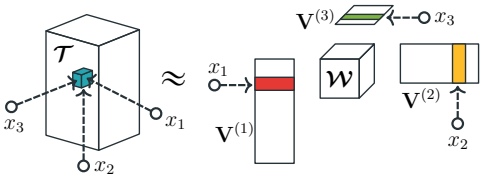


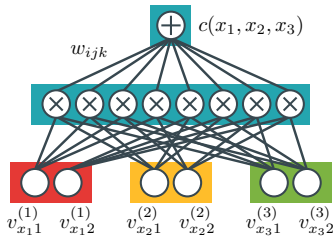




*Mari et al., "Unifying and Understanding Overparameterized Circuit Representations via Low-Rank Tensor Decompositions", 6th Workshop on Tractable Probabilistic Modeling, 2023*

# tensor factorizations

*as circuits*




*Mari et al., "What is the Relationship between Circuits and Tensor Factorizations (and How Can We Exploit it)?", Under submission, 2024*

# ...why PCs?

***1. A grammar for tractable models***

One formalism to represent many probabilistic and logical models

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

and other *PGMs*...

# ...why PCs?

***1. A grammar for tractable models***

One formalism to represent many probabilistic and logical models

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

and other *PGMs*...

***2. Expressiveness***

Competitive with intractable models, VAEs, Flow...*#hierachical #mixtures #polynomials*

# how expressive?

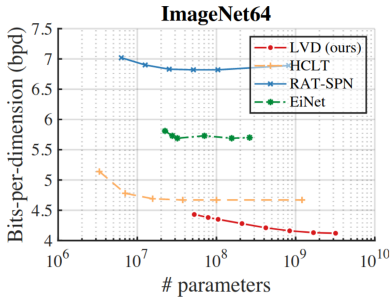
| | QPC | PC | Sp-PC | HCLT | RAT | IDF | BitS | BBans | McB |
|---|---|---|---|---|---|---|---|---|---|
| MNIST | **1.11** | 1.17 | 1.14 | 1.21 | 1.67 | 1.90 | 1.27 | 1.39 | 1.98 |
| F-MNIST | **3.16** | 3.32 | 3.27 | 3.34 | 4.29 | 3.47 | 3.28 | 3.66 | 3.72 |
| EMN-MN | 1.55 | 1.64 | **1.52** | 1.70 | 2.56 | 2.07 | 1.88 | 2.04 | 2.19 |
| EMN-LE | **1.54** | 1.62 | 1.58 | 1.75 | 2.73 | 1.95 | 1.84 | 2.26 | 3.12 |
| EMN-BA | **1.59** | 1.66 | 1.60 | 1.78 | 2.78 | 2.15 | 1.96 | 2.23 | 2.88 |
| EMN-BY | 1.53 | **1.47** | 1.54 | 1.73 | 2.72 | 1.98 | 1.87 | 2.23 | 3.14 |

**competitive with Flows and VAEs!**

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

# how scalable?

| Dataset | TPMs | | | | DGMs | | |
|---|---|---|---|---|---|---|---|
| | LVD (ours) | HCLT | EiNet | RAT-SPN | Glow | RealNVP | BIVA |
| ImageNet32 | **4.39**±0.01 | 4.82 | 5.63 | 6.90 | 4.09 | 4.28 | 3.96 |
| ImageNet64 | **4.12**±0.00 | 4.67 | 5.69 | 6.82 | 3.81 | 3.98 | - |
| CIFAR | **4.38**±0.02 | 4.61 | 5.81 | 6.95 | 3.35 | 3.49 | 3.08 |




## up to billions of parameters

*Liu et al., "Scaling Up Probabilistic Circuits by Latent Variable Distillation", The Eleventh International Conference on Learning Representations, 2022*

# ...why PCs?

***1. A grammar for tractable models***

One formalism to represent many probabilistic and logical models

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

and other *PGMs*...

***2. Expressiveness***

Competitive with intractable models, VAEs, Flow...*#hierachical #mixtures #polynomials*

***3. Tractability == Structural Properties***!!!

Exact computations of reasoning tasks are certified by guaranteeing certain structural properties. *#marginals #expectations #MAP, #product ...*

# Structural properties

**smoothness**

**decomposability**

**determinism**

**compatibility**

---

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

# Structural properties

**property A**

**property B**

**property C**

**property D**

---

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

# Structural properties

**property A**

**property B**

**property C**

**property D**

***tractable*** computation of ***arbitrary integrals***

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

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

$\Rightarrow$ *tractable partition function*

---

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

# Structural properties

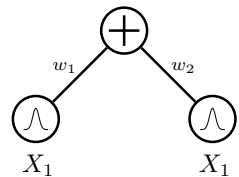
**smoothness**

**decomposability**

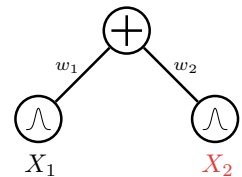
**compatibility**

**determinism**

the inputs of sum units are defined over the same variables




***smooth* circuit**




***non-smooth* circuit**

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

# Structural properties

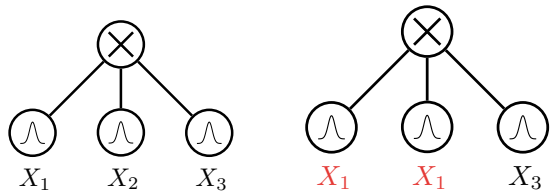
***smoothness***

***decomposability***

***compatibility***

***determinism***

the inputs of prod units are defined over disjoint variable sets

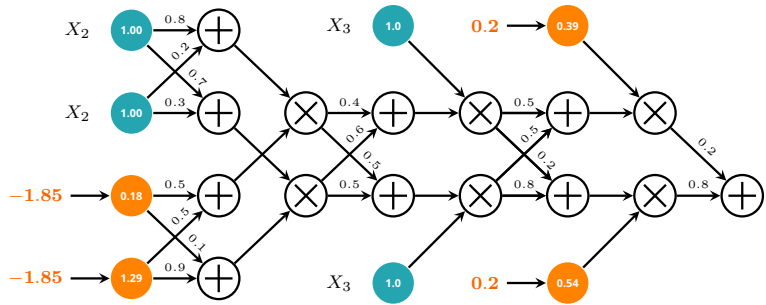



***decomposable circuit*** ***non-decomposable circuit***

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

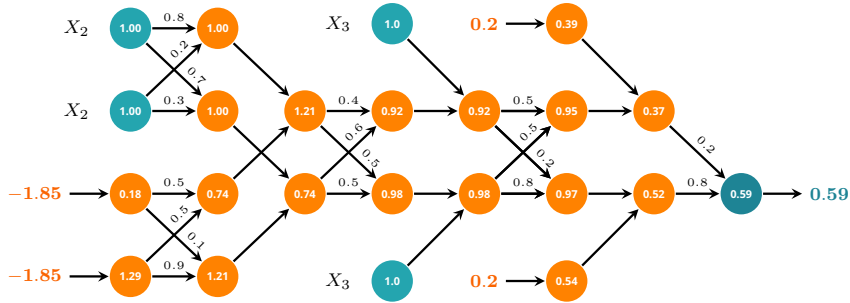
**Probabilistic queries** = **feedforward** evaluation

