Subtractive mixture models

representation and learning

antonio vergari **(he/him)**

@tetraduzione

joint work with Lorenzo Loconte, Aleksanteri M. Sladek, Stefan Mangel, Martin Trapp, Arno Solin, Nicolas Gillis

19th Oct 2023 - **Simons Institute**

april

april is probably a recursive, identifier of a lab

april

about
probabilities
reasoning,
integrals &
logic

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

TCS crowd

a circuit lowerbound to play with

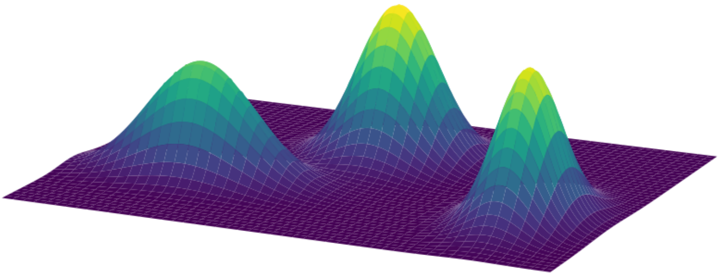
connections with mixtures/PGMs/learning

ML crowd

(some) new tractable model(s) to play with

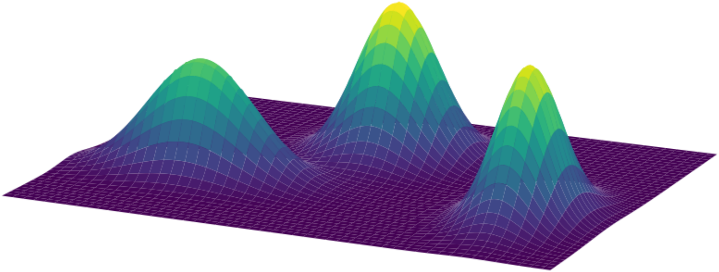
a tensorized way to represent circuits

why subtractions in mixture models



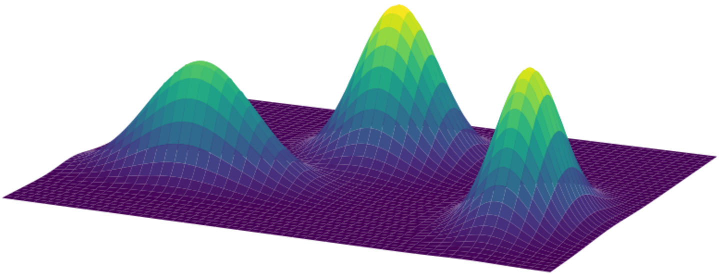
mixtures are a staple in probML

image taken from Hao Tang's course on ASR



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

image taken from Hao Tang's course on ASR

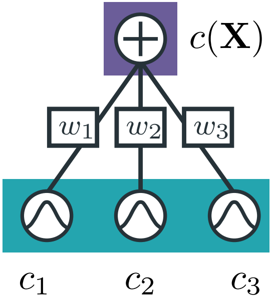


$$c(\mathbf{X}) = \sum_{i=1}^{K} w_i c_i(\mathbf{X}), \quad \text{with} \quad w_i \geq 0, \quad \sum_{i=1}^{K} w_i = 1$$

image taken from Hao Tang's course on ASR

additive MMs

are so cool!



easily represented as shallow **probabilistic circuits** (PCs)

$\Rightarrow$ *smooth, (structured) decomposable*

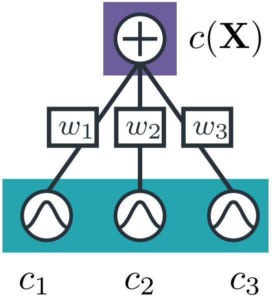
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



easily represented as shallow **probabilistic circuits** (PCs)

$\Longrightarrow$ *smooth, (structured) decomposable*

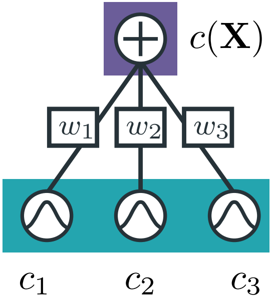
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



easily represented as shallow **probabilistic circuits** (PCs)

$\Rightarrow$ *smooth, (structured) decomposable*

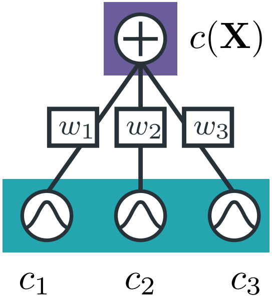
these are **monotonic** PCs

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

universal approximators...

additive MMs

are so cool!



easily represented as shallow **probabilistic circuits** (PCs)

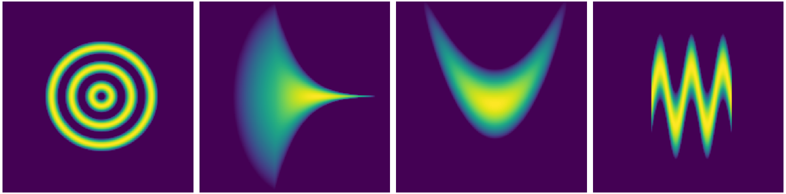
$\Longrightarrow$ *smooth, (structured) decomposable*

these are **monotonic** PCs

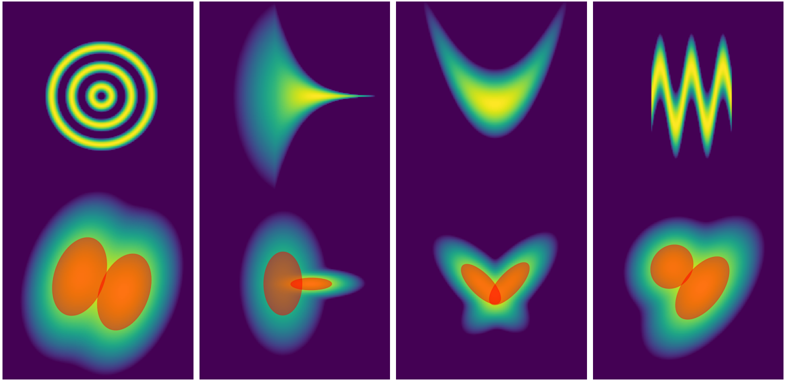
if marginals/conditionals are tractable for the components, they are tractable for the MM

universal approximators...

