Tractable Computation of Expected Kernels by Circuits

Wenzhe Li*
Tsinghua University

Zhe Zeng*
University of California, Los Angeles

Antonio Vergari
University of California, Los Angeles

Guy Van den Broeck
University of California, Los Angeles

April 29th, 2021 - ***Yahoo Research Seminar***

Tractable Computation of Expected Kernels by Circuits

*Wenzhe Li**
Tsinghua University

*Zhe Zeng**
University of California, Los Angeles

Antonio Vergari
University of California, Los Angeles

Guy Van den Broeck
University of California, Los Angeles

April 29th, 2021 - ***Yahoo Research Seminar***

Tractable Computation of Expected Kernels by Circuits

Wenzhe Li*
Tsinghua University

Zhe Zeng*
University of California, Los Angeles

Antonio Vergari
University of California, Los Angeles

Guy Van den Broeck
University of California, Los Angeles

April 29th, 2021 - ***Yahoo Research Seminar***

Tractable Computation of Expected Kernels by Circuits

*Wenzhe Li**
Tsinghua University

*Zhe Zeng**
University of California, Los Angeles

Antonio Vergari
University of California, Los Angeles

Guy Van den Broeck
University of California, Los Angeles

April 29th, 2021 - ***Yahoo Research Seminar***

Motivation

A Fundamental Task

Given two distributions $\mathbf{p}$ and $\mathbf{q}$, and a kernel $\mathbf{k}$, the task is to compute the *expected kernel*

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

Motivation

A Fundamental Task

Given two distributions $\mathbf{p}$ and $\mathbf{q}$, and a kernel $\mathbf{k}$, the task is to compute the *expected kernel*

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

$\Longrightarrow$ *In kernel-based frameworks, expected kernels are omnipresent!*

Motivation

A Fundamental Task

Given two distributions $\mathbf{p}$ and $\mathbf{q}$, and a kernel $\mathbf{k}$, the task is to compute the *expected kernel*

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

$\Rightarrow$ *In kernel-based frameworks, expected kernels are omnipresent!*

squared Maximum Mean Discrepancy (MMD)

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{p}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] + \mathbb{E}_{\mathbf{x}\sim\mathbf{q},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] - 2\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

Motivation

A Fundamental Task

Given two distributions $\mathbf{p}$ and $\mathbf{q}$, and a kernel $\mathbf{k}$, the task is to compute the *expected kernel*

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

$\Longrightarrow$ *In kernel-based frameworks, expected kernels are omnipresent!*

Discrete Kernelized Stein Discrepancy (KDSD)

$$\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}_{\mathbf{p}}(\mathbf{x},\mathbf{x}')]$$

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\,d\mathbf{x}\,d\mathbf{x}'$$

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\, d\mathbf{x}\, d\mathbf{x}'$$

$\mathbf{p}, \mathbf{q}, \mathbf{k}$ fully factorized

$\mathbf{p}(\mathbf{x}) = \prod_i \mathbf{p}(x_i)$, $\mathbf{q}(\mathbf{x}) = \prod_i \mathbf{q}(x_i)$

$\mathbf{k}(\mathbf{x},\mathbf{x}') = \prod_i \mathbf{k}(x_i, x_i')$

$\Rightarrow$ expected kernel is **tractable**

$\prod_i (\int_{x_i, x_i'} \mathbf{p}(x_i)\mathbf{q}(x_i')\mathbf{k}(x_i, x_i'))$

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\, d\mathbf{x}\, d\mathbf{x}'$$

$\mathbf{p}, \mathbf{q}, \mathbf{k}$ fully factorized
$\mathbf{p}(\mathbf{x}) = \prod_i \mathbf{p}(x_i)$, $\mathbf{q}(\mathbf{x}) = \prod_i \mathbf{q}(x_i)$
$\mathbf{k}(\mathbf{x},\mathbf{x}') = \prod_i \mathbf{k}(x_i, x_i')$
$\Rightarrow$ expected kernel is **tractable**
$\prod_i (\int_{x_i,x_i'} \mathbf{p}(x_i)\mathbf{q}(x_i')\mathbf{k}(x_i,x_i'))$

A computation is **tractable** if it can be done exactly in polynomial time

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\,d\mathbf{x}\,d\mathbf{x}'$$

$\mathbf{p}, \mathbf{q}, \mathbf{k}$ fully factorized

PRO. Tractable exact computation

CON. Model being too restrictive

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\,d\mathbf{x}\,d\mathbf{x}'$$

$\mathbf{p},\mathbf{q},\mathbf{k}$ fully factorized

PRO. Tractable exact computation

CON. Model being too restrictive

Hard to compute in general.