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



**Probabilistic queries** = **feedforward** evaluation

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

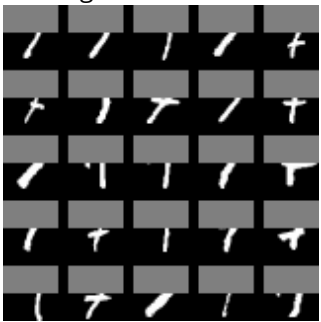


# Tractable inference on PCs

*Einsum networks*

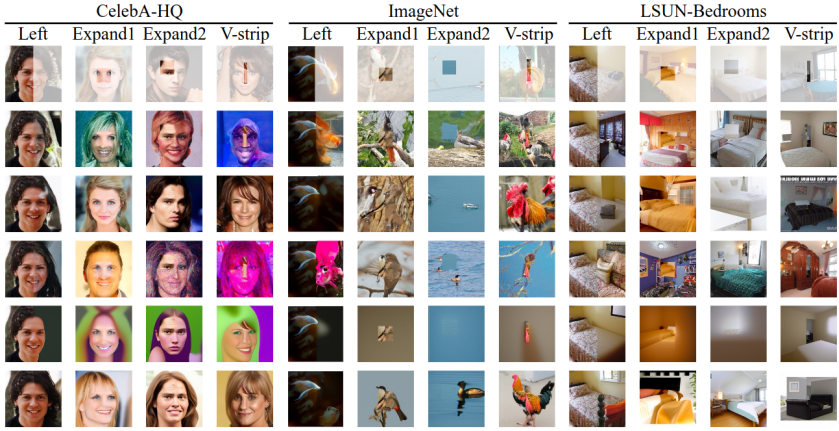
Original

Missing



Conditional sample

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




*Liu et al., "Image Inpainting via Tractable Steering of Diffusion Models", ICLR, 2024*

# general expectations

Integrals involving two or more functions:

$$\int p(\mathbf{x}) f(\mathbf{x}) d\,\mathbf{X}$$

# general expectations

Integrals involving two or more functions:

$$\int p(\mathbf{x}) f(\mathbf{x}) d\,\mathbf{X}$$

represent both $p$ and $f$ as circuits...but with which structural properties?

# general expectations

Integrals involving two or more functions:

$$\int \boldsymbol{p}(\mathbf{x})\boldsymbol{f}(\mathbf{x})d\,\mathbf{X}$$

represent both $\boldsymbol{p}$ and $\boldsymbol{f}$ as circuits...but with which structural properties?

E.g.,

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

# Structural properties

***smoothness***

***decomposability***

***compatibility***

***determinism***

---

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

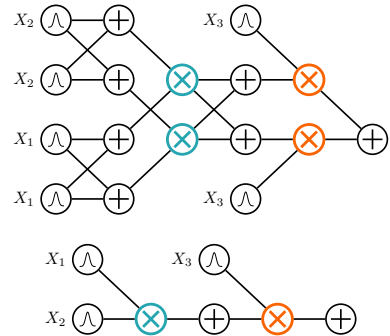
# Structural properties

***smoothness***

***decomposability***

***compatibility***

***determinism***




***compatible circuits***

---

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

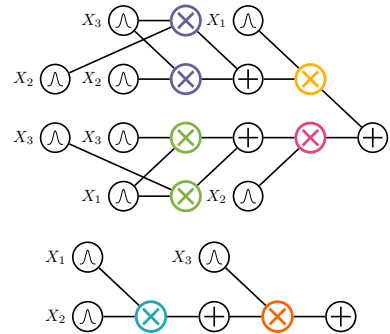
# Structural properties

***smoothness***

***decomposability***

***compatibility***

***determinism***




***non-compatible circuits***

---

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

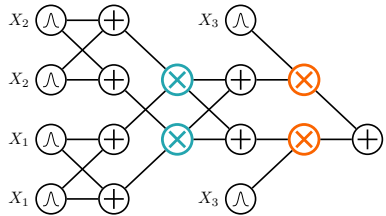
# Structural properties

***smoothness***

***decomposability***

***compatibility***

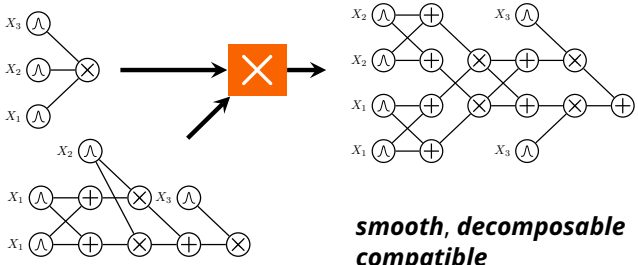
***determinism***




a circuit that is compatible with itself is called
***structured decomposable***

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

# Tractable products




***smooth*, *decomposable***
***compatible***

**exactly compute** $\int \boldsymbol{p}(\mathrm{x})\boldsymbol{f}(\mathrm{x})d\mathrm{X}$ **in time** $O(|\boldsymbol{p}||\boldsymbol{f}|)$

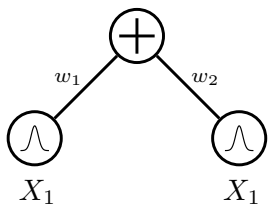
---

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

# Tractable products

*an intuition why/how*

$$p(\mathbf{X}) = \sum_{i=1}^{K} w_i p_i(\mathbf{X}), \qquad \sum_{i=1}^{K} w_i = 1, \quad w_i \geq 0,$$



# Tractable products

*an intuition why/how*

$$p(\mathbf{X}) = \sum_{i=1}^{K} w_i p_i(\mathbf{X}), \qquad \sum_{i=1}^{K} w_i = 1, \quad w_i \geq 0,$$

$$q(\mathbf{X}) = \sum_{j=1}^{L} w'_j q_j(\mathbf{X}), \qquad \sum_{j=1}^{L} w'_j = 1, \quad w'_j \geq 0,$$

# Tractable products

*an intuition why/how*

$$p(\mathbf{X}) = \sum_{i=1}^{K} w_i p_i(\mathbf{X}), \qquad \sum_{i=1}^{K} w_i = 1, \quad w_i \geq 0,$$

$$q(\mathbf{X}) = \sum_{j=1}^{L} w'_j q_j(\mathbf{X}), \qquad \sum_{j=1}^{L} w'_j = 1, \quad w'_j \geq 0,$$

$$p(\mathbf{X}) \times q(\mathbf{X}) = \sum_{i=1}^{K} \sum_{j=1}^{L} w_i w_j p_i(\mathbf{X}) q_j(\mathbf{X}),$$

**cartesian product of “local” mixtures**

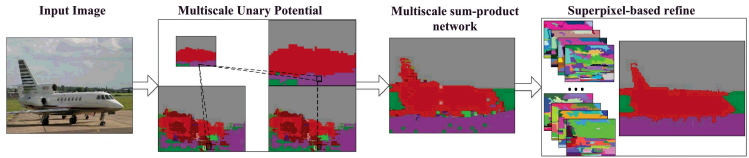
# Maximum-a-posteriori (MAP) Inference

*aka Most probable explanation (MPE)*

E.g., multi-label classification: what are the most likely labels $\mathbf{y}$ for an input $\mathbf{x}$?

$$\operatorname*{argmax}_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$$

E.g., image segmentation: what is the most likely latent space for the given pixels?




*Yuan et al., "Modeling spatial layout for scene image understanding via a novel multiscale sum-product network", , 2016*

*Friesen et al., "Submodular Sum-product Networks for Scene Understanding", , 2016*

# Structural properties

***smoothness***

***decomposability***

***compatibility***

***determinism***

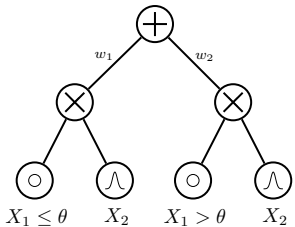
---

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

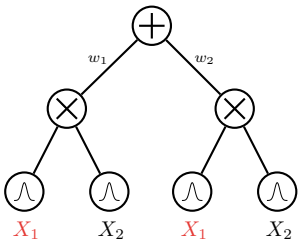
# determinism

*aka support-decomposability*

A sum unit is deterministic if its inputs have disjoint supports




***deterministic circuit***




***non*-deterministic circuit**

*Darwiche et al., "A knowledge compilation map", , 2002*

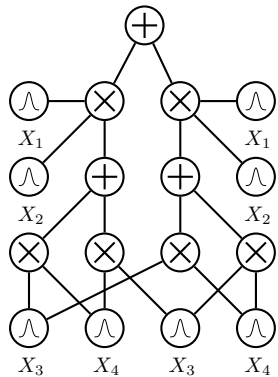
***determinism*** **+** ***decomposability*** **=** ***tractable MAP***