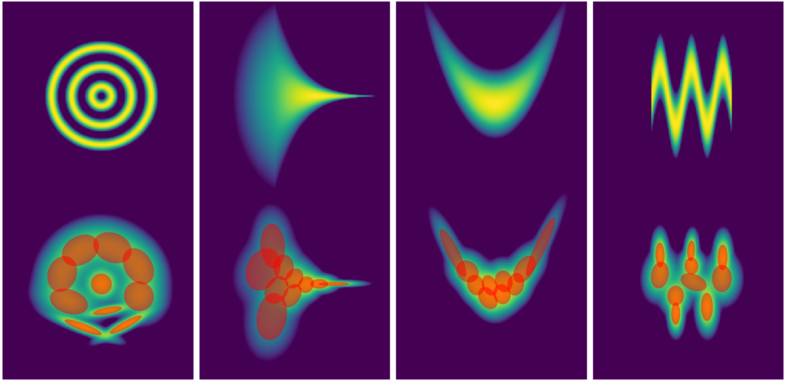
additive MMs



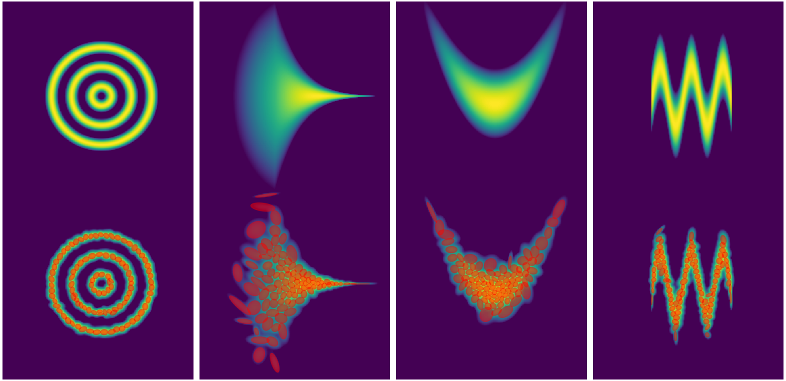
additive MMs



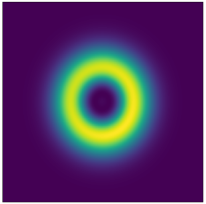
additive MMs



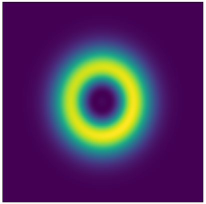
additive MMs

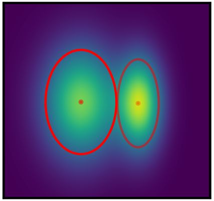


however...



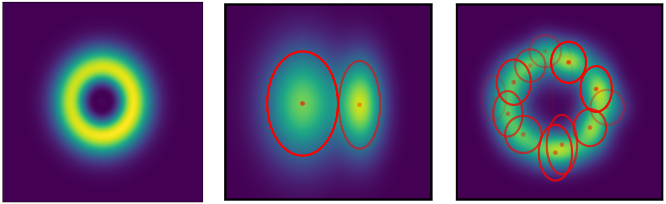
however...





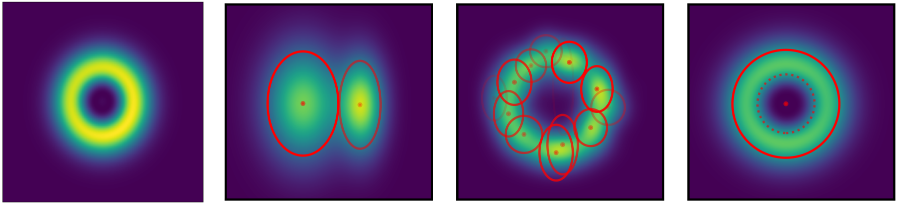
GMM ($K = 2$)

however...



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

however...

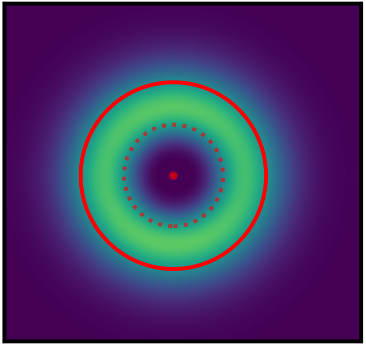


GMM ($K = 2$) GMM ($K = 16$) nGMM^2 ($K = 2$)

SPOILER ALERT

Shallow mixtures with negative parameters can be *exponentially more compact* than deep ones with positive ones.

subtractive MMs



sometimes called **negative** MMs

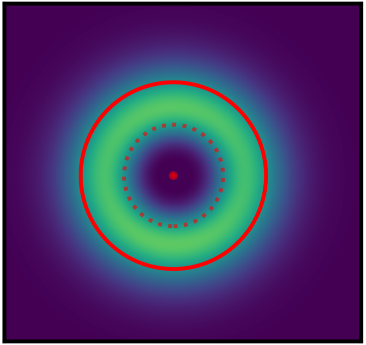
$\Rightarrow$ *or **non-monotonic** circuits,...*

issue: how to preserve non-negative outputs?

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

$\Rightarrow$ *constraints on variance, mean*

subtractive MMs



sometimes called **negative** MMs

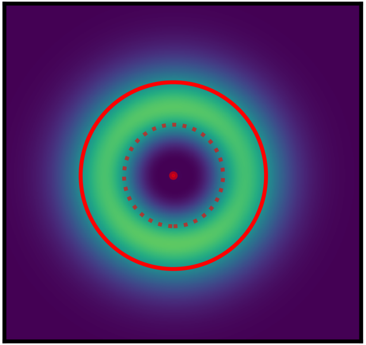
$\Rightarrow$ *or **non-monotonic** circuits,...*

issue: how to preserve non-negative outputs?

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

$\Rightarrow$ *constraints on variance, mean*

subtractive MMs



sometimes called **negative** MMs

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

issue: how to preserve non-negative outputs?

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

$\Rightarrow$ *constraints on variance, mean*

tl;dr

"Understand when and how
we can use negative parameters
in deep subtractive mixture models"

tl;dr

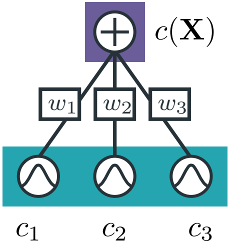
***"Understand when and how
we can use negative parameters
in deep non-monotonic squared circuits"***

tl;dr

"Understand when and how
we can use negative parameters
in deep non-monotonic squared circuits"

$\Rightarrow$ ***later PSD kernel models, tensor networks, ...***

subtractive MMs as circuits

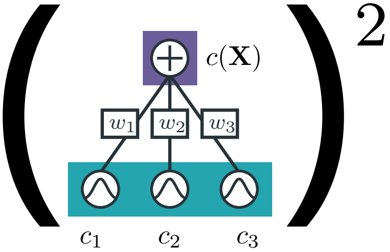


a **non-monotonic** smooth and (structured) decomposable circuit

$\Rightarrow$ *possibly with negative outputs*

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

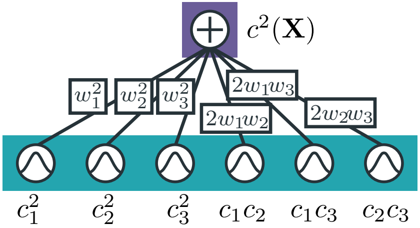
squaring shallow MMs



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

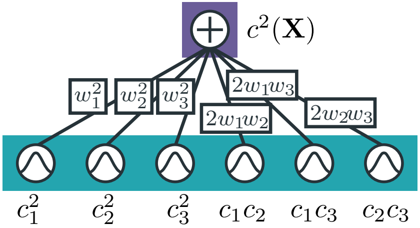
$\Longrightarrow$ *ensure non-negative output*

squaring shallow MMs



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

squaring shallow MMs

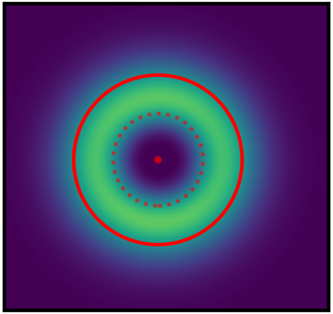


$$
\begin{aligned}
c^2(\mathbf{X}) &= \left(\sum_{i=1}^{K} w_i c_i(\mathbf{X})\right)^2 \\
&= \sum_{i=1}^{K} \sum_{j=1}^{K} w_i w_j c_i(\mathbf{X}) c_j(\mathbf{X})
\end{aligned}
$$

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

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

squaring shallow MMs



e.g., a squared GMM with negative parameters w_i

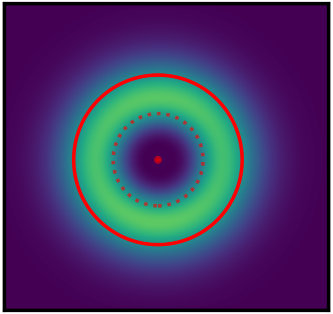
but we do not require non-negative inputs!