$\Longrightarrow$ *approximate with MC or variational inference*

PRO. Efficient computation

CON. *no guarantees* on error bounds

Challenge

Reliability vs. Flexibility

$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')] = \int_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x},\mathbf{x}')\, d\mathbf{x}\, d\mathbf{x}'$$

$\mathbf{p}, \mathbf{q}, \mathbf{k}$ fully factorized

PRO. Tractable exact computation

CON. Model being too restrictive

trade-off?

Hard to compute in general.

$\Longrightarrow$ *approximate with MC or variational inference*

PRO. Efficient computation

CON. *no guarantees* on error bounds

Expressive distribution models

+

Exact computation of expected kernels?

Expressive distribution models

+

Exact computation of expectated kernels

=

Circuits!

Circuits

Probabilistic Circuits

deep generative models + deep guarantees

Circuits

Probabilistic Circuits

deep generative models + deep guarantees

Kernel Circuits

express kernels as circuits

Circuits

Probabilistic Circuits

deep generative models + deep guarantees

Kernel Circuits

express kernels as circuits

$$\Rightarrow \quad \mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x},\mathbf{x}')]$$

Probabilistic Circuits (PCs)

Tractable computational graphs

I. *A simple tractable distribution is a PC*

$\Rightarrow$ *e.g., a multivariate Gaussian*

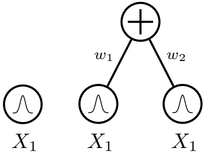
Probabilistic Circuits (PCs)

Tractable computational graphs

I. *A simple tractable distribution is a PC*

II. *A convex combination of PCs is a PC*

$\Rightarrow$ *e.g., a mixture model*



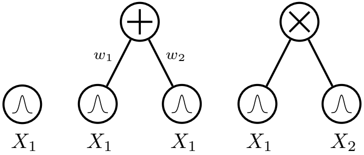
Probabilistic Circuits (PCs)

Tractable computational graphs

I. *A simple tractable distribution is a PC*

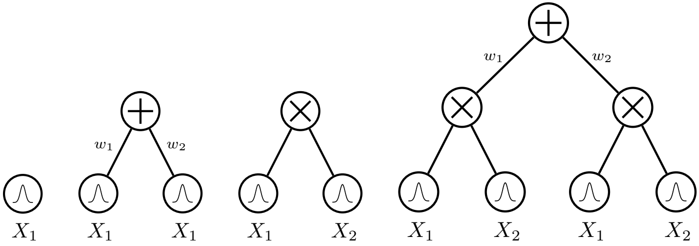
II. *A convex combination of PCs is a PC*

III. *A product of PCs is a PC*



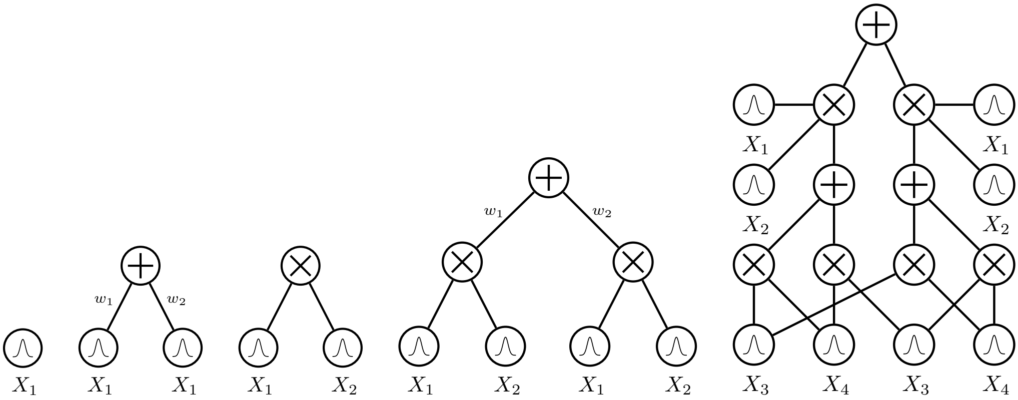
Probabilistic Circuits (PCs)