Computing maximization with arbitrary evidence $\mathbf{e}$

$\Rightarrow$ *linear in circuit size!*

E.g., suppose we want to compute:

$$\max_{\mathbf{q}} p(\mathbf{q} \mid \mathbf{e})$$

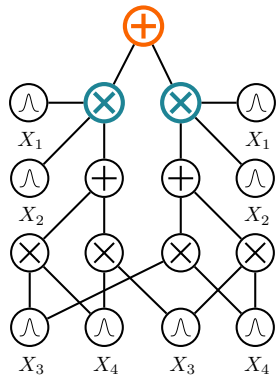


**determinism** + **decomposability** = **tractable MAP**

If $\boldsymbol{p}(\mathbf{q}, \mathbf{e}) = \sum_i w_i \boldsymbol{p}_i(\mathbf{q}, \mathbf{e}) = \max_i w_i \boldsymbol{p}_i(\mathbf{q}, \mathbf{e})$,
(***deterministic*** sum unit):

$$\begin{aligned}\max_{\mathbf{q}} \boldsymbol{p}(\mathbf{q}, \mathbf{e}) &= \max_{\mathbf{q}} \sum_i w_i \boldsymbol{p}_i(\mathbf{q}, \mathbf{e}) \\ &= \max_{\mathbf{q}} \max_i w_i \boldsymbol{p}_i(\mathbf{q}, \mathbf{e}) \\ &= \max_i \max_{\mathbf{q}} w_i \boldsymbol{p}_i(\mathbf{q}, \mathbf{e})\end{aligned}$$

$\Longrightarrow$ *one non-zero term, thus sum is max*

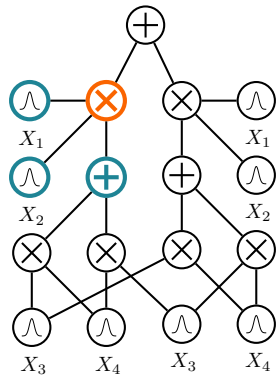


**determinism** + **decomposability** = **tractable MAP**

If $\boldsymbol{p}(\mathbf{q}, \mathbf{e}) = \boldsymbol{p}(\mathbf{q_x}, \mathbf{e_x}, \mathbf{q_y}, \mathbf{e_y}) = \boldsymbol{p}(\mathbf{q_x}, \mathbf{e_x})\boldsymbol{p}(\mathbf{q_y}, \mathbf{e_y})$
(***decomposable*** product unit):

$$\begin{aligned}\max_{\mathbf{q}} \boldsymbol{p}(\mathbf{q} \mid \mathbf{e}) &= \max_{\mathbf{q}} \boldsymbol{p}(\mathbf{q}, \mathbf{e}) \\ &= \max_{\mathbf{q_x}, \mathbf{q_y}} \boldsymbol{p}(\mathbf{q_x}, \mathbf{e_x}, \mathbf{q_y}, \mathbf{e_y}) \\ &= \max_{\mathbf{q_x}} \boldsymbol{p}(\mathbf{q_x}, \mathbf{e_x}) \cdot \max_{\mathbf{q_y}} \boldsymbol{p}(\mathbf{q_y}, \mathbf{e_y})\end{aligned}$$

$\Rightarrow$ *solving optimization independently*



**determinism** + **decomposability** = **tractable MAP**

E.g., for $\operatorname{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

turn sum into max units and

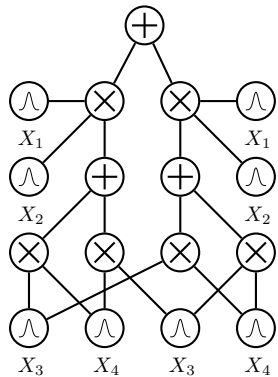
input distributions into max distributions

feedforward evaluation for

$\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$

retrieve max activations in backward pass

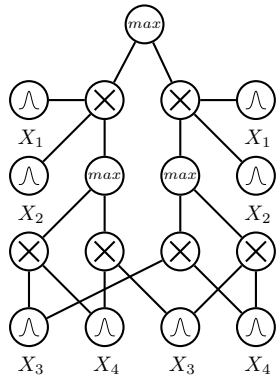
compute **MAP states** for $X_1$ and $X_3$ at input units



**determinism** + **decomposability** = **tractable MAP**

E.g., for $\operatorname{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

- turn sum into max units and
  input distributions into max distributions
- feedforward evaluation for
  $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
- retrieve max activations in backward pass
- compute **MAP states** for $X_1$ and $X_3$ at input units



**determinism** + **decomposability** = **tractable MAP**

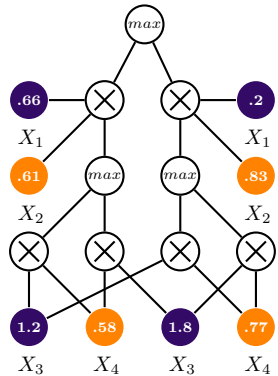
E.g., for $\text{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

turn sum into max units and
input distributions into max distributions

feedforward evaluation for
$\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$

retrieve max activations in backward pass

compute **MAP states** for $X_1$ and $X_3$ at input units



**determinism** + **decomposability** = **tractable MAP**

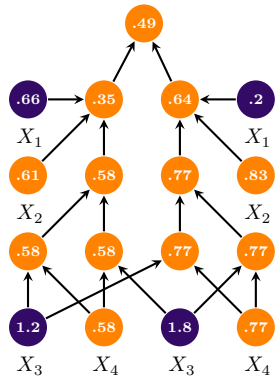
E.g., for $\text{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

turn sum into max units and
input distributions into max distributions

feedforward evaluation for
$\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$

retrieve max activations in backward pass

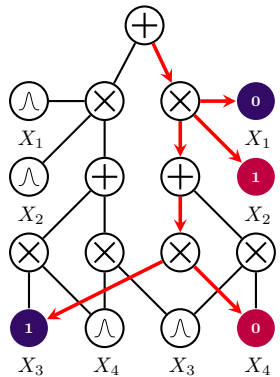
compute **MAP states** for $X_1$ and $X_3$ at input units



**determinism** + **decomposability** = **tractable MAP**

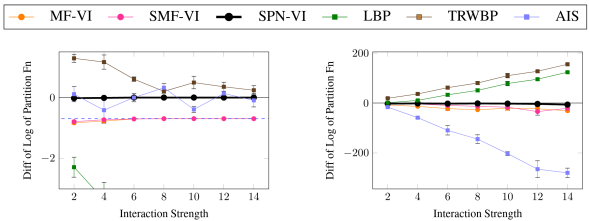
E.g., for $\operatorname{argmax}_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$:

- turn sum into max units and
  input distributions into max distributions
- feedforward evaluation for
  $\max_{x_1,x_3} p(x_1, x_3 \mid x_2, x_4)$
- retrieve max activations in backward pass
- compute **MAP states** for $X_1$ and $X_3$ at input units



**Determinism** + **decomposability** = **tractable ELBO**

Using deterministic and decomposable PCs as expressive variational family $\mathcal{Q}$ for discrete polynomial log-densities, i.e. $\operatorname{argmax}_{q \in \mathcal{Q}} \mathbb{E}_{\mathbf{x} \sim q} [\log w(\mathbf{x})] + \mathbb{H}(q)$




Closed-form computation for the entropy $\mathbb{H}$ *[**Vergari** et al. 2021]*

*Shih et al., "Probabilistic Circuits for Variational Inference in Discrete Graphical Models", NeurIPS, 2020*

## Goal

*all this inference is cool...*

*but how can we learn these circuits?*

# *learning probabilistic circuits*

Probabilistic circuits are (peculiar) neural networks...