$\Longrightarrow$ *e.g. use* ***splines***

to renormalize we need to compute

$$\int c^2(\mathbf{x})\,\mathrm{d}\mathbf{x} = \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

squaring shallow MMs



e.g., a squared GMM with negative parameters w_i

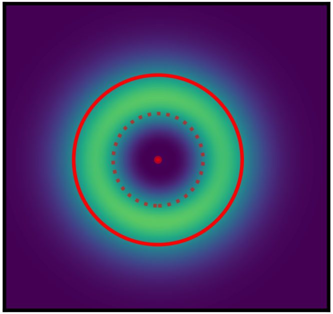
but we do not require non-negative inputs!

$\Longrightarrow$ *e.g. use* ***splines***

to renormalize we need to compute

$$\int c^2(\mathbf{x})\,\mathrm{d}\mathbf{x} = \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

squaring shallow MMs



e.g., a squared GMM with negative parameters w_i

but we do not require non-negative inputs!

$\Longrightarrow$ *e.g. use **splines***

to renormalize we need to compute

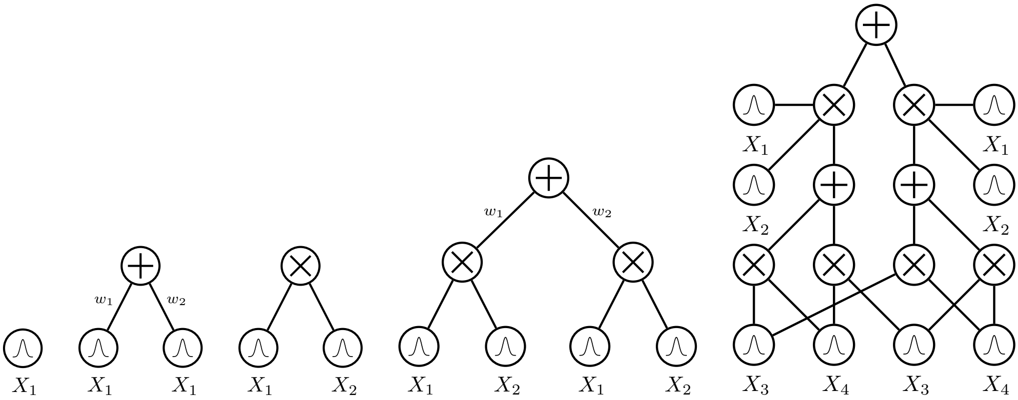
$$\int c^2(\mathbf{x})\,\mathrm{d}\mathbf{x} = \sum_{i=1}^{K}\sum_{j=1}^{K} w_i w_j \int c_i(\mathbf{x}) c_j(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

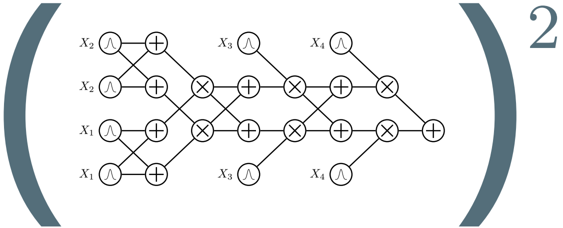
why subtractions in mixture models

how to represent them as deep squared circuits?

Circuits

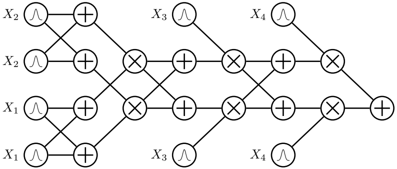
A grammar for tractable computational graphs





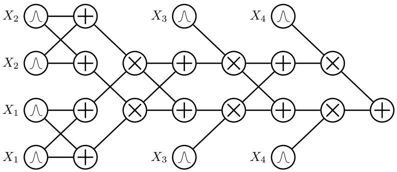
how to efficiently square (and renormalize) a deep PC?

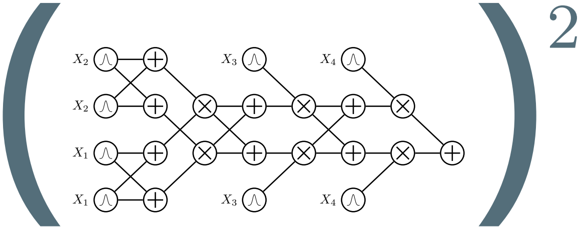
Tractable square



smooth
structured decomposable

Tractable square

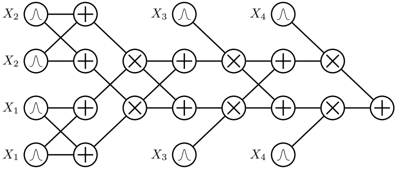


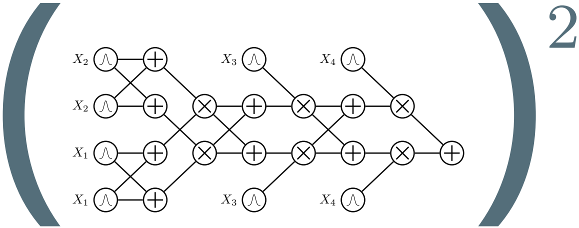


smooth
structured decomposable

smooth
structured decomposable

Tractable square





exactly compute $\int \boldsymbol{c}(\mathrm{x})\boldsymbol{c}(\mathrm{x})d\mathrm{X}$ **in time** $O(|\boldsymbol{c}|^2)$

Tractable square

Algorithm 3 MULTIPLY(p, q, cache)

