tractable probabilistic modeling with probabilistic circuits

antonio vergari (he/him)

@tetraduzione

16th Oct 2024 - **PIC PhD School** *Copenhagen*

april

`april-tools.github.io`

april

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

april

about
probabilities
integrals &
logic

april

april is
probably a
recursive
identifier of a
lab

deep generative models

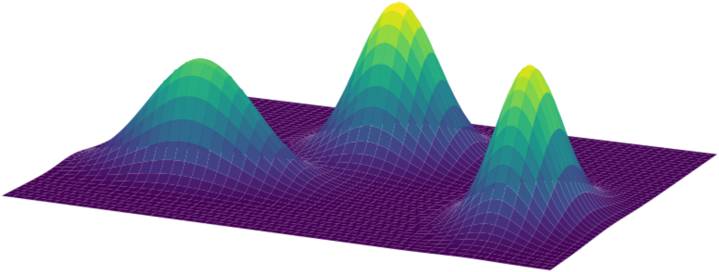
+

flexible and reliable
(logic &) probabilistic reasoning?

i) probabilistic circuits: syntax and semantics

ii) reliable and efficient neuro-symbolic AI

iii) beyond PCs: subtractive mixture models



a love letter to mixture models...

image taken from Hao Tang's course on ASR

Reasoning about ML models

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

q_2 *“How **fair** is the prediction is a certain protected attribute changes?”*

q_3 *“Can we certify no **adversarial examples** exist?”*

Reasoning about ML models

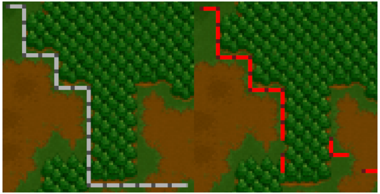
q1 $\int p(\mathbf{x}_o, \mathbf{x}_m) d\mathbf{X}_m$
(missing values)

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

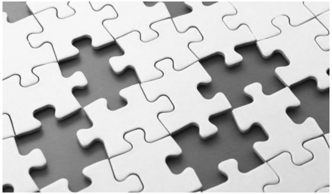
q3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$
(adversarial robust.)

...in the language of probabilities

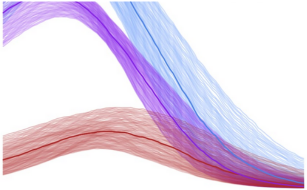
more complex reasoning



neuro-symbolic AI



probabilistic programming



computing uncertainties (Bayesian inference)

...and more application scenarios

Reasoning about ML models

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

(missing values)

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

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

hard to compute in general!

Reasoning about ML models

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

(missing values)

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

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)} \left[f(\mathbf{x} + \mathbf{e}) \right]$

(adversarial robust.)

it is crucial we compute them exactly and in polytime!

Reasoning about ML models

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

(missing values)

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

(fairness)

q_3 $\mathbb{E}_{\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)}\left[f(\mathbf{x} + \mathbf{e})\right]$

(adversarial robust.)

it is crucial we compute them tractably!

Tractable Probabilistic Inference

A class of queries $\mathcal{Q}$ *is tractable on a family of probabilistic models* $\mathcal{M}$
iff for any query $\mathbf{q} \in \mathcal{Q}$ *and model* $\mathbf{m} \in \mathcal{M}$
exactly *computing* $\mathbf{q}(\mathbf{m})$ *runs in time* $O(\mathrm{poly}(|\mathbf{m}|))$.

$\implies$ ***model-centric*** *definition...*

Tractable Probabilistic Inference

A class of queries $\mathcal{Q}$ is tractable on a family of probabilistic models $\mathcal{M}$
iff for any query $\mathbf{q} \in \mathcal{Q}$ and model $\mathbf{m} \in \mathcal{M}$
***exactly** computing $\mathbf{q}(\mathbf{m})$ runs in time $O(\text{poly}(|\mathbf{m}|))$.*

$\Longrightarrow$ ***model-centric** definition...*

$\Longrightarrow$ *...and **query-centric**: Tractability is not a universal property!*

Tractable Probabilistic Inference

A class of queries $\mathcal{Q}$ *is tractable on a family of probabilistic models* $\mathcal{M}$
iff for any query $\mathbf{q} \in \mathcal{Q}$ *and model* $\mathbf{m} \in \mathcal{M}$
exactly *computing* $\mathbf{q}(\mathbf{m})$ *runs in time* $O(\text{poly}(|\mathbf{m}|))$.

$\Longrightarrow$ ***model-centric*** *definition...*

$\Longrightarrow$ *...and* ***query-centric****: Tractability is not a universal property!*

$\Longrightarrow$ *often* poly *will in fact be* ***linear****!*

Tractable Probabilistic Inference

A class of queries $\mathcal{Q}$ *is tractable on a family of probabilistic models* $\mathcal{M}$
iff for any query $\mathbf{q} \in \mathcal{Q}$ *and model* $\mathbf{m} \in \mathcal{M}$
***exactly** computing* $\mathbf{q}(\mathbf{m})$ *runs in time* $O(\text{poly}(|\mathbf{m}|))$.

$\Rightarrow$ ***model-centric** definition...*

$\Rightarrow$ *...and **query-centric**: Tractability is not a universal property!*

$\Rightarrow$ *often* poly *will in fact be **linear**!*

$\Rightarrow$ *Note: if* $|\mathbf{m}| \in O(\text{poly}(|\mathbf{X}|))$*, then query time is* $O(\text{poly}(|\mathbf{X}|))$.

Tractable Probabilistic Inference

A class of queries $\mathcal{Q}$ *is tractable on a family of probabilistic models* $\mathcal{M}$
iff for any query $\mathbf{q} \in \mathcal{Q}$ *and model* $\mathbf{m} \in \mathcal{M}$
***exactly** computing* $\mathbf{q}(\mathbf{m})$ *runs in time* $O(\text{poly}(|\mathbf{m}|))$.

$\Rightarrow$ ***model-centric** definition...*

$\Rightarrow$ *...and **query-centric**: Tractability is not a universal property!*

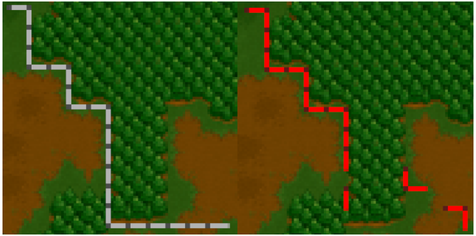
$\Rightarrow$ *often* poly *will in fact be **linear**!*

$\Rightarrow$ *Note: if* $|\mathbf{m}| \in O(\text{poly}(|\mathbf{X}|))$*, then query time is* $O(\text{poly}(|\mathbf{X}|))$.

$\Rightarrow$ *Why **exactness**? Highest guarantee possible!*

why tractable models?

exactness can be crucial in safety-driven applications



guarantee constraint satisfaction

[Ahmed et al. 2022]

estimation error is bounded (0)

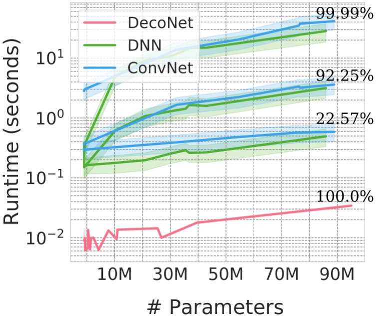
[Choi 2022]

why tractable models?

they can be much faster than intractable ones!