# *learning probabilistic circuits*

Probabilistic circuits are (peculiar) neural networks... ***just backprop with SGD!***

***...end of Learning section***!

# *learning probabilistic circuits*

Probabilistic circuits are (peculiar) neural networks... ***just backprop with SGD!***

***wait but...***

*SGD is slow to converge...can we do better?*

*How to learn normalized weights?*

*How to enforce and exploit structural properties?*

# Learning probabilistic circuits

| | *Parameters* | *Structure* |
|---|---|---|
| *Generative* | ? | ? |
| *Discriminative* | ? | ? |

# Maximum likelihood

*the go-to objective in ProbML*

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and your parametric model $p_\theta(\mathbf{X})$ solve

$$\hat{\theta}_{\mathsf{ML}} = \max_\theta \prod_{i=1}^N p_\theta(\mathbf{x}^{(i)}) = \min_\theta -\sum_{i=1}^N \log p_\theta(\mathbf{x}^{(i)})$$

$\Rightarrow$ *minimize the negative log-likelihood (NLL)*

# *which parameters?*

*how to reparameterize circuits*

**Input distributions.**

**Sum unit parameters.**

# *which parameters?*

*how to reparameterize circuits*

**Input distributions.** Each input can be a different parametric distribution

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

**Sum unit parameters.**

# which parameters?

*how to reparameterize circuits*

**Input distributions.** Each input can be a different parametric distribution

**Sum unit parameters.** Enforce them to be non-negative, i.e., $w_i \geq 0$ but unnormalized

$$w_i = \exp(\alpha_i), \quad \alpha_i \in \mathbb{R}, \quad i = 1, \ldots, K$$

and renormalize the loss

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

or just renormalize the weights, i.e., $\sum_i w_i = 1$

$$\mathbf{w} = \mathsf{softmax}(\boldsymbol{\alpha}), \quad \boldsymbol{\alpha} \in \mathbb{R}^K$$

# learning input distributions

*As simple as tossing a coin*


The simplest PC: a single input distribution $p_{\mathrm{L}}$ with parameters $\boldsymbol{\theta}$

$\Longrightarrow$ *maximum likelihood (ML) estimation over data* $\mathcal{D}$

# learning input distributions

*As simple as tossing a coin*


The simplest PC: a single input distribution $p_{\mathsf{L}}$ with parameters $\boldsymbol{\theta}$

$\Rightarrow$ *maximum likelihood (ML) estimation over data* $\mathcal{D}$

E.g. Bernoulli with parameter $\theta$

$$\hat{\theta}_{\mathsf{ML}} = \frac{\sum_{x \in \mathcal{D}} \mathbb{1}[x = 1] + \alpha}{|\mathcal{D}| + 2\alpha} \quad \Rightarrow \text{ Laplace smoothing}$$

# learning input distributions

*General case: still simple*

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are ***exponential families*** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

# learning input distributions

*General case: still simple*

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are ***exponential families*** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

Where:

- $A(\boldsymbol{\theta})$: log-normalizer
- $\boldsymbol{h}(\mathbf{x})$ base-measure
- $\boldsymbol{T}(\mathbf{x})$ sufficient statistics
- $\boldsymbol{\theta}$ natural parameters

# learning input distributions

*General case: still simple*

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are ***exponential families*** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

Where:

$A(\boldsymbol{\theta})$: log-normalizer

$\boldsymbol{h}(\mathbf{x})$ base-measure

$\boldsymbol{T}(\mathbf{x})$ sufficient statistics

$\boldsymbol{\theta}$ natural parameters

or $\boldsymbol{\phi}$ expectation parameters — 1:1 mapping with $\boldsymbol{\theta} \Longrightarrow \boldsymbol{\theta} = \boldsymbol{\theta}(\boldsymbol{\phi})$

# learning input distributions

*General case: still simple*

Bernoulli, Gaussian, Dirichlet, Poisson, Gamma are ***exponential families*** of the form:

$$p_{\mathsf{L}}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\ \exp(\boldsymbol{T}(\mathbf{x})^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))$$

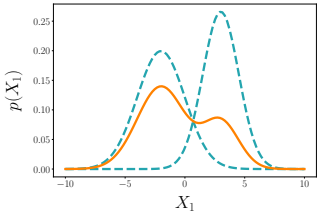
Maximum likelihood estimation is still "***counting***":

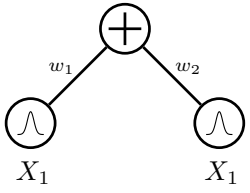
$$\hat{\boldsymbol{\phi}}_{\mathsf{ML}} = \mathbb{E}_{\mathcal{D}}[\boldsymbol{T}(\mathbf{x})] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \boldsymbol{T}(\mathbf{x})$$

$$\hat{\boldsymbol{\theta}}_{\mathsf{ML}} = \boldsymbol{\theta}(\hat{\boldsymbol{\phi}}_{\mathsf{ML}})$$

# *a single sum unit*

*the simplest "real" PC*



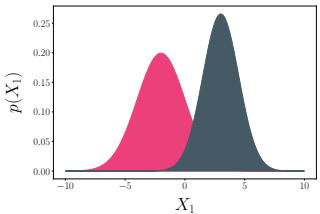



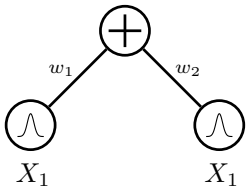

Recall that sum units represent ***mixture models***:

$$p_{\mathsf{S}}(\mathbf{x}) = \sum_{k=1}^{K} w_k p_{\mathsf{L}_k}(\mathbf{x})$$

# a single sum unit

*the simplest "real" PC*



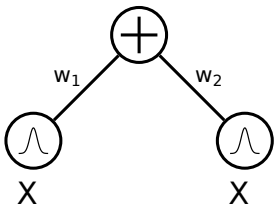




Recall that sum units represent ***latent variable models***:

$$p_{\mathsf{S}}(\mathbf{x}) = \sum_{k=1}^{K} p(Z = k)p(\mathbf{x} \mid Z = k)$$

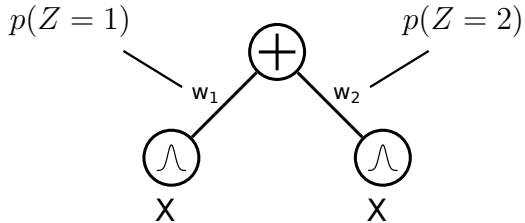
# Expectation-Maximization

*for deep mixtures/PCs*



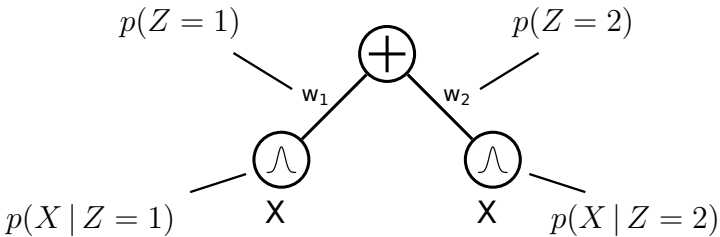
# Expectation-Maximization

*for deep mixtures/PCs*



# Expectation-Maximization

*for deep mixtures/PCs*



# Expectation-Maximization

*for shallow mixtures/PCs*

ML if $Z$ was observed:

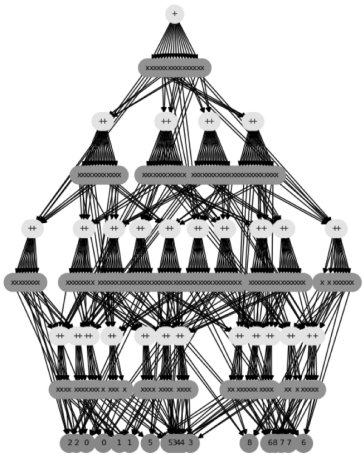
$$\hat{w}_k = \frac{\sum_{z \in \mathcal{D}} \mathbb{1}[z = k]}{|\mathcal{D}|} \qquad \hat{\boldsymbol{\phi}}_k = \frac{\sum_{\mathbf{x}, z \in \mathcal{D}} \mathbb{1}[z = k] T(\mathbf{x})}{\sum_{z \in \mathcal{D}} \mathbb{1}[z = k]}$$

$Z$ is unobserved—but we have $p(Z = k \mid \mathbf{x}) \propto w_k \, \mathsf{L}_k(\mathbf{x})$.

$$w_k^{new} = \frac{\sum_{\mathbf{x} \in \mathcal{D}} p(Z = k \mid \mathbf{x})}{|\mathcal{D}|} \qquad \boldsymbol{\phi}_k^{new} = \frac{\sum_{\mathbf{x}, z \in \mathcal{D}} p(Z = k \mid \mathbf{x}) T(\mathbf{x})}{\sum_{z \in \mathcal{D}} p(Z = k \mid \mathbf{x})}$$