Input: two circuits $p(\mathbf{Z})$ and $q(\mathbf{Y})$ that are compatible over $\mathbf{X} = \mathbf{Z} \cap \mathbf{Y}$ and a cache for memoization
Output: their product circuit $m(\mathbf{Z} \cup \mathbf{Y}) = p(\mathbf{Z})q(\mathbf{Y})$
if $(p, q) \in \mathsf{cache}$ **then return** $\mathsf{cache}(p, q)$
if $\phi(p) \cap \phi(q) = \emptyset$ **then**
  $m \leftarrow \textsc{Product}(\{p, q\});\ s \leftarrow \text{True}$
else if p, q are input units **then**
  $m \leftarrow \textsc{Input}(p(\mathbf{Z}) \cdot q(\mathbf{Y}), \mathbf{Z} \cup \mathbf{Y})$
  $s \leftarrow [\![\mathsf{supp}(p(\mathbf{X})) \cap \mathsf{supp}(q(\mathbf{X})) \neq \emptyset]\!]$
else if p is an input unit **then**
  $n \leftarrow \{\}; s \leftarrow \text{False}\ // q(\mathbf{Y}) = \sum_j \theta'_j q_j(\mathbf{Y})$
  for $j = 1$ **to** $|\mathsf{in}(q)|$ **do**
    $n', s' \leftarrow \text{MULTIPLY}(p, q_j, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, \{\theta'_j\}_{j=1}^{|\mathsf{in}(q)|})$ **else** $m \leftarrow \mathit{null}$
else if q is an input unit **then**
  $n \leftarrow \{\}; s \leftarrow \text{False}\ // p(\mathbf{Z}) = \sum_i \theta_i p_i(\mathbf{Z})$
  for $i = 1$ **to** $|\mathsf{in}(p)|$ **do**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, \{\theta_i\}_{i=1}^{|\mathsf{in}(p)|})$ **else** $m \leftarrow \mathit{null}$
else if p, q are product units **then**
  $n \leftarrow \{\}; s \leftarrow \text{True}$
  $\{p_i, q_i\}_{i=1}^{k} \leftarrow \mathsf{sortPairsByScope}(p, q, \mathbf{X})$
  for $i = 1$ **to** k **do**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q_i, \mathsf{cache})$
    $n \leftarrow n \cup \{n'\};\ s \leftarrow s \wedge s'$
  if s **then** $m \leftarrow \textsc{Product}(n)$ **else** $m \leftarrow \mathit{null}$
else if p, q are sum units **then**
  $n \leftarrow \{\};\ w \leftarrow \{\};\ s \leftarrow \text{False}$
  for $i = 1$ **to** $|\mathsf{in}(p)|, j = 1$ **to** $|\mathsf{in}(q)|$ **do**
    $n', s' \leftarrow \text{MULTIPLY}(p_i, q_j, \mathsf{cache})$
    $n \leftarrow n \cup n'; w \leftarrow w \cup \{\theta_i \theta'_j\}; s \leftarrow s \vee s'$
  if s **then** $m \leftarrow \textsc{Sum}(n, w)$ **else** $m \leftarrow \mathit{null}$
$\mathsf{cache}(p, q) \leftarrow (m, s)$
return m, s

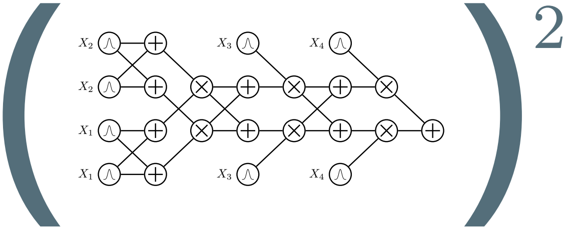
on tensorized PCs

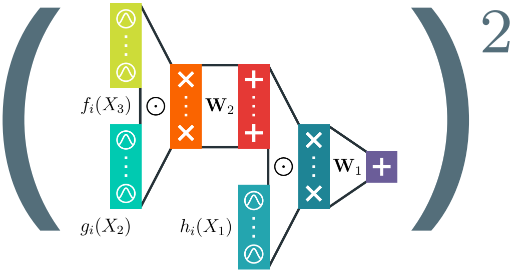
Algorithm 1 $\mathsf{squareTensorizedCircuit}(\boldsymbol{\ell}, \mathcal{R})$

Input: A tensorized circuit having output layer $\boldsymbol{\ell}$ and defined on a tree RG rooted by $\mathcal{R}$.
Output: The tensorized squared circuit defined on the same tree RG having $\boldsymbol{\ell}^2$ as output layer computing $\boldsymbol{\ell} \otimes \boldsymbol{\ell}$.

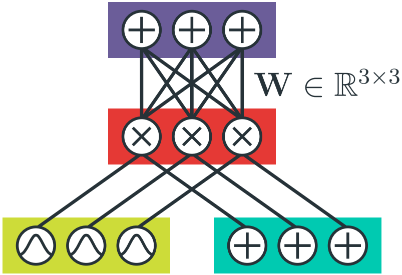
if $\boldsymbol{\ell}$ is an input layer **then**
  $\boldsymbol{\ell}$ computes K functions $f_i(\mathcal{R})$
  return An input layer $\boldsymbol{\ell}^2$ computing all K^2
    product combinations $f_i(\mathcal{R}) f_j(\mathcal{R})$
else if $\boldsymbol{\ell}$ is a product layer **then**
  $\{(\boldsymbol{\ell}_{\mathsf{i}}, \mathcal{R}_{\mathsf{i}}), (\boldsymbol{\ell}_{\mathsf{ii}}, \mathcal{R}_{\mathsf{ii}})\} \leftarrow \mathsf{getInputs}(\boldsymbol{\ell}, \mathcal{R})$
  $\boldsymbol{\ell}_{\mathsf{i}}^2 \leftarrow \mathsf{squareTensorizedCircuit}(\boldsymbol{\ell}_{\mathsf{i}}, \mathcal{R}_{\mathsf{i}})$
  $\boldsymbol{\ell}_{\mathsf{ii}}^2 \leftarrow \mathsf{squareTensorizedCircuit}(\boldsymbol{\ell}_{\mathsf{ii}}, \mathcal{R}_{\mathsf{ii}})$
  return $\boldsymbol{\ell}_{\mathsf{i}}^2 \odot \boldsymbol{\ell}_{\mathsf{ii}}^2$
else ▷ $\boldsymbol{\ell}$ is a sum layer
  $\{(\boldsymbol{\ell}_{\mathsf{i}}, \mathcal{R})\} \leftarrow \mathsf{getInputs}(\boldsymbol{\ell}, \mathcal{R})$
  $\boldsymbol{\ell}_{\mathsf{i}}^2 \leftarrow \mathsf{squareTensorizedCircuit}(\boldsymbol{\ell}_{\mathsf{i}}, \mathcal{R})$
  $\mathbf{W} \in \mathbb{R}^{S \times K} \leftarrow \mathsf{getParameters}(\boldsymbol{\ell})$
  $\mathbf{W}' \in \mathbb{R}^{S^2 \times K^2} \leftarrow \mathbf{W} \otimes \mathbf{W}$
  return $\mathbf{W}' \boldsymbol{\ell}_{\mathsf{i}}^2$

Loconte et al., "Subtractive Mixture Models via Squaring: Representation and Learning", arXiv preprint arXiv:2310.00724, 2023