Method	MNIST (10,000 test images)		
	Theoretical bpd	Comp. bpd	En- & decoding time
PC (small)	1.26	1.30	**53**
PC (large)	**1.20**	**1.24**	168
IDF	1.90	1.96	880
BitSwap	1.27	1.31	904

[Liu, Mandt, and Broeck 2022]



[Subramani et al. 2021]

Why?

tractable inference

Why?

we always perform
tractable inference
over an approximate model!

VI

$$\min_{\boldsymbol{q}\in\mathcal{Q}} \mathsf{KL}(\boldsymbol{q}||p)$$

we pick a ***tractable*** variational distribution $\boldsymbol{q}$

$\Longrightarrow$ *e.g., Gaussian, GMM, HMM, flow, etc*

VI

$$\min_{\boldsymbol{q} \in \mathcal{Q}} \mathsf{KL}(\boldsymbol{q}||p)$$

we pick a ***tractable*** variational distribution $\boldsymbol{q}$

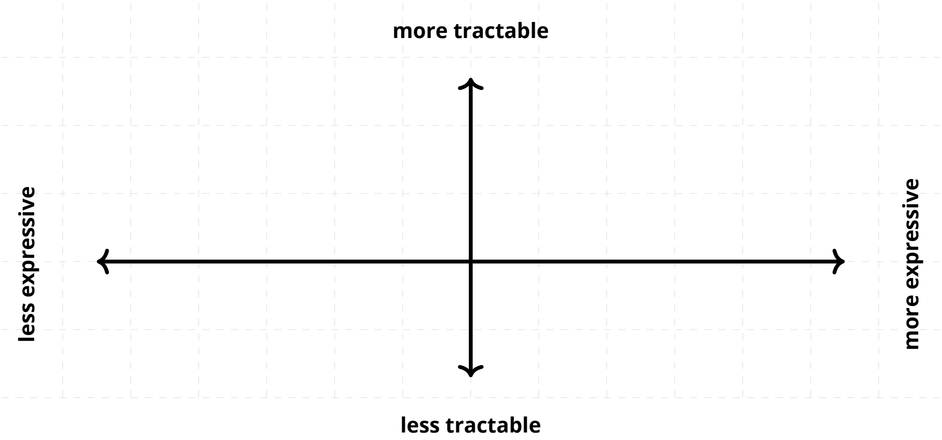
MC

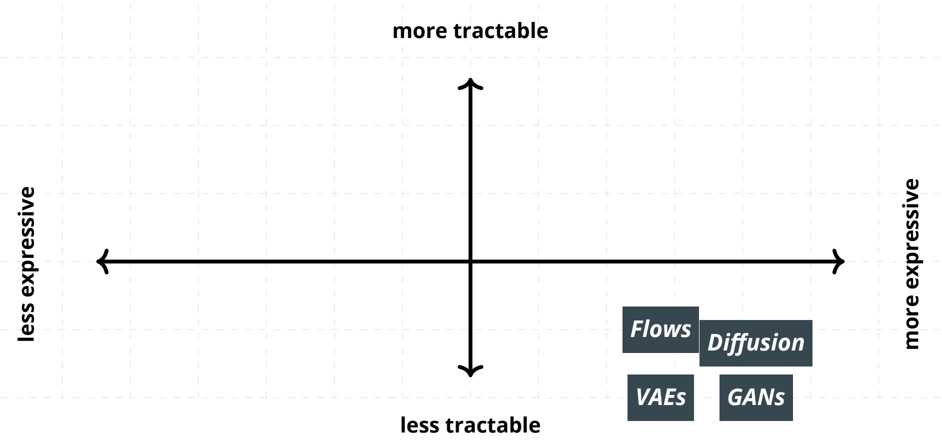
$$\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}^{(i)})$$

we turn an intractable integral into a ***tractable sum***

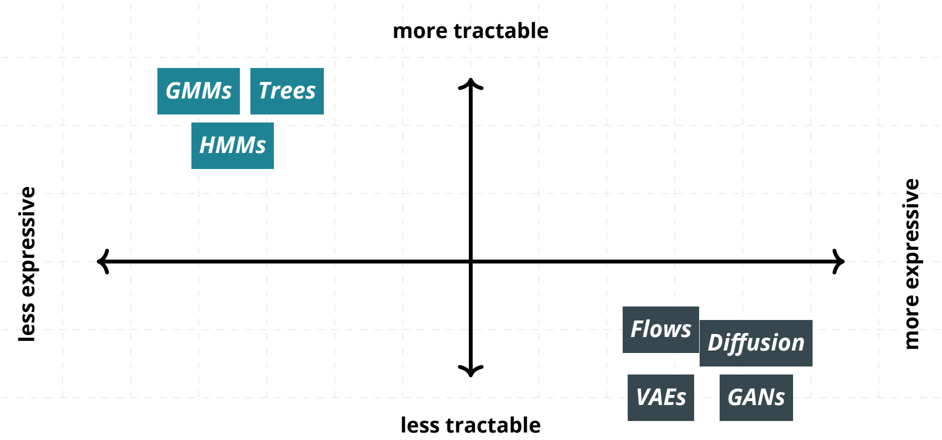
Goal

"Can we find
a middle ground
between
tractability and expressiveness?"

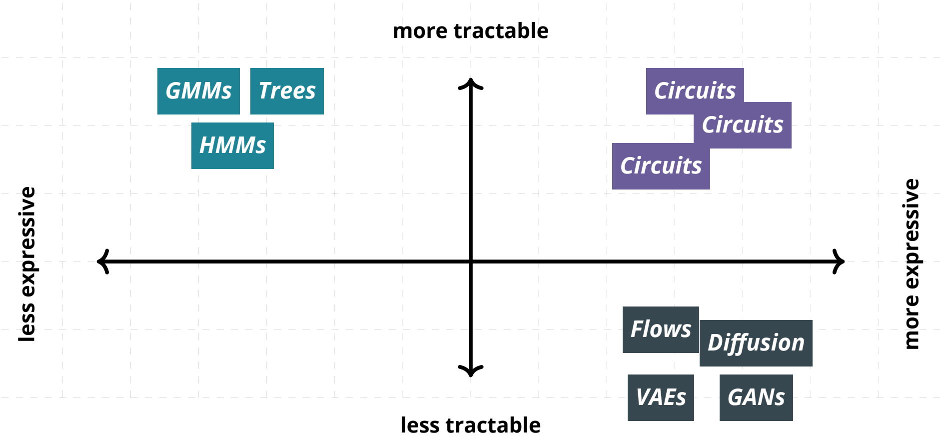

more tractable
less expressive
more expressive
less tractable



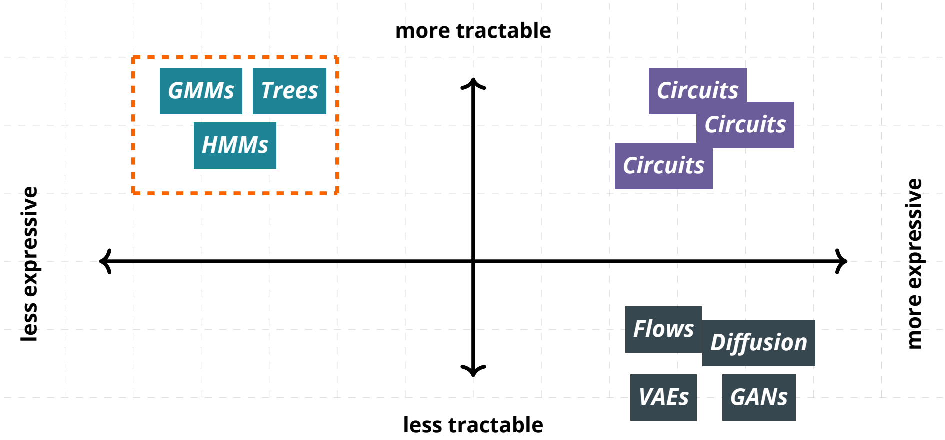
expressive models are not very tractable...