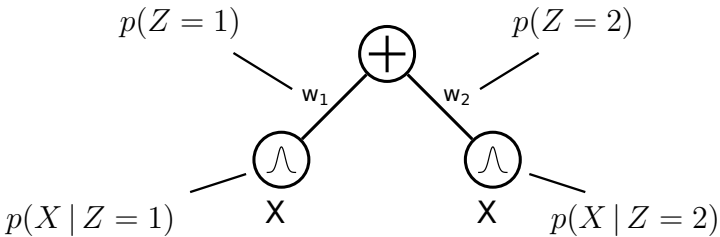
*how to generalize...*

*EM to deep mixtures/circuits?*



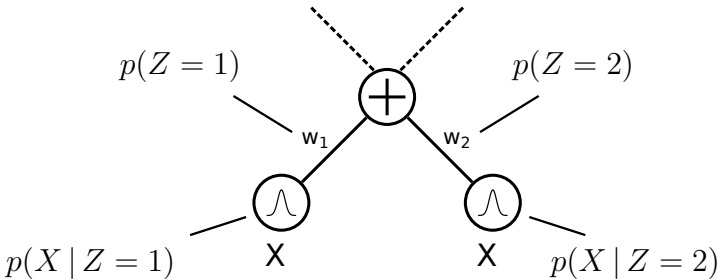
# Expectation-Maximization

*for deep mixtures/PCs*



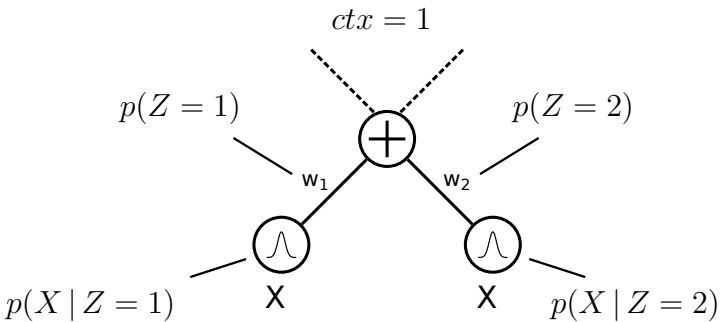
# Expectation-Maximization

*for deep mixtures/PCs*



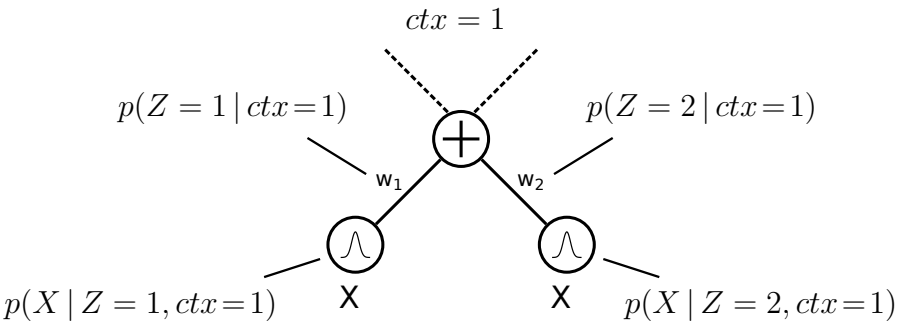
# Expectation-Maximization

*for deep mixtures/PCs*



# Expectation-Maximization

*for deep mixtures/PCs*



# Expectation-Maximization

*for smooth and decomposable circuits*

$$w_{i,j}^{new} \leftarrow \frac{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}]}{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1 \mid \mathbf{x}; \mathbf{w}^{old}]}$$

*Darwiche, "A Differential Approach to Inference in Bayesian Networks", , 2003*
*Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", , 2016*

# Expectation-Maximization

*for smooth and decomposable circuits*

$$w_{i,j}^{new} \leftarrow \frac{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}]}{\sum_{\mathbf{x}\in\mathcal{D}} p[ctx_i = 1 \mid \mathbf{x}; \mathbf{w}^{old}]}$$

We get **all** the required statistics with a single backprop pass:

$$p[ctx_i = 1, Z_i = j \mid \mathbf{x}; \mathbf{w}^{old}] = \frac{1}{p(\mathbf{x})} \frac{\partial p(\mathbf{x})}{\partial \mathsf{S}_i(\mathbf{x})} \mathsf{N}_j(\mathbf{x}) w_{i,j}^{old}$$

---

*Darwiche, "A Differential Approach to Inference in Bayesian Networks", , 2003*
*Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", , 2016*

# Bayesian parameter learning

Formulate a prior $p(\mathbf{w}, \boldsymbol{\theta})$ over sum-weights and parameters of input units. Then perform posterior inference:

$$p(\mathbf{w}, \boldsymbol{\theta}|\mathcal{D}) \propto p(\mathbf{w}, \boldsymbol{\theta})\, p(\mathcal{D}|\mathbf{w}, \boldsymbol{\theta})$$

Moment matching (oBMM) *[Jaini et al. 2016; Rashwan et al. 2016]*

Collapsed variational inference algorithm *[Zhao et al. 2016a]*

Gibbs sampling *[Trapp et al. 2019; Vergari et al. 2019]*

# Learning probabilistic circuits

| | Parameters | Structure |
|---|---|---|
| Generative | ***deterministic***<br>closed-form MLE *[Kisa et al. 2014; Peharz et al. 2014]*<br>***non-deterministic***<br>EM *[Poon et al. 2011; Peharz 2015; Zhao et al. 2016b]*<br>SGD *[Sharir et al. 2016; Peharz et al. 2019b]*<br>Bayesian *[Jaini et al. 2016; Rashwan et al. 2016]*<br>*[Zhao et al. 2016a; Trapp et al. 2019; Vergari et al. 2019]* | ? |
| Discriminative | ? | ? |

# Learning probabilistic circuits

| | Parameters | Structure |
|---|---|---|
| Generative | ***deterministic***<br>closed-form MLE *[Kisa et al. 2014; Peharz et al. 2014]*<br>***non-deterministic***<br>EM *[Poon et al. 2011; Peharz 2015; Zhao et al. 2016b]*<br>SGD *[Sharir et al. 2016; Peharz et al. 2019b]*<br>Bayesian *[Jaini et al. 2016; Rashwan et al. 2016]*<br>*[Zhao et al. 2016a; Trapp et al. 2019; Vergari et al. 2019]* | ***greedy***<br>top-down *[Gens et al. 2013; Rahman et al. 2014; Rooshenas et al. 2014]*<br>*[Vergari et al. 2015]* bottom-up *[Peharz et al. 2013]*<br>***hill climbing*** *[Lowd et al. 2008, 2013; Peharz et al. 2014]*<br>*[Dennis et al. 2015; Liang et al. 2017; Galindez Olascoaga et al. 2019]*<br>*[Dang et al. 2022]*<br>***random*** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di Mauro et al. 2017]* |
| Discriminative | ? | ? |

# Learning probabilistic circuits

| | Parameters | Structure |
|---|---|---|
| Generative | ***deterministic***<br>closed-form MLE *[Kisa et al. 2014; Peharz et al. 2014]*<br>***non-deterministic***<br>EM *[Poon et al. 2011; Peharz 2015; Zhao et al. 2016b]*<br>SGD *[Sharir et al. 2016; Peharz et al. 2019b]*<br>Bayesian *[Jaini et al. 2016; Rashwan et al. 2016]*<br>*[Zhao et al. 2016a; Trapp et al. 2019; Vergari et al. 2019]* | ***greedy***<br>top-down *[Gens et al. 2013; Rahman et al. 2014; Rooshenas et al. 2014]*<br>*[Vergar*<br>***hill clim***<br>*[Dennis* *al. 2019]*<br>*[Dang et al. 2022]*<br>***random*** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di Mauro et al. 2017]* |
| Discriminative | ? | ? |

**a single pipeline**

# Probabilistic Circuits (PCs)

*the unit-wise definition*

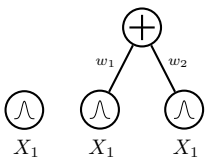
I. *A simple tractable function is a circuit*