Tractable computational graphs



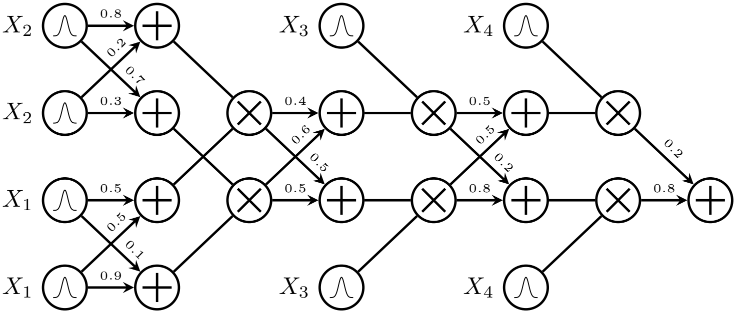
Probabilistic Circuits (PCs)

Tractable computational graphs



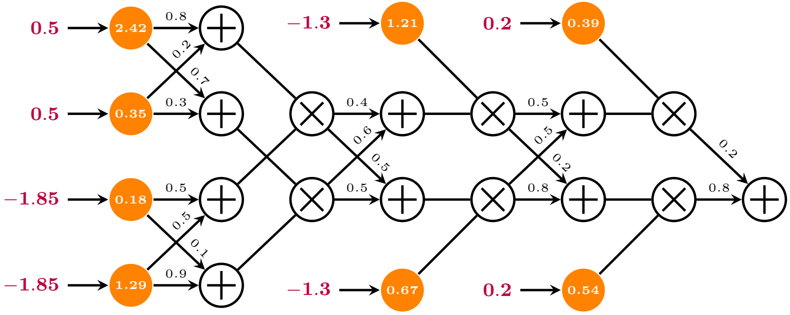
Probabilistic queries = feedforward evaluation

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



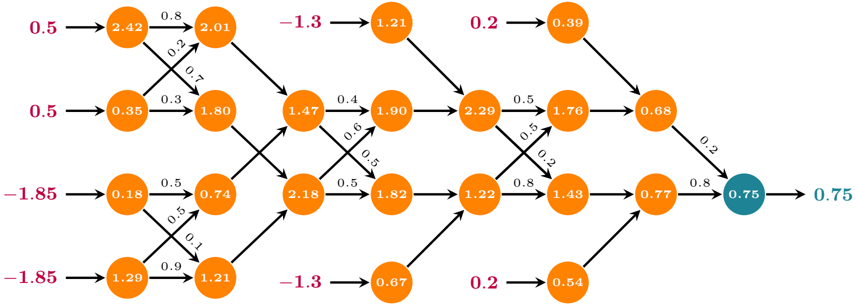
Probabilistic queries = feedforward evaluation

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



Probabilistic queries = feedforward evaluation

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



PCs = *deep learning*

PCs are computational graphs

PCs = *deep learning*

PCs are computational graphs encoding ***deep mixture models***

$\Longrightarrow$ *stacking (categorical) latent variables*

PCs = *deep learning*

PCs are computational graphs encoding ***deep mixture models***

$\Longrightarrow$ *stacking (categorical) latent variables*

PCs compactly represent ***polynomials with exponentially many terms***

$\Longrightarrow$ *universal approximators*

PCs = *deep learning*

PCs are computational graphs encoding ***deep mixture models***

$\Longrightarrow$ *stacking (categorical) latent variables*

PCs compactly represent ***polynomials with exponentially many terms***

$\Longrightarrow$ *universal approximators*

PCs are expressive *deep generative models*!

$\Longrightarrow$ *we can learn PCs with millions of parameters in minutes on the GPU [Peharz et al. 2020]*

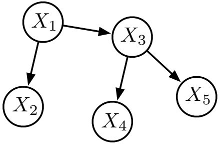
On par with intractable models!

How expressive are PCs?

dataset	best circuit	BN	MADE	VAE	dataset	best circuit	BN	MADE	VAE
nltcs	**-5.99**	-6.02	-6.04	**-5.99**	***dna***	**-79.88**	-80.65	-82.77	-94.56
msnbc	**-6.04**	**-6.04**	-6.06	-6.09	***kosarek***	**-10.52**	-10.83	-	-10.64
kdd	-2.12	-2.19	**-2.07**	-2.12	***msweb***	-9.62	-9.70	**-9.59**	-9.73
plants	**-11.84**	-12.65	-12.32	-12.34	***book***	-33.82	-36.41	-33.95	**-33.19**
audio	-39.39	-40.50	-38.95	**-38.67**	***movie***	-50.34	-54.37	-48.7	**-47.43**
jester	-51.29	**-51.07**	-52.23	-51.54	***webkb***	-149.20	-157.43	-149.59	**-146.9**
netflix	-55.71	-57.02	-55.16	**-54.73**	***cr52***	-81.87	-87.56	-82.80	**-81.33**
accidents	-26.89	**-26.32**	-26.42	-29.11	***c20ng***	-151.02	-158.95	-153.18	**-146.9**
retail	**-10.72**	-10.87	-10.81	-10.83	***bbc***	**-229.21**	-257.86	-242.40	-240.94
pumbs*	-22.15	**-21.72**	-22.3	-25.16	***ad***	-14.00	-18.35	**-13.65**	-18.81