X_2
X_2
X_1
X_1
X_3
X_3
X_4
X_4
2


$f_i(X_3)$
$\odot$
$\mathbf{W}_2$
$g_i(X_2)$
$h_i(X_1)$
$\odot$
$\mathbf{W}_1$
2

Tensorizing str-dec PCs



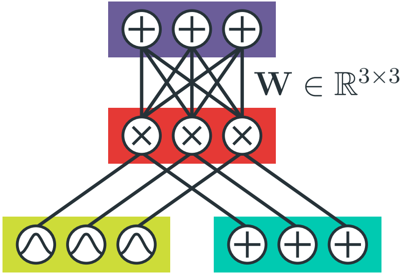
abstract computations into ***layers***

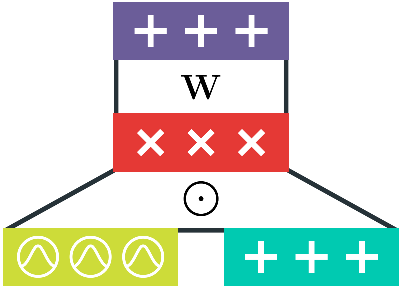
group units with the same scope

parameterize connections by matrix/vector operations

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

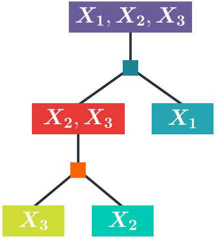
Tensorizing str-dec PCs



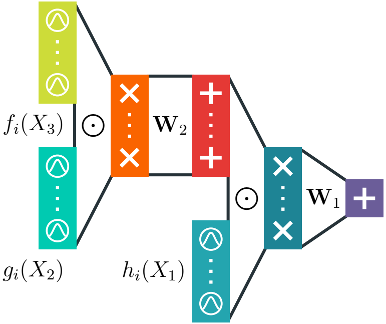


Mari, Vessio, and Vergari, "Unifying and Understanding Overparameterized Circuit Representations via Low-Rank Tensor Decompositions", , 2023

Tensorizing str-dec PCs



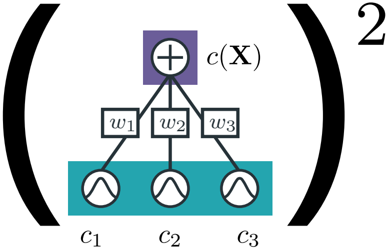
region graph / vtree / pseudotree



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

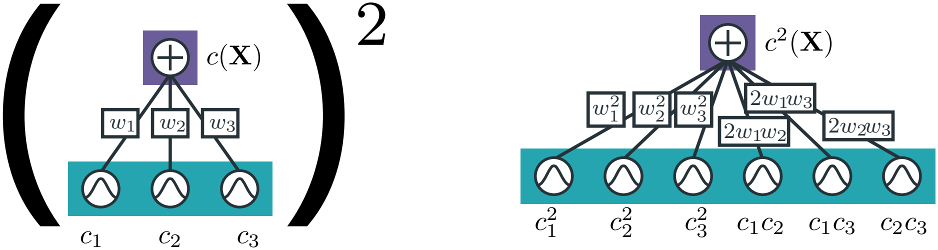
squaring deep PCs

the tensorized way



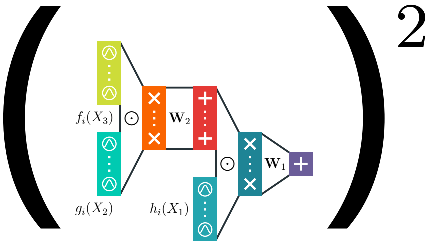
squaring deep PCs

the tensorized way



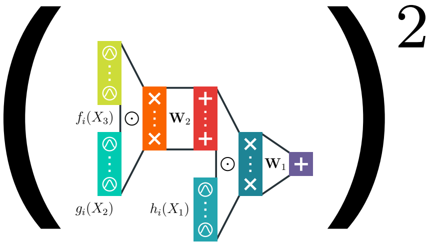
squaring deep PCs

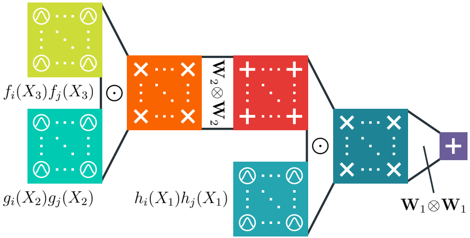
the tensorized way



squaring deep PCs

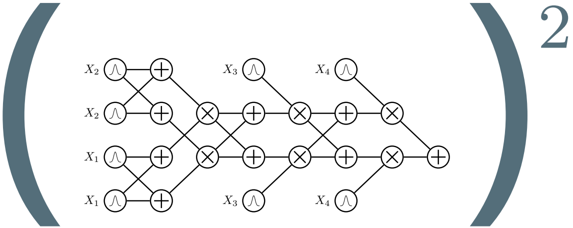
the tensorized way





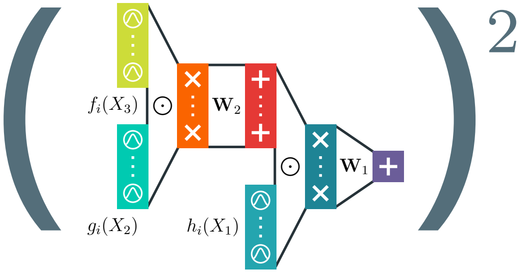
squaring reduces to square layers

Tractable squares



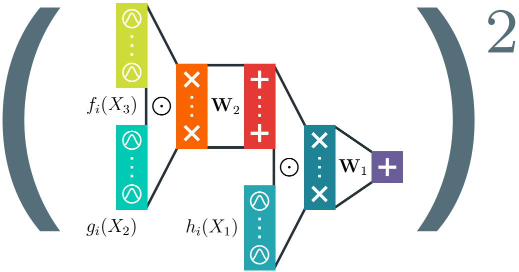
exactly compute $\int \boldsymbol{c}(\mathbf{x})\boldsymbol{c}(\mathbf{x})d\mathbf{X}$ **in time** $O(|\boldsymbol{c}|^2)$

Tractable squares



exactly compute $\int \boldsymbol{c}(\mathrm{x})\boldsymbol{c}(\mathrm{x})d\mathrm{X}$ **in time** $O((\boldsymbol{L}\boldsymbol{K})^2)$

Tractable squares



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

why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

Fully factorized

NaiveBayes *AndOrGraphs* *PDGs*

Trees *PSDDs* *CNets* *LTMs* *SPNs* *NADEs*

Thin Junction Trees *ACs* *MADEs* *MAFs* *VAEs*

DPPs *FVSBNs* *TACs* *IAFs* *NAFs* *RAEs*

Mixtures *BNs* *NICE* *FGs* *GANs*

RealNVP *MNs*

the alphabet soup of *tractable* models

Fully factorized

PSDs NaiveBayes AndOrGraphs PDGs MDPs

Trees PSDDs CNets LTMs SPNs NADEs

Thin Junction Trees ACs MADEs MAFs VAEs

DPPs FVSBNs TACs IAFs NAFs RAEs

Mixtures BNs NICE FGs GANs

RealNVP MNs

new entries in the family!

PSD kernels

Given a ***kernel*** κ and a set of d data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(d)}$ with $\boldsymbol{\kappa}(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}^{(1)}), \ldots, \kappa(\mathbf{x}, \mathbf{x}^{(d)})]^\top \in \mathbb{R}^d$, define the non-negative function

$$f(\mathbf{x}; \mathbf{A}, \boldsymbol{\kappa}) = \boldsymbol{\kappa}(\mathbf{x})^\top \mathbf{A} \boldsymbol{\kappa}(\mathbf{x})$$

where $\mathbf{A}$ is a real $d \times d$ ***positive semi-definite*** matrix.