# Probabilistic Circuits (PCs)

*the unit-wise definition*

I. *A simple tractable function is a circuit*

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



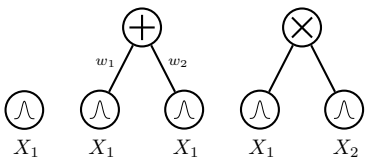
# Probabilistic Circuits (PCs)

*the unit-wise definition*

I. *A simple tractable function is a circuit*

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

III. *A product of circuits is a circuit*



# Probabilistic Circuits (PCs)

*the layer-wise definition*

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

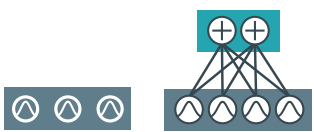
# Probabilistic Circuits (PCs)

*the layer-wise definition*

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

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

$$c(\mathbf{x}) = \mathbf{W}\boldsymbol{l}(\mathbf{x})$$



# Probabilistic Circuits (PCs)

*the layer-wise definition*

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

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

$$c(\mathbf{x}) = \mathbf{W}\boldsymbol{l}(\mathbf{x})$$

# Probabilistic Circuits (PCs)

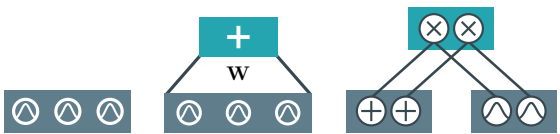
*the layer-wise definition*

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

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

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

$$c(\mathbf{x}) = \boldsymbol{l}(\mathbf{x}) \odot \boldsymbol{r}(\mathbf{x}) \qquad \text{// Hadamard}$$



# Probabilistic Circuits (PCs)

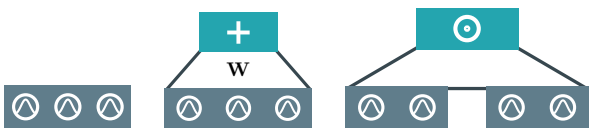
*the layer-wise definition*

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

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

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

$$c(\mathbf{x}) = \boldsymbol{l}(\mathbf{x}) \odot \boldsymbol{r}(\mathbf{x}) \qquad \text{// Hadamard}$$



# Probabilistic Circuits (PCs)

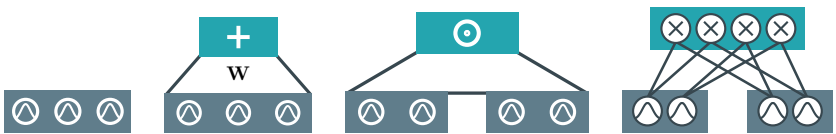
*the layer-wise definition*

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

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

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

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



# Probabilistic Circuits (PCs)

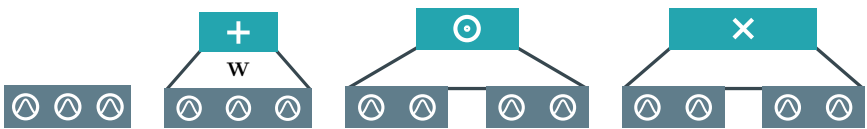
*the layer-wise definition*

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

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

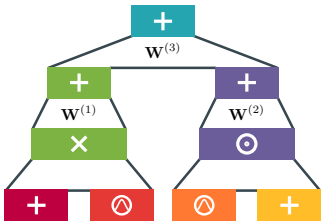
III. *The product of two layers is a circuit layer*

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



# Probabilistic Circuits (PCs)

*the layer-wise definition*



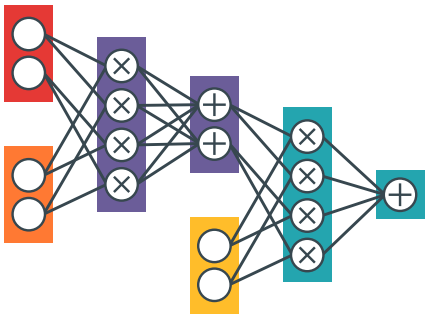

I. *A set of tractable functions is a circuit layer*

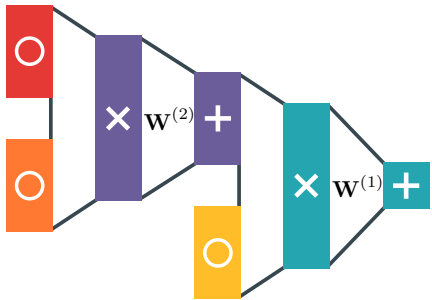
II. *A linear projection of a layer is a circuit layer*

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

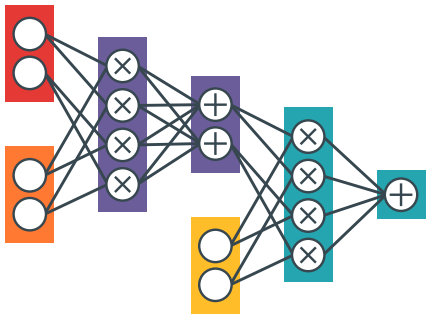
***stack layers to build a deep circuit!***

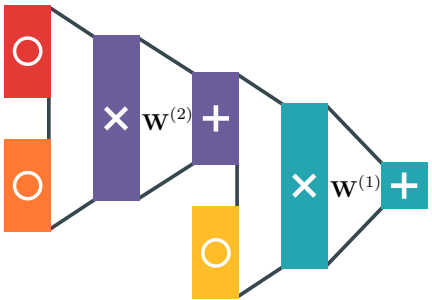
# Abstracting layers





# Abstracting layers

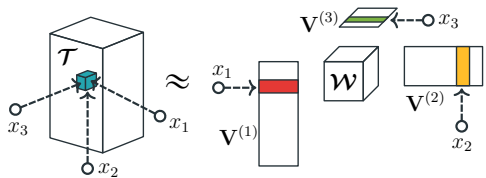


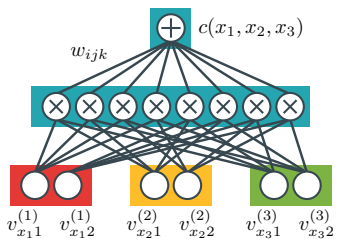



*combine sum-product layers together to create new layers*

# circuits layers

*as tensor factorizations*

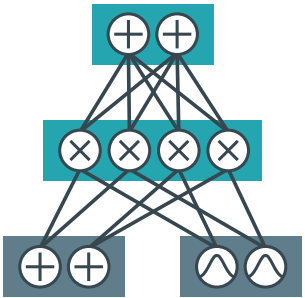


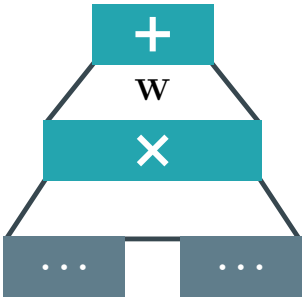




*Mari et al., "What is the Relationship between Circuits and Tensor Factorizations (and How Can We Exploit it)?", Under submission, 2024*