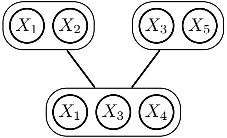
Peharz et al., "Random sum-product networks: A simple but effective approach to probabilistic deep learning", 2019

Unifying existing tractable models



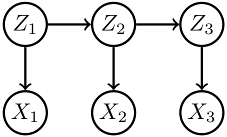
Chow-Liu trees

[Chow and Liu 1968]



Junction trees

[Bach and Jordan 2001]

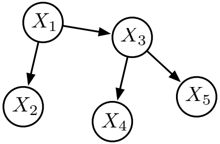


HMMs

[Rabiner and Juang 1986]

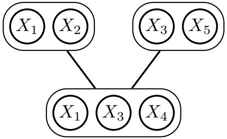
Classical tractable models can be compactly represented as PCs

Dang et al., "Strudel: Learning Structured-Decomposable Probabilistic Circuits", 2020



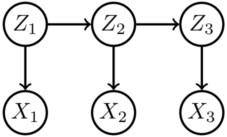
Chow-Liu trees

[Chow and Liu 1968]



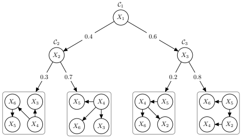
Junction trees

[Bach and Jordan 2001]



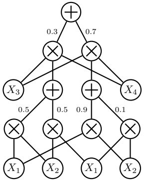
HMMs

[Rabiner and Juang 1986]



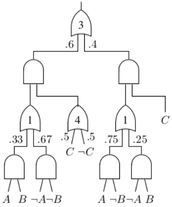
CNets

[Rahman et al. 2014]



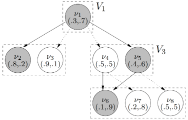
SPNs

[Poon et al. 2011]



PSDDs

[Kisa et al. 2014]



PDGs

[Jaeger 2004]

PCs = ***deep learning*** + ***deep guarantees***

PCs are expressive *deep generative models*!

&

Certifying tractability for a class of queries

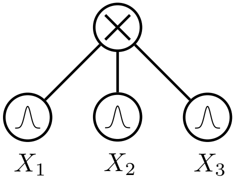
=

verifying structural properties of the graph

Which structural constraints ensure tractability?

decomposable + *smooth* PCs

A PC is ***decomposable*** if all inputs of product units depend on disjoint sets of variables

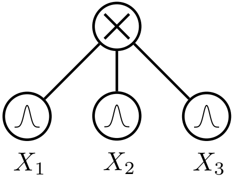


decomposable circuit

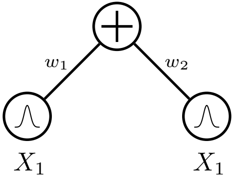
Darwiche and Marquis, "A knowledge compilation map", 2002

decomposable **+** ***smooth*** **PCs**

A PC is ***decomposable*** if all inputs of product units depend on disjoint sets of variables
A PC is ***smooth*** if all inputs of sum units depend of the same variable sets



decomposable circuit



smooth circuit

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

decomposable + *smooth* PCs = ...

Choi et al., "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling", 2020

decomposable + ***smooth*** PCs = ...

MAR ***sufficient*** *and* ***necessary*** *conditions for computing any marginal*

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

$\Longrightarrow$ *by a single feedforward evaluation*

Choi et al., "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling", 2020

decomposable + smooth PCs = ...

MAR ***sufficient*** *and* ***necessary*** *conditions for computing any marginal* $\int p(\mathbf{z}, \mathbf{y})\, d\mathbf{Z}$

CON ***sufficient*** *and* ***necessary*** *conditions for any conditional distribution*

$$p(\mathbf{y} \mid \mathbf{z}) = \frac{\int_{\mathsf{val}(\mathbf{H})} p(\mathbf{z}, \mathbf{y}, \mathbf{h})\, d\boldsymbol{H}}{\int_{\mathsf{val}(\mathbf{H})} \int_{\mathsf{val}(\mathbf{Y})} p(\mathbf{z}, \mathbf{y}, \mathbf{h})\, d\mathrm{H}\, d\mathbf{Y}}, \quad \forall \mathbf{Z}, \mathbf{Y} \subseteq \mathbf{X}$$

$\Longrightarrow$ *by* ***two*** *feedforward evaluations