Rudi and Ciliberto, "PSD Representations for Effective Probability Models", , 2021

PSD kernels

Given a kernel κ and a set of d data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(d)}$ with $\boldsymbol{\kappa}(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}^{(1)}), \ldots, \kappa(\mathbf{x}, \mathbf{x}^{(d)})]^\top \in \mathbb{R}^d$, define the non-negative function

$$f(\mathbf{x}; \mathbf{A}, \boldsymbol{\kappa}) = \boldsymbol{\kappa}(\mathbf{x})^\top \mathbf{A} \boldsymbol{\kappa}(\mathbf{x})$$

where $\mathbf{A}$ is a real $d \times d$ positive semi-definite matrix. Just a ***mixture of squared PCs***

$$f(\mathbf{x}; \mathbf{A}, \boldsymbol{\kappa}) = \boldsymbol{\kappa}(\mathbf{x})^\top \left(\sum_{i=1}^{r} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top\right) \boldsymbol{\kappa}(\mathbf{x}) = \sum_{i=1}^{r} \lambda_i \left(\mathbf{u}_i^\top \boldsymbol{\kappa}(\mathbf{x})\right)^2,$$

Rudi and Ciliberto, "PSD Representations for Effective Probability Models", , 2021

tensor networks

matrix-product state, tensor trains, Born machines,...

A ***maxtrix-product state*** or ***tensor-train*** factorizes a D-dimensional tensor $\mathcal{T}$ as

$$\mathcal{T}[x_1, \ldots, x_D] = \sum_{i_1=1}^{r} \sum_{i_2=1}^{r} \cdots \sum_{i_{D-1}=1}^{r} \mathbf{A}_1[x_1, i_1] \mathbf{A}_2[x_2, i_1, i_2] \cdots \mathbf{A}_D[x_D, i_{D-1}]$$

and a ***Born machine*** squares it

$$\mathcal{B}[x_1, \ldots, x_D] = \left(\sum_{i_1=1}^{r} \sum_{i_2=1}^{r} \cdots \sum_{i_{D-1}=1}^{r} \mathbf{A}_1[x_1, i_1] \mathbf{A}_2[x_2, i_1, i_2] \cdots \mathbf{A}_D[x_D, i_{D-1}] \right)^2$$

tensor networks

matrix-product state, tensor trains, Born machines,...

A maxtrix-product state or tensor train factorizes a D-dimensional tensor $\mathcal{T}$ as

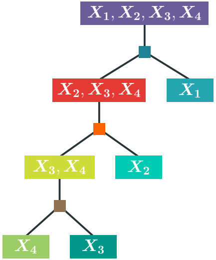
$$\mathcal{T}[x_1, \ldots, x_D] = \sum_{i_1=1}^{r} \sum_{i_2=1}^{r} \cdots \sum_{i_{D-1}=1}^{r} \mathbf{A}_1[x_1, i_1]\mathbf{A}_2[x_2, i_1, i_2] \cdots \mathbf{A}_D[x_D, i_{D-1}]$$

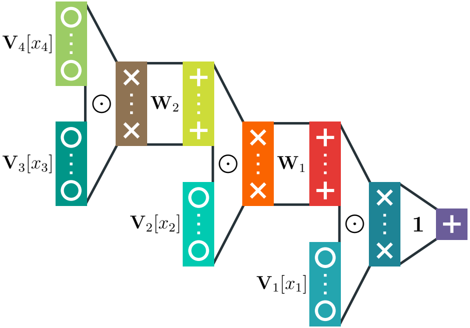
and a Born machine squares it

$$\mathcal{B}[x_1, \ldots, x_D] = \left(\sum_{i_1=1}^{r} \sum_{i_2=1}^{r} \cdots \sum_{i_{D-1}=1}^{r} \mathbf{A}_1[x_1, i_1]\mathbf{A}_2[x_2, i_1, i_2] \cdots \mathbf{A}_D[x_D, i_{D-1}] \right)^2$$

tensor networks

matrix-product state, tensor trains, Born machines,...