# more layers

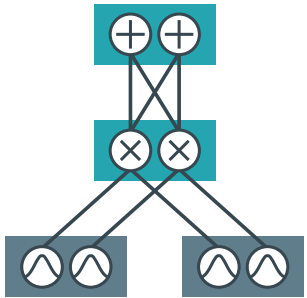
*Tucker decomposition*

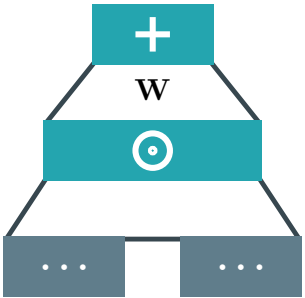




# more layers

*Candecomp Parafac (CP) decomposition*



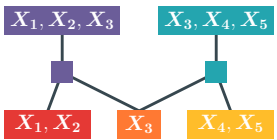


# *more layers*


**CP** layer

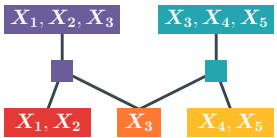

**Tucker** layer

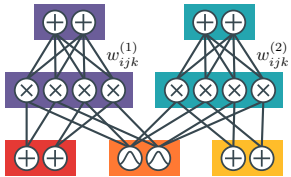
# Learning recipe




**1)** Build a ***region graph***

# Learning recipe



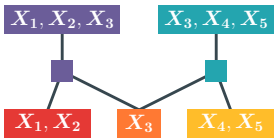



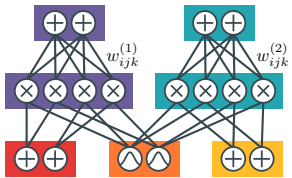

**1)** Build a ***region graph***

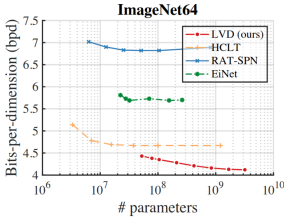
**2)** Overparameterize

***2.1) pick a (composite) layer type***
***2.2) choose how many units per layer***

# Learning recipe



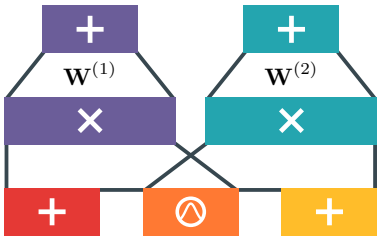



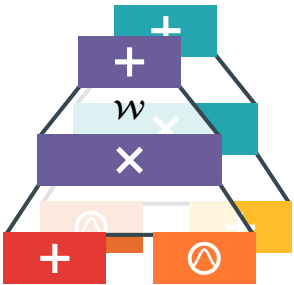

**1)** Build a ***region graph***

**2)** Overparameterize

**3)** Learn parameters

# folding layers



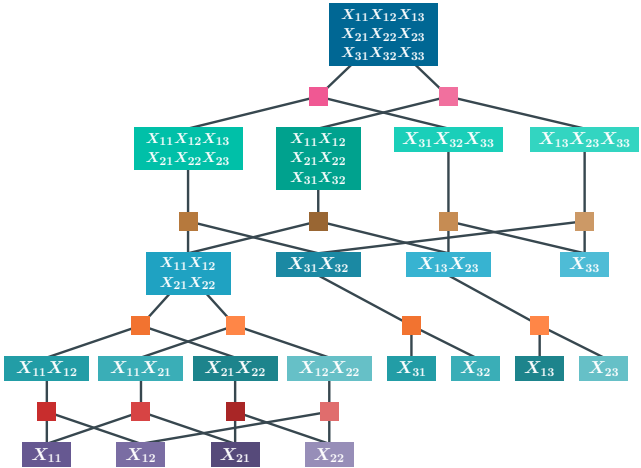

*stack layers to increase GPU parallelism*

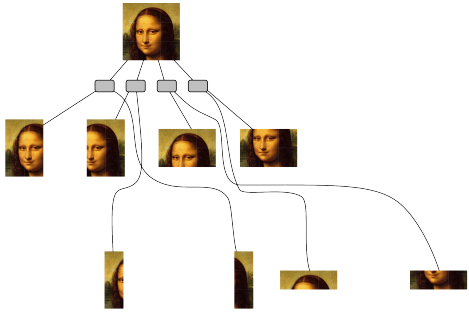
# building circuits

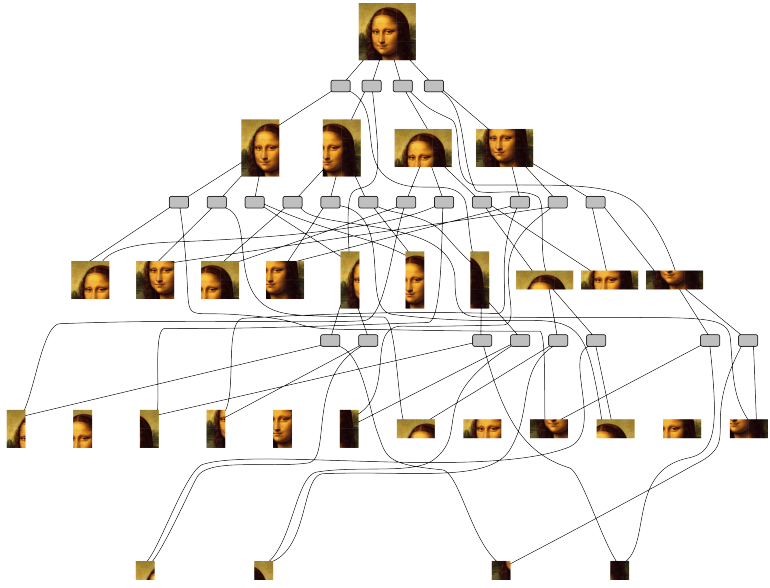
*mix&match RGs and layers*

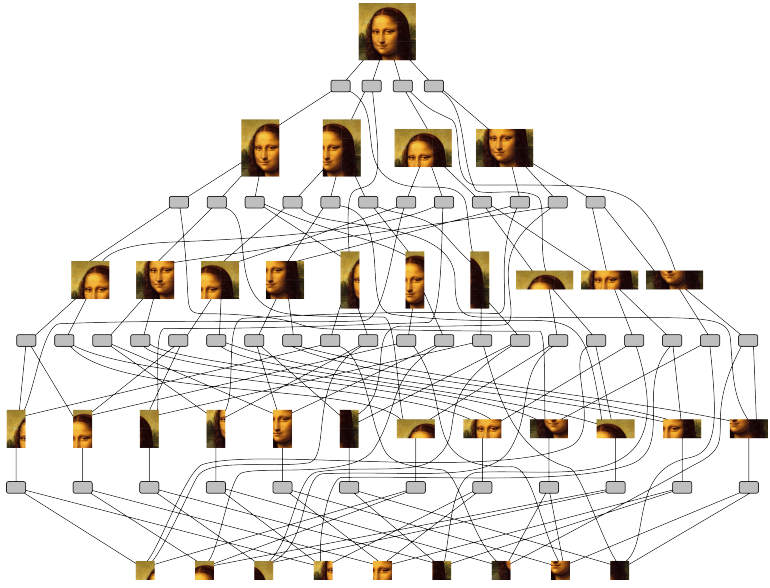
| PC Architecture | Region Graph | Sum-Product Layer | Fold |
|---|---|---|---|
| Poon&Domingos (Poon & Domingos, 2011) | PD | CP | ✗ |
| RAT-SPN (Peharz et al., 2020b) | RND | Tucker | ✗ |
| EiNet (Peharz et al., 2020a) | { RND, PD } | Tucker | ✓ |
| HCLT (Liu & Van den Broeck, 2021) | CL | CP | ✗ |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

# which region graph?





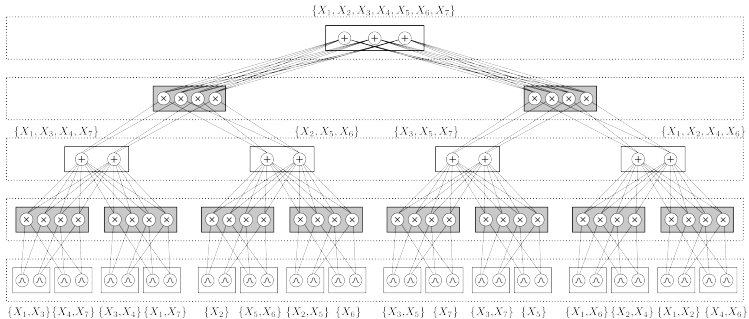




# random regions graphs

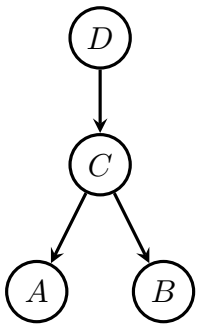
*The "no-learning" option* *[Peharz et al. 2019a]*