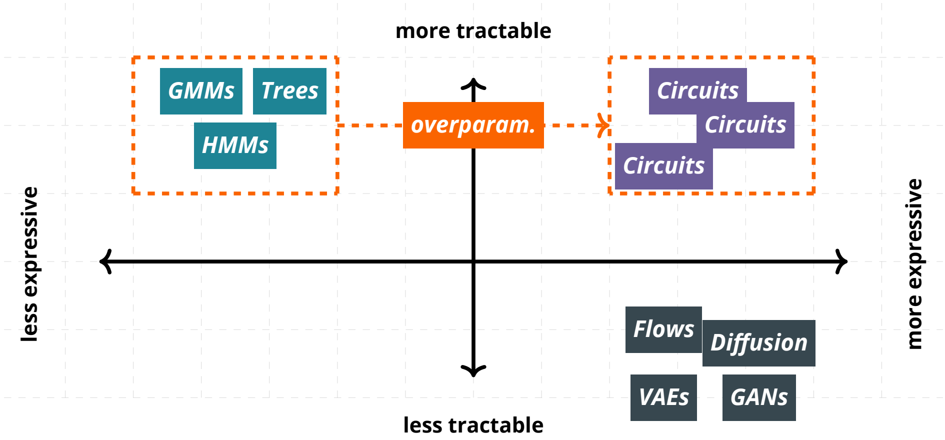
tractable models are not that expressive...



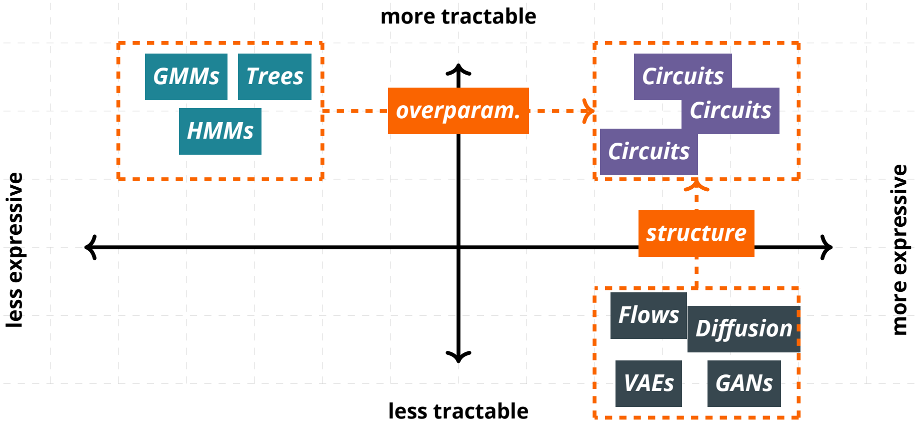
circuits can be both expressive and tractable!



start simple...



then make it more expressive!



impose structure!

Goal

"Can we design computational graphs that efficiently encode inference?"

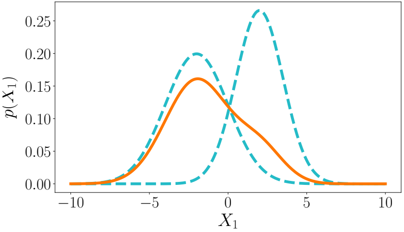
Goal

"Can we design computational graphs that efficiently encode inference?"

$\Rightarrow$ ***yes! with circuits!***

GMMs

as computational graphs

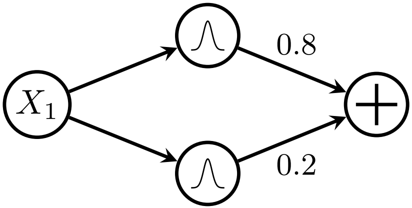


$$p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

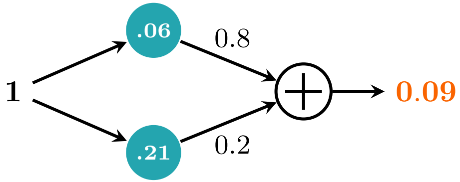


$$p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1)$$

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

GMMs

as computational graphs

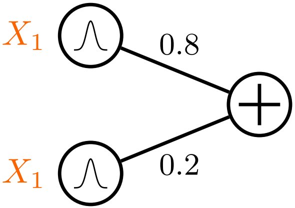


$$p(X=1) = 0.2 \cdot p_1(X_1=1) + 0.8 \cdot p_2(X_1=1)$$

$\Longrightarrow$ *inference = feedforward evaluation*

GMMs

as computational graphs



A simplified notation:

$\Rightarrow$ ***scopes*** *attached to inputs*

$\Rightarrow$ *edge directions omitted*

GMMs

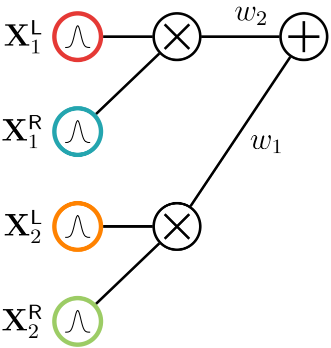
as computational graphs

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

$\Longrightarrow$ *local factorizations...*

GMMs

as computational graphs



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

$\Rightarrow$ *...are product units*

Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

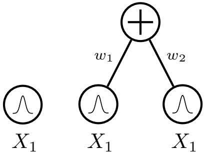
I. *A simple tractable function is a circuit*

Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

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



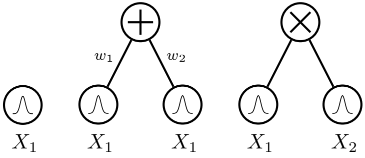
Probabilistic Circuits (PCs)

A grammar for tractable computational graphs

I. *A simple tractable function is a circuit*

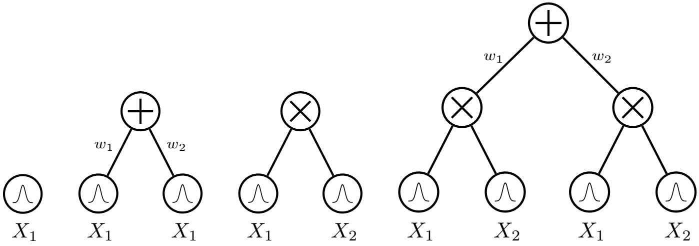
II. *A weighted combination of circuits is a circuit*