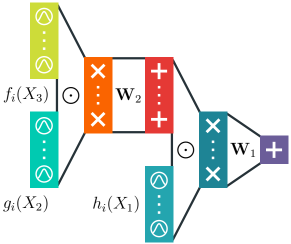




Marginals

and conditionals

monotonic

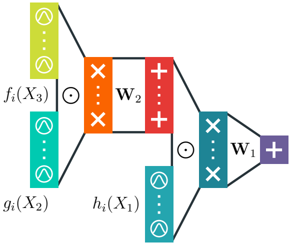


$$\mathcal{O}(|c|)$$

Marginals

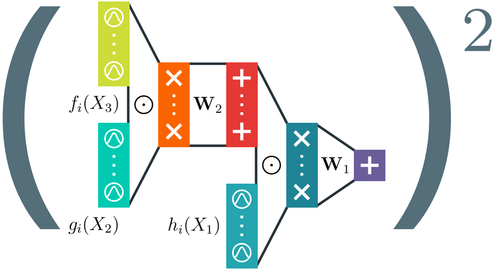
and conditionals

monotonic



$$\mathcal{O}(|c|)$$

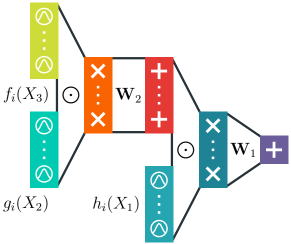
non-monotonic



$$\mathcal{O}(|c|^2)$$

Sampling

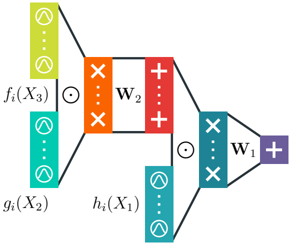
monotonic



$$\mathcal{O}(|c|)$$

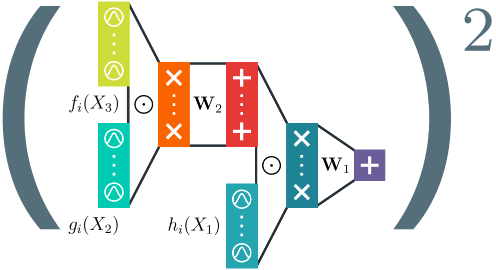
Sampling

monotonic



$$\mathcal{O}(|c|)$$

non-monotonic

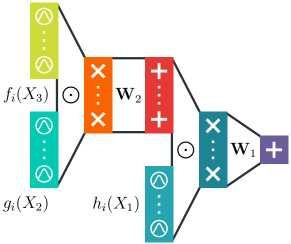


$$\mathcal{O}(|\mathbf{X}||c|^2)$$

by autoregressive sampling

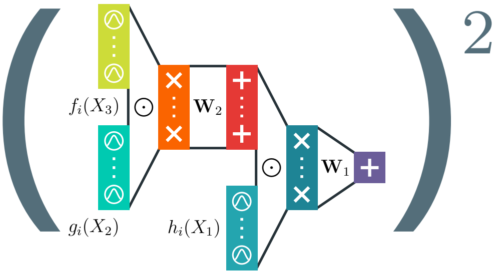
Sampling

monotonic



$$\mathcal{O}(|c|)$$

non-monotonic

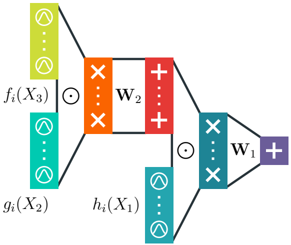


$$\mathcal{O}(\log|\mathbf{X}||c|^2)$$

for balanced vtrees

MAP

monotonic

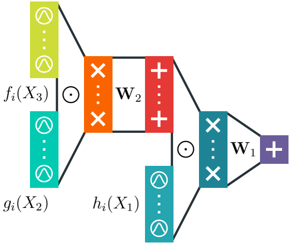


$$\mathcal{O}(|c|)$$

if deterministic

MAP

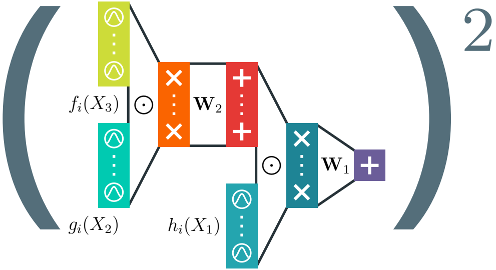
monotonic



$$\mathcal{O}(|c|)$$

if deterministic

non-monotonic



$$\mathcal{O}(|c|)$$

if deterministic

why subtractions in mixture models

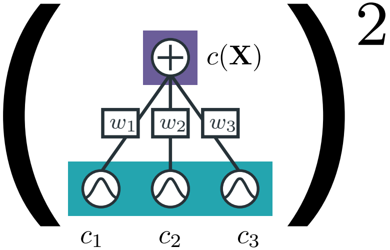
how to represent them as deep squared circuits?

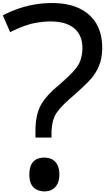
what inference and model classes they support?

when are they more expressive

more expressive?

squared probabilistic (o)BDDs, SDDs, str-d-DNNF?

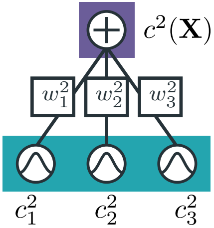




smooth, *deterministic*
structured decomposable

more expressive?

squared probabilistic (o)BDDs, SDDs, str-d-DNNF?



smooth, *deterministic*
structured decomposable

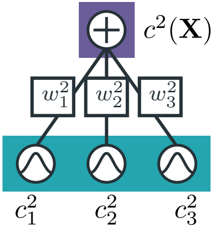
no increase in size

no increase in expressiveness

no negative weights

more expressive?

squared probabilistic (o)BDDs, SDDs, str-d-DNNF?



smooth, *deterministic*
structured decomposable

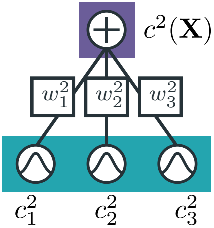
no increase in size

no increase in expressiveness

no negative weights

more expressive?

squared probabilistic (o)BDDs, SDDs, str-d-DNNF?



smooth, *deterministic*
structured decomposable

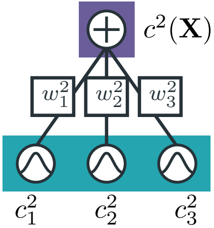
no increase in size

no increase in expressiveness

no negative weights

more expressive?

squared probabilistic (o)BDDs, SDDs, str-d-DNNF?



smooth, *deterministic*
structured decomposable

no increase in size

no increase in expressiveness

no negative weights

more expressive?

exponential separation

Theorem: there is a class of non-negative functions $\mathcal{F}$ over variables $\mathbf{X}$ that can be represented by ***compact a squared non-monotonic str-dec PC*** but for which the smallest monotonic str-dec PC computing $F \in \mathcal{F}$ has size $2^{\Omega(|\mathbf{X}|)}$***smallest monotonic str-dec PC computing*** $F \in \mathcal{F}$ ***has exponential size*** $2^{\Omega(|\mathbf{X}|)}$.

more expressive?

exponential separation

Theorem: there is a class of non-negative functions $\mathcal{F}$ over variables $\mathbf{X}$ that can be represented by compact a squared non-monotonic str-dec PC but for which the ***smallest monotonic str-dec PC computing*** $F \in \mathcal{F}$ ***has exponential size*** $2^{\Omega(|\mathbf{X}|)}$.

more expressive?

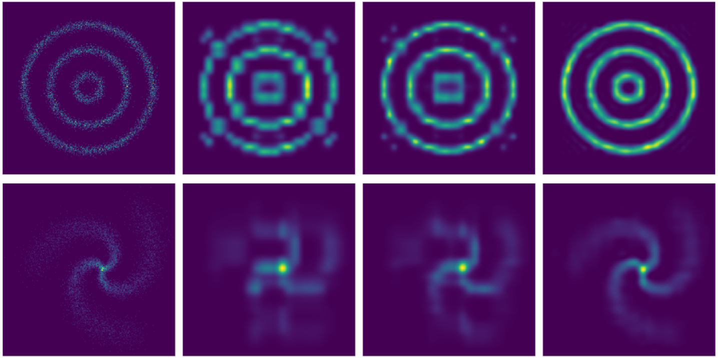
exponential separation

Theorem: there is a class of non-negative functions $\mathcal{F}$ over variables $\mathbf{X}$ that can be represented by compact a squared non-monotonic str-dec PC but for which the smallest monotonic str-dec PC computing $F \in \mathcal{F}$ has size $2^{\Omega(|\mathbf{X}|)}$ ***smallest monotonic str-dec PC computing $F \in \mathcal{F}$ has exponential size $2^{\Omega(|\mathbf{X}|)}$***.

$$\text{UDISJ}_G(\mathbf{X}_v) := \left(1 - \sum_{uv \in E} X_u X_v\right)^2$$

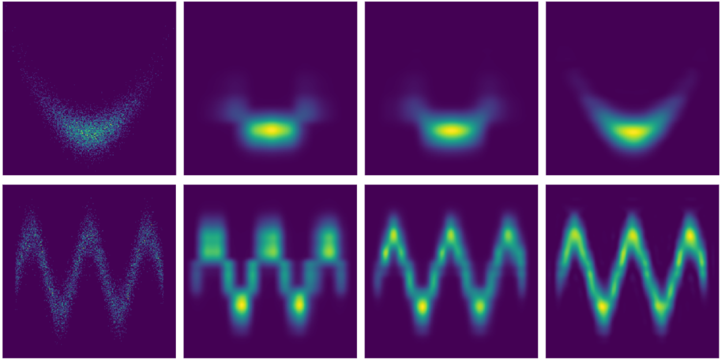
$\mathbf{Y} \backslash \mathbf{Z}$	000	100	010	001	110	101	011	111
000	1	1	1	1	1	1	1	1
100	1	0	1	1	0	0	1	0
010	1	1	0	1	0	1	0	0
001	1	1	1	0	1	0	0	0
110	1	0	0	1	1	0	0	1
101	1	0	1	0	0	1	0	1
011	1	1	0	0	0	0	1	1
111	1	0	0	0	1	1	1	4

more expressive?