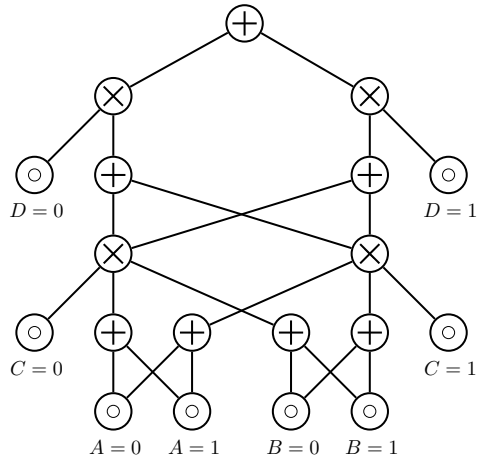
Generating a random region graph, by recursively splitting $\mathbf{X}$ into two random parts:



# RGs from PGMs

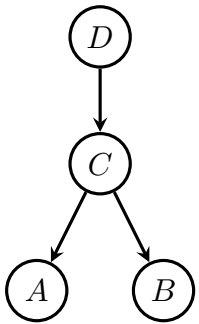
*via compilation*

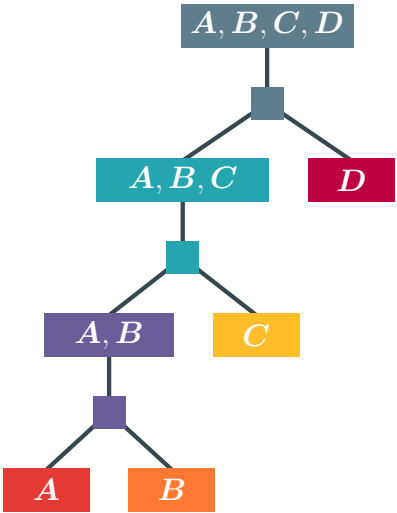


# RGs from PGMs

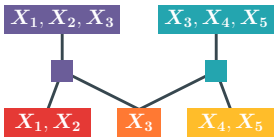
*via compilation*



# *learning recipe*

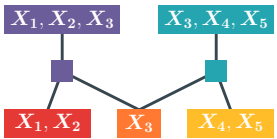
*recap*

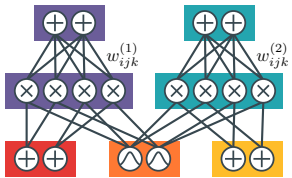



**1)** Build a ***region graph***

# learning recipe

*recap*



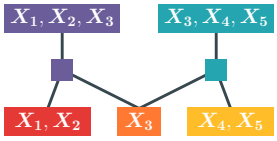



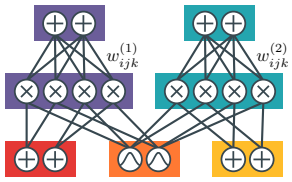

**1)** Build a ***region graph***

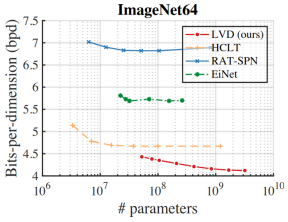
**2)** Overparameterize

# learning recipe

*recap*




**1)** Build a ***region graph***

**2)** Overparameterize

**3)** Learn parameters

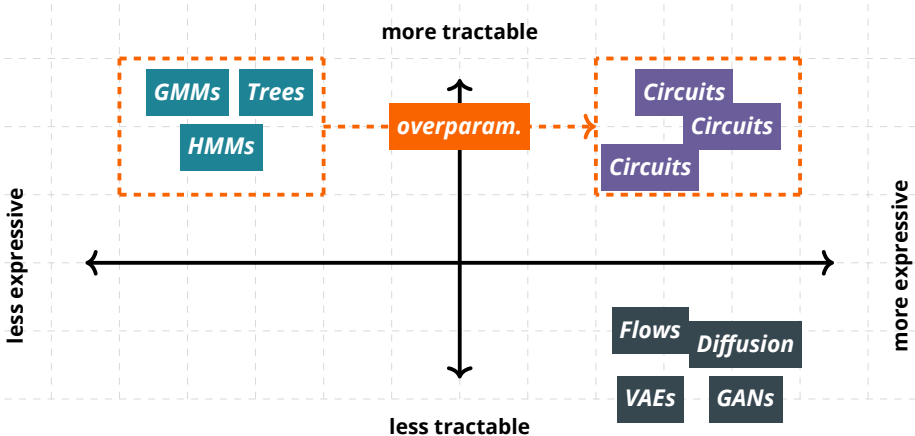
# learning probabilistic circuits

| | Parameters | Structure |
|---|---|---|
| Generative | ***deterministic***<br>closed-form MLE *[Kisa et al. 2014; Peharz et al. 2014]*<br>***non-deterministic***<br>EM *[Poon et al. 2011; Peharz 2015; Zhao et al. 2016b]*<br>SGD *[Sharir et al. 2016; Peharz et al. 2019b]*<br>Bayesian *[Jaini et al. 2016; Rashwan et al. 2016]*<br>*[Zhao et al. 2016a; Trapp et al. 2019; Vergari et al. 2019]* | ***greedy***<br>top-down *[Gens et al. 2013; Rahman et al. 2014; Rooshenas et al. 2014]*<br>*[Vergar*<br>***hill clim***<br>*[Dennis* *al. 2019]*<br>*[Dang et al. 2022]*<br>***random*** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di Mauro et al. 2017]* |
| Discriminative | ? | ? |

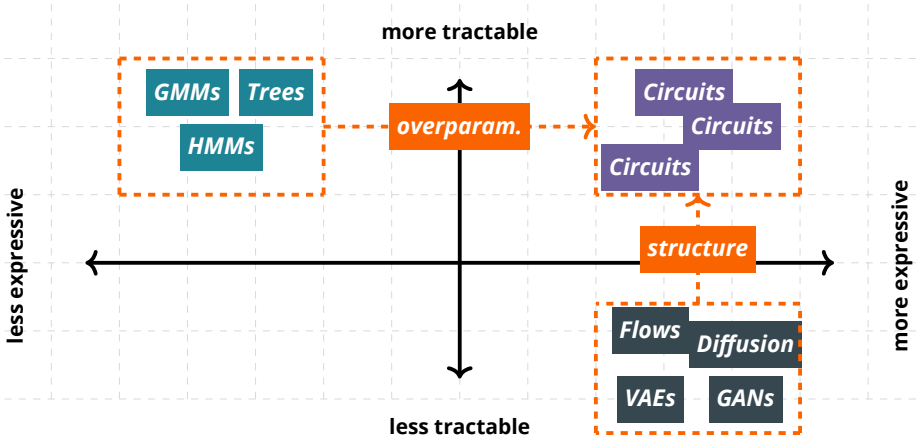
**a single pipeline**

# learning probabilistic circuits

| | Parameters | Structure |
|---|---|---|
| ***Generative*** | ***deterministic***<br>closed-form MLE *[Kisa et al. 2014; Peharz et al. 2014]*<br>***non-deterministic***<br>EM *[Poon et al. 2011; Peharz 2015] [Zhao et al. 2016b]*<br>SGD *[Sharir et al. 2016; Peharz et al. 2019b]*<br>Bayesian *[Jaini et al. 2016; Rashwan et al. 2016] [Zhao et al. 2016a; Trapp et al. 2019; Vergari et al. 2019]* | ***greedy***<br>top-down *[Gens et al. 2013; Rahman et al. 2014; Rooshenas et al. 2014] [Vergari et al. 2015]* bottom-up *[Peharz et al. 2013]*<br>***hill climbing*** *[Lowd et al. 2008, 2013; Peharz et al. 2014] [Dennis et al. 2015; Liang et al. 2017; Galindez Olascoaga et al. 2019] [Dang et al. 2022]*<br>***random*** RAT-SPNs *[Peharz et al. 2019b]* XCNet *[Di Mauro et al. 2017]* |
| ***Discriminative*** | ***deterministic***<br>convex-opt MLE *[Liang et al. 2019]*<br>***non-deterministic***<br>EM *[Rashwan et al. 2018]*<br>SGD *[Gens et al. 2012; Sharir et al. 2016] [Peharz et al. 2019b]* | ***greedy***<br>top-down *[Shao et al. 2020]*<br>***hill climbing*** *[Rooshenas et al. 2016; Liang et al. 2019]* |




***then make it more expressive!***




***impose structure!***