III. *A product of circuits is a circuit*



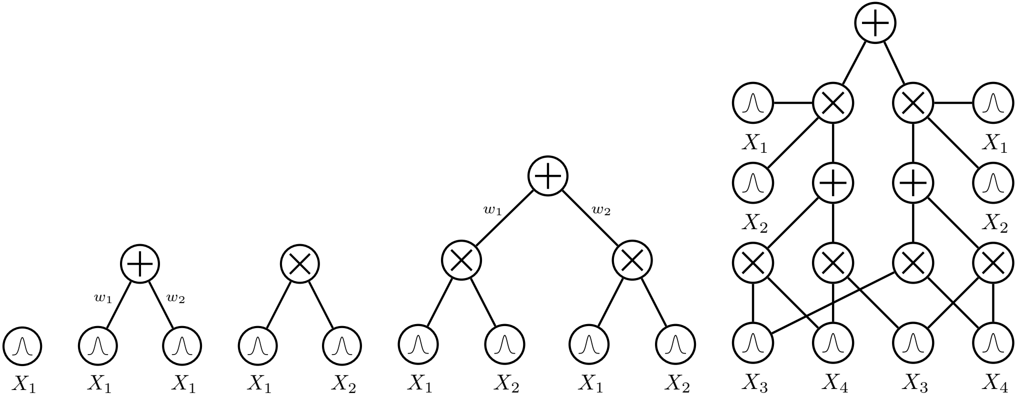
Probabilistic Circuits (PCs)

A grammar for tractable computational graphs



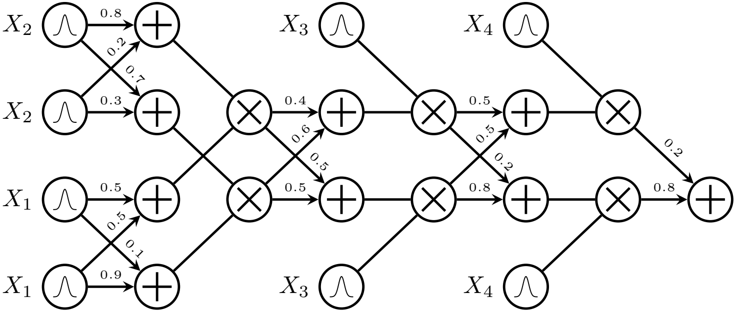
Probabilistic Circuits (PCs)

A grammar for tractable computational graphs



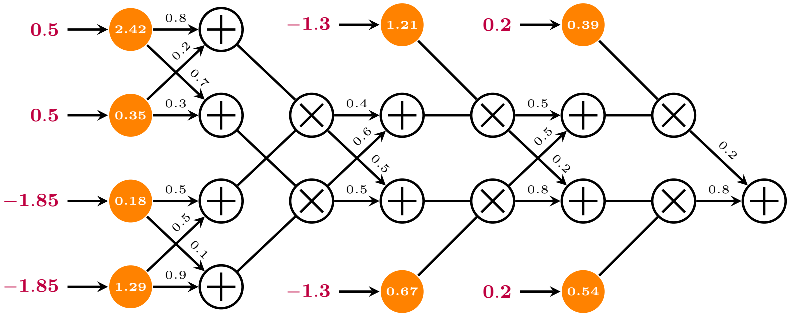
Probabilistic queries = feedforward evaluation

$$p(X_1 = \mathbf{-1.85}, X_2 = \mathbf{0.5}, X_3 = \mathbf{-1.3}, X_4 = \mathbf{0.2})$$



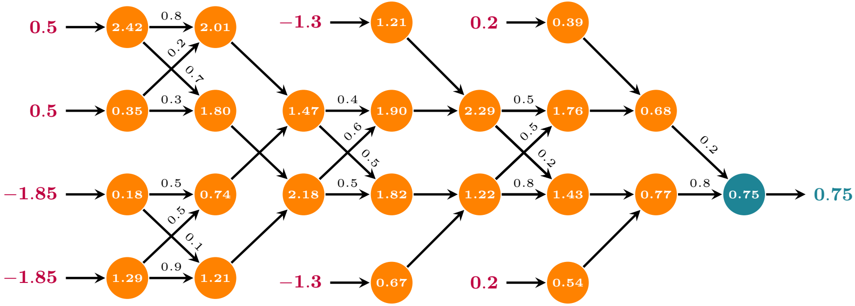
Probabilistic queries = feedforward evaluation

$$p(X_1 = -\mathbf{1.85}, X_2 = \mathbf{0.5}, X_3 = -\mathbf{1.3}, X_4 = \mathbf{0.2})$$



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

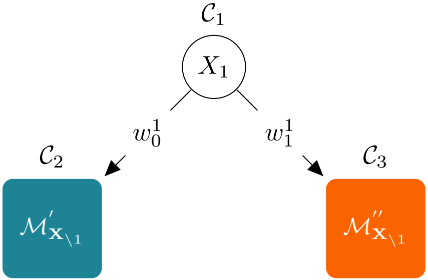
One formalism to represent many probabilistic and logical models

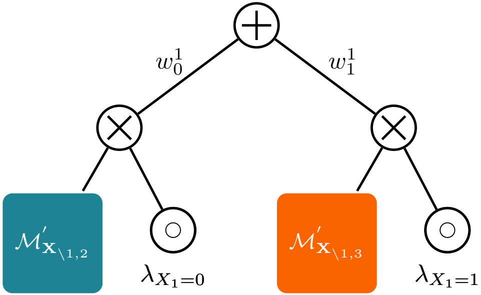
$\Longrightarrow$ *#HMMs #Trees #XGBoost, Tensor Networks, ...*

O/BDDs, *SDDs* and other *PGMs*...

decision trees

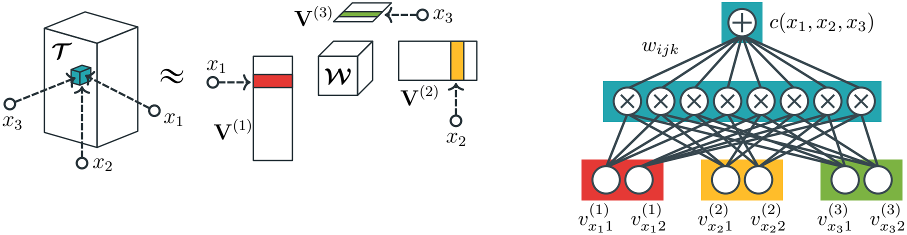
but also OB/S/DDs, as circuits





tensor factorizations

as circuits



Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", arXiv, 2024

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic and logical models

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

O/BDDs, *SDDs* and other *PGMs*...

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic and logical models

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

O/BDDs, *SDDs* and other *PGMs*...

2. Expressiveness

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

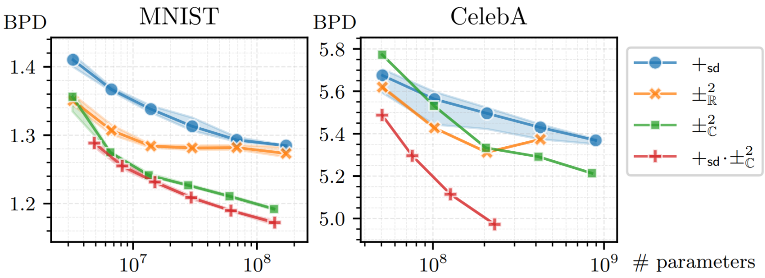
How expressive?

	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", NeurIPS, 2024

How scalable?



up to billions of parameters

Loconte, Mengel, and Vergari, "Sum of Squares Circuits", arXiv, 2024

How to build
& learn
probabilistic circuits?

cirkit

learning & reasoning with circuits in pytorch

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

```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                    # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

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

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

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic and logical models

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

O/BDDs, *SDDs* and other *PGMs*...

2. Expressiveness

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

...why PCs?

1. A grammar for tractable models

One formalism to represent many probabilistic and logical models

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

O/BDDs, SDDs 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***

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

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

$\Longrightarrow$ *tractable partition function*

$\Longrightarrow$ *also any* ***conditional*** *is tractable*

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

Structural properties

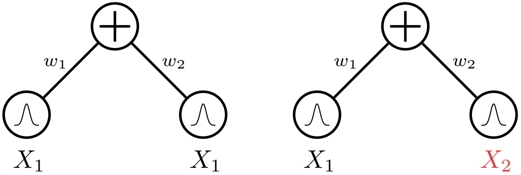
smoothness

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

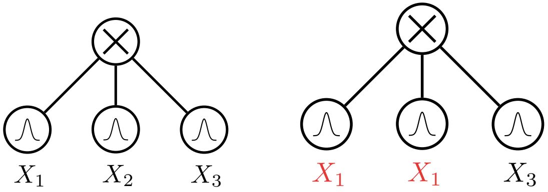
smoothness

decomposability

compatibility

determinism

the inputs of prod units are defined over disjoint variable sets

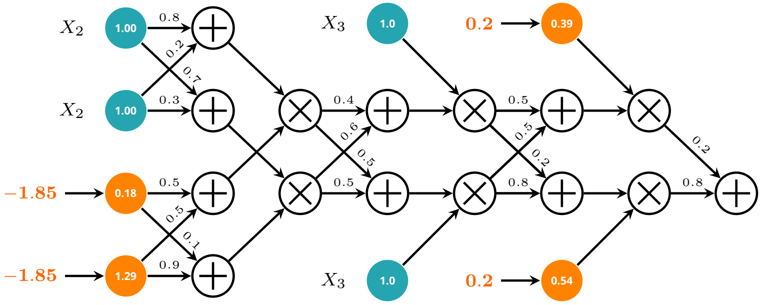


***decomposable* circuit** **non-decomposable circuit**

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

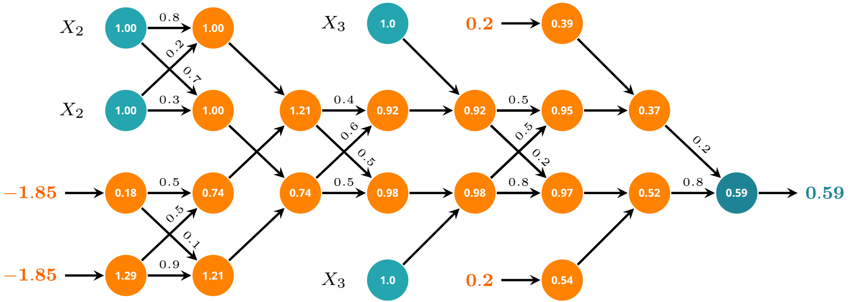
Probabilistic queries = feedforward evaluation

$$p(X_1 = -\mathbf{1.85}, X_4 = \mathbf{0.2})$$



Probabilistic queries = feedforward evaluation

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



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

Computing arbitrary integrations (or summations)

$\Longrightarrow$ *linear in circuit size!*

E.g., suppose we want to compute Z:

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

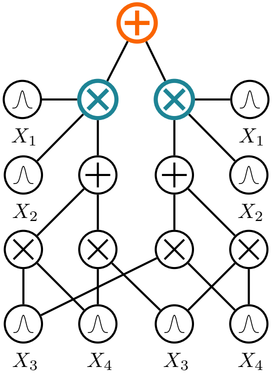
smooth + decomposable circuits = ...

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

$$\int \boldsymbol{p}(\mathbf{x}) d\mathbf{x} = \int \sum_i w_i \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x} =$$

$$= \sum_i w_i \int \boldsymbol{p}_i(\mathbf{x}) d\mathbf{x}$$

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



smooth + decomposable circuits = ...

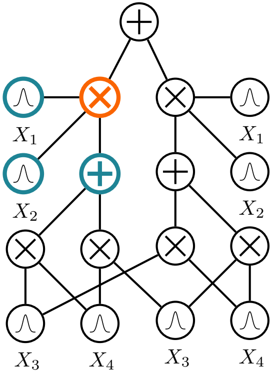
If $p(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})$, (***decomposability***):

$$\int\int\int p(\mathbf{x}, \mathbf{y}, \mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int\int\int p(\mathbf{x})p(\mathbf{y})p(\mathbf{z})d\mathbf{x}d\mathbf{y}d\mathbf{z} =$$

$$= \int p(\mathbf{x})d\mathbf{x} \int p(\mathbf{y})d\mathbf{y} \int p(\mathbf{z})d\mathbf{z}$$

$\Longrightarrow$ *integrals decompose into easier ones*

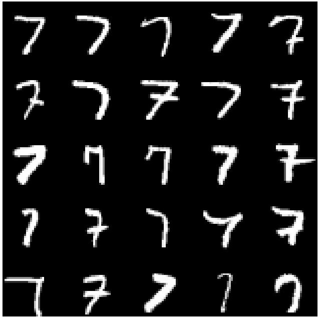


```python
from cirkit.backend.torch.queries import IntegrateQuery
marginal_query = IntegrateQuery(circuit)

with torch.no_grad():
    test_marginal_lls = 0.0

    for batch, _ in test_dataloader:
        batch = batch.to(device).unsqueeze(dim=1)
        marginal_log_likelihoods = marginal_query(batch, integrate_vars=vars_to_marginalize)
        test_marginal_lls += marginal_log_likelihoods.sum().item()

    marg_ll = test_marginal_lls / len(data_test)
    print(f"marg LL: {marg_ll:.3f}") # marg LL:: -378.417
```

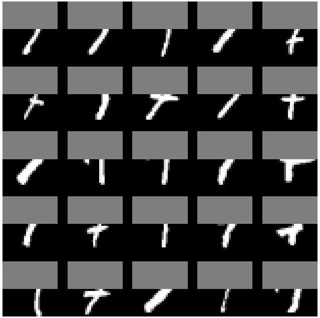
Tractable inference on PCs

Einsum networks

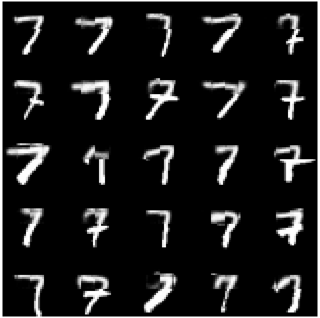
Original



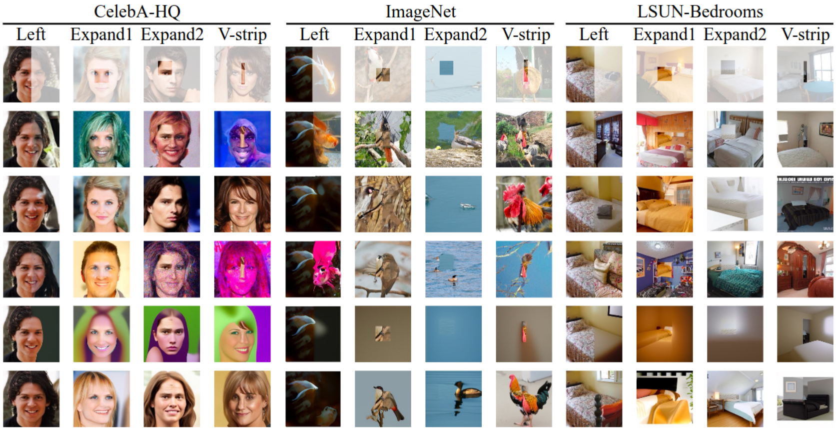
Missing



Conditional sample



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



Liu, Niepert, and Broeck, "Image Inpainting via Tractable Steering of Diffusion Models", ICLR, 2024

Which structural properties

for complex reasoning

smooth + decomposable

???????

???????

General expectations

Integrals involving two or more functions:

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

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

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? E.g.,

Structural properties

smoothness

decomposability

compatibility

determinism

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

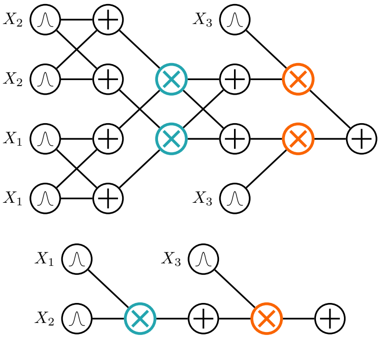
Structural properties

smoothness

decomposability

compatibility

determinism



compatible circuits

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

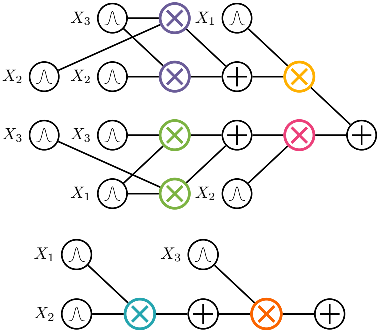
Structural properties

smoothness

decomposability

compatibility

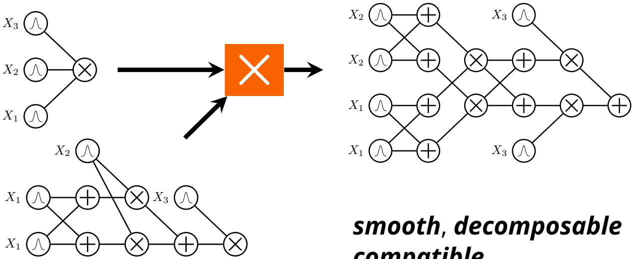
determinism



non-compatible circuits

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

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

Semantic Probabilistic Layers for Neuro-Symbolic Learning

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

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

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

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

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

***circuit products for reliable NeSy:* tomorrow**

```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                    # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

learning probabilistic circuits

learning probabilistic circuits

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

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...

which loss?

how to learn normalized weights?

how to exploit structural properties?

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

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

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

```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                    # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

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., $\sum_i w_i = 1$

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

```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                    # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

Probabilistic Circuits (PCs)

the unit-wise definition

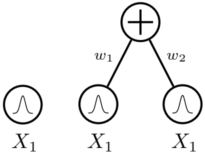
I. *A simple tractable function is a circuit*

Probabilistic Circuits (PCs)

the unit-wise definition

I. *A simple tractable function is a circuit*

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



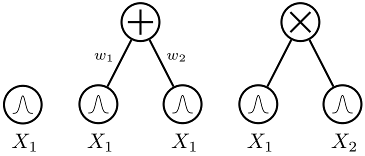
Probabilistic Circuits (PCs)

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*

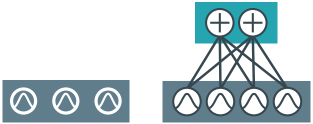
Probabilistic Circuits (PCs)

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)

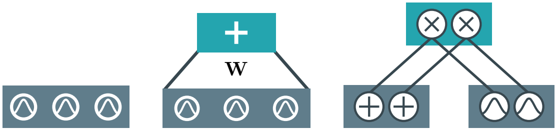
the layer-wise definition

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)

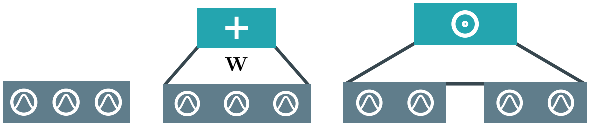
the layer-wise definition

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)

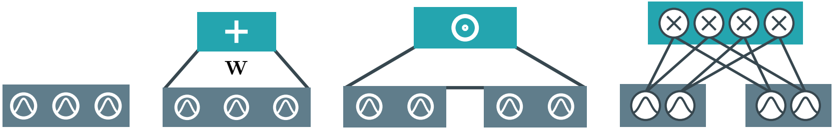
the layer-wise definition

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*

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



Probabilistic Circuits (PCs)

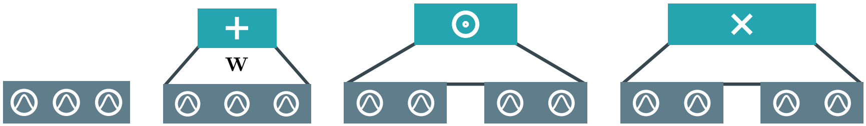
the layer-wise definition

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

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

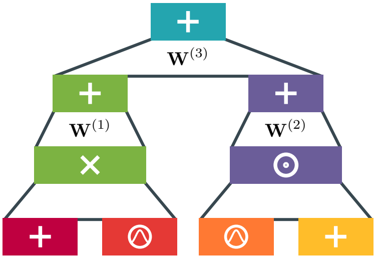
III. *The product of two layers is a circuit layer*

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



Probabilistic Circuits (PCs)

the layer-wise definition



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

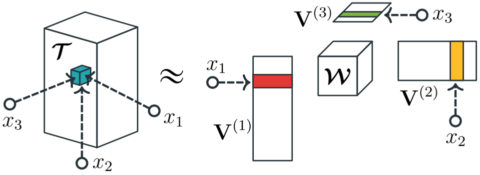
II. *A linear projection of a layer is a circuit layer*

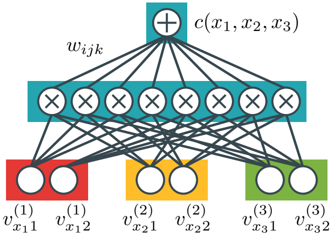
III. *The product of two layers is a circuit layer*

stack layers to build a deep circuit!

circuits layers

as tensor factorizations

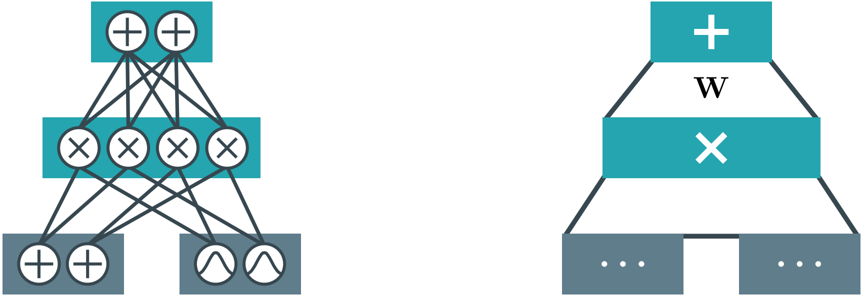




Loconte et al., "What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?", arXiv, 2024

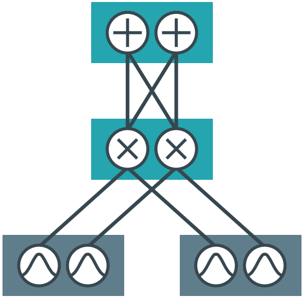
more layers

Tucker decomposition

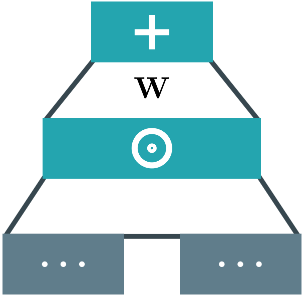


more layers

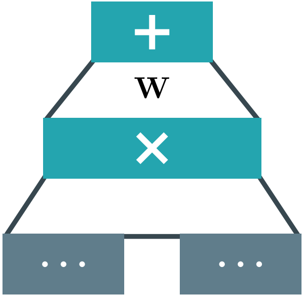
Candecomp Parafac (CP) decomposition



more layers



CP layer

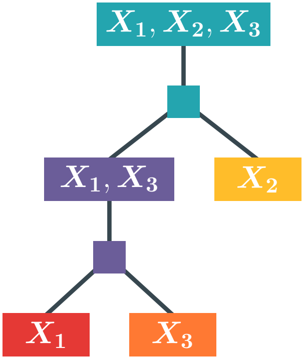


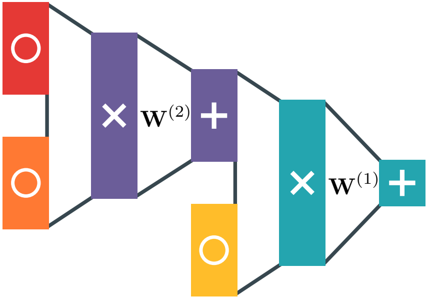
Tucker layer

```python
from cirkit.templates import circuit_templates

symbolic_circuit = circuit_templates.image_data(
(1, 28, 28),                    # The shape of MNIST
    region_graph='quad-graph',
    input_layer='categorical',  # input distributions
    sum_product_layer='cp',     # CP, Tucker, CP-T
    num_input_units=64,         # overparameterizing
    num_sum_units=64,
    sum_weight_param=circuit_templates.Parameterization(
        activation='softmax',
        initialization='normal'
    )
)
```

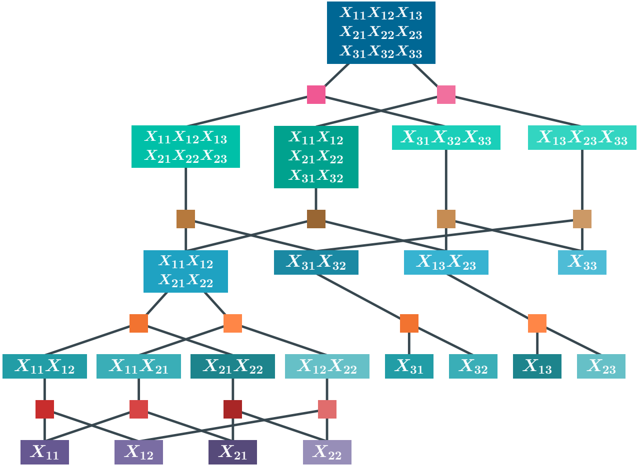
region graphs

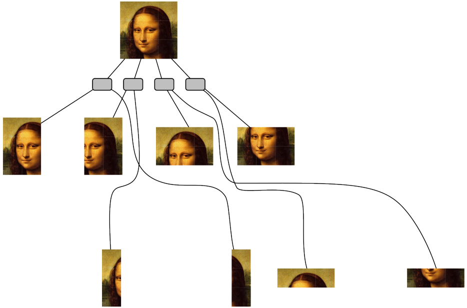
a template for smooth&decomposable PCs

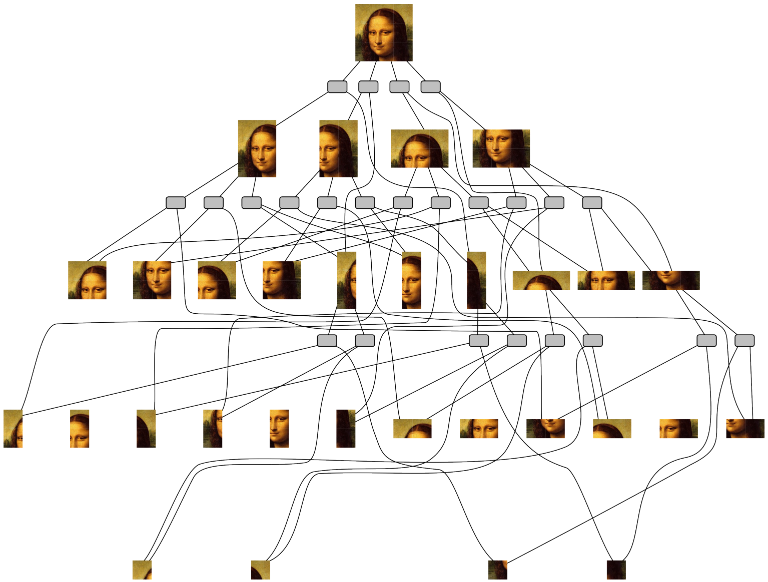


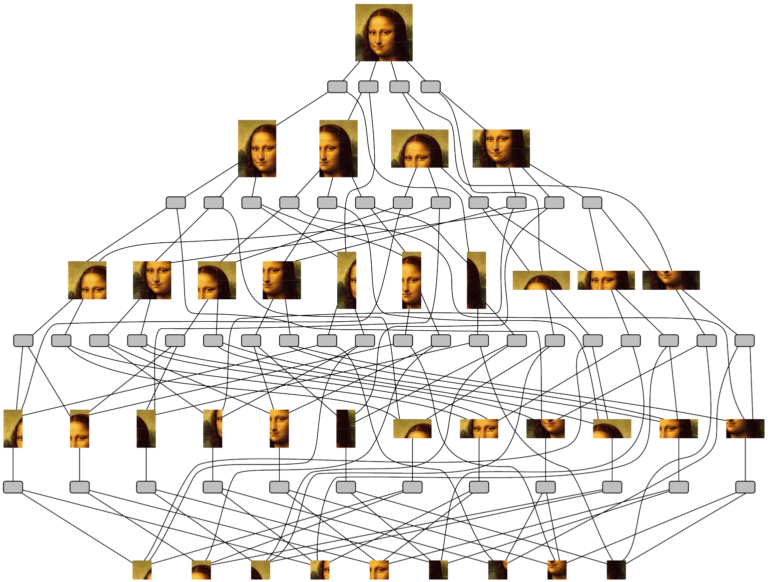


which region graph?





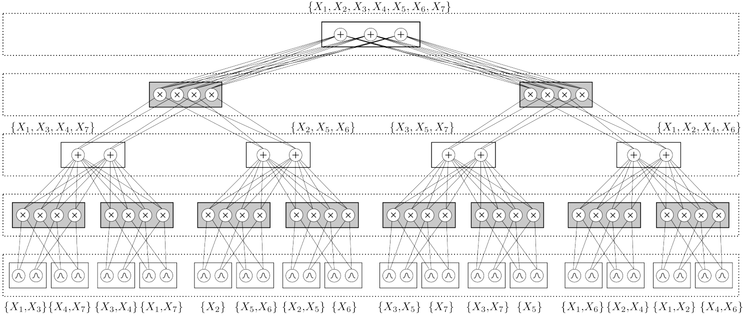




random regions graphs

The "no-learning" option

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



main · cirkit / notebooks / region-graphs-and-parametrisation.ipynb · Go to file

loreloc updated notebooks with respect to API changes · e3e7e80 · 2 days ago

Preview · Code · Blame · 1082 lines (1082 loc) · 793 KB · Raw

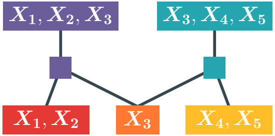
Notebook on Region Graphs and Sum Product Layers

Goals

By the end of this tutorial you will:

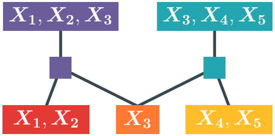
- know what a region graph is
- know how to choose between region graphs for your circuit
- understand how to parametrize a circuit by choosing a sum product layer
- build circuits to **tractably** estimate a probability distribution over images[1]

learning recipe

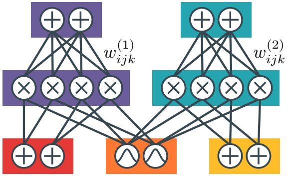


1) Build a ***region graph***

learning recipe



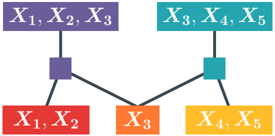
1) Build a ***region graph***



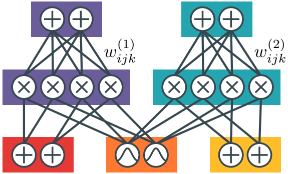
2) Overparameterize

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

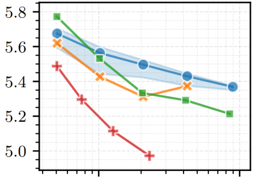
learning recipe



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



2) Overparameterize



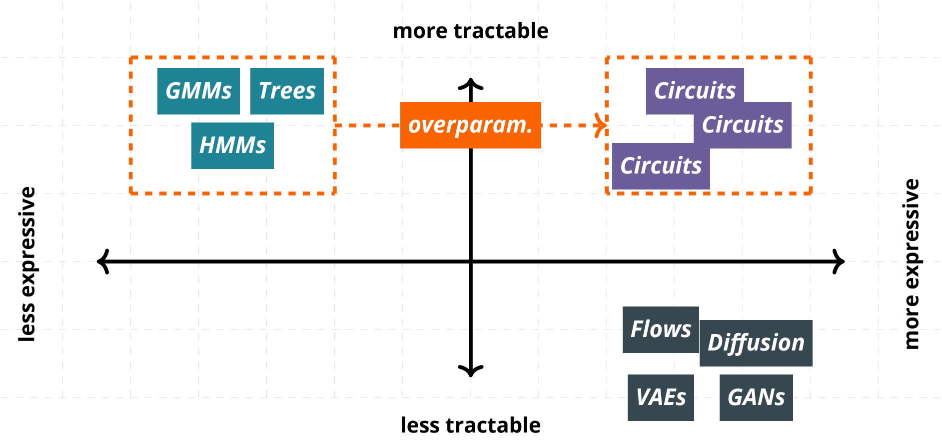
3) Learn parameters

use any optimizer in pytorch

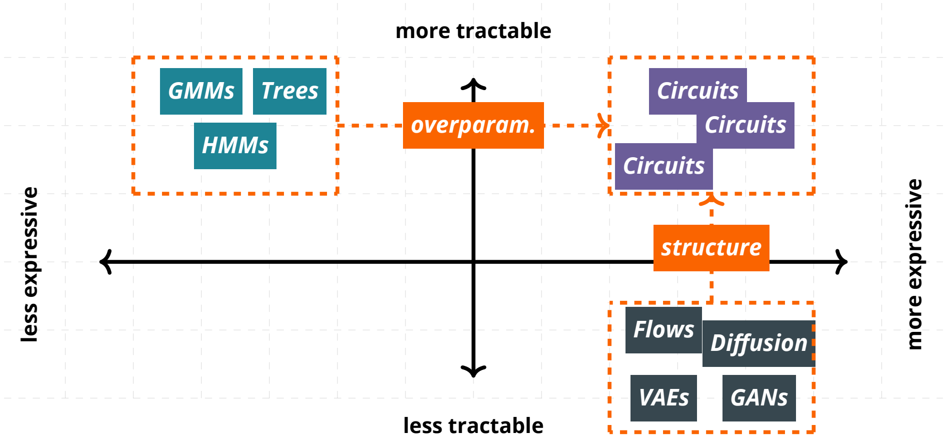
cirkit

learning & reasoning with circuits in pytorch

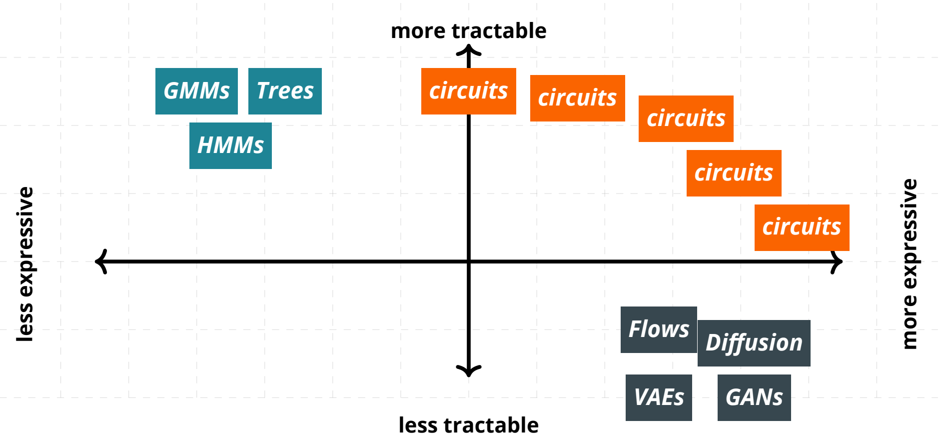
`https://github.com/april-tools/cirkit`



make it more expressive!



impose structure!



navigate the spectrum!