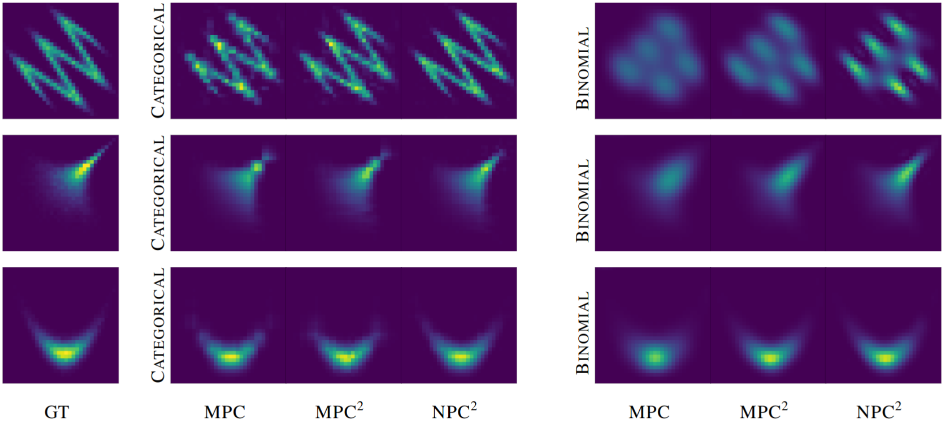
data $\quad$ monoPC $\quad$ monoPC^2 $\quad$ $\mathrm{non-monoPC}^2$

more expressive?



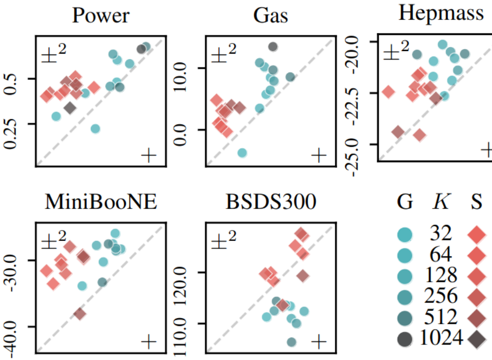
data | monoPC | monoPC^2 | $\text{non}-\text{monoPC}^2$

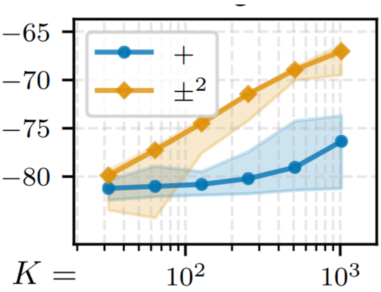
more expressive?



how more expressive?

for the ML crowd





why subtractions in mixture models

how to represent them as deep squared circuits?

what inference and model classes they support?

when are they more expressive

open problems

???

I how to retrieve a latent variable semantics?

II how to perform structure learning?

III more expressive than other circuit classes?

IV use logic-as-circuits for physics

???

I how to retrieve a latent variable semantics?

II how to perform structure learning?

III more expressive than other circuit classes?

IV use logic-as-circuits for physics

???

I how to retrieve a latent variable semantics?

II how to perform structure learning?

III more expressive than other circuit classes?

IV use logic-as-circuits for physics

???

I how to retrieve a latent variable semantics?

II how to perform structure learning?

III more expressive than other circuit classes?

IV use logic-as-circuits for physics

TCS crowd **ML crowd**

a circuit lowerbound to play with

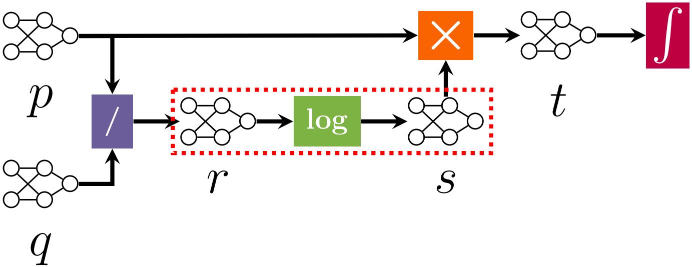
connections with mixtures/PGMs/learning

ML crowd **TCS crowd**

(some) new tractable model(s) to play with

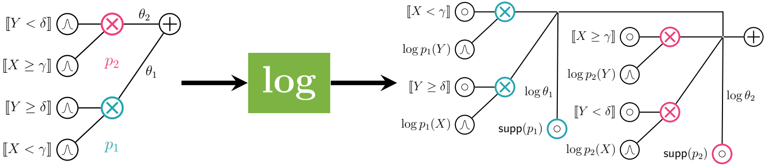
a tensorized way to represent circuits

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



build a LEGO-like query calculus...

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



property A*, *property B
property C

property A*, *property B

automating probabilistic reasoning

The TCS perspective

	Query	Tract. Conditions
Cross Entropy	$-\int p(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\mathbf{X}$	Cmp, q Det
Shannon Entropy	$-\sum p(\boldsymbol{x}) \log p(\boldsymbol{x})$	Sm, Dec, Det
Rényi Entropy	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	SD
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}_{+}$	Sm, Dec, Det
Mutual Information	$\int p(\boldsymbol{x}, \boldsymbol{y}) \log(p(\boldsymbol{x}, \boldsymbol{y})/(p(\boldsymbol{x})p(\boldsymbol{y})))$	Sm, SD, Det*
Kullback-Leibler Div.	$\int p(\boldsymbol{x}) \log(p(\boldsymbol{x})/q(\boldsymbol{x})) d\mathbf{X}$	Cmp, Det
Rényi's Alpha Div.	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{N}$	Cmp, q Det
	$(1-\alpha)^{-1} \log \int p^{\alpha}(\boldsymbol{x}) q^{1-\alpha}(\boldsymbol{x})\, d\mathbf{X}, \alpha \in \mathbb{R}$	Cmp, Det
Itakura-Saito Div.	$\int [p(\boldsymbol{x})/q(\boldsymbol{x}) - \log(p(\boldsymbol{x})/q(\boldsymbol{x})) - 1] d\, \mathbf{X}$	Cmp, Det
Cauchy-Schwarz Div.	$-\log \frac{\int p(\boldsymbol{x}) q(\boldsymbol{x}) d\mathbf{X}}{\sqrt{\int p^2(\boldsymbol{x}) d\mathbf{X} \int q^2(\boldsymbol{x}) d\mathbf{X}}}$	Cmp
Squared Loss	$\int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\, \mathbf{X}$	Cmp

compositionally *derive the tractability of many more queries*

The TCS perspective

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

***proving hardness** for when some input properties are not satisfied*

UNREAL

ML models | Queries | Data

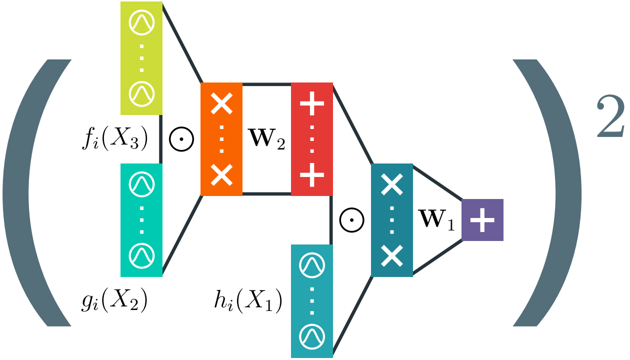
Distill | Compile | Learn

Computational abstractions

Reliable reasoning primitives

Hardware | Software

realizing a full “virtual machine” for reasoning



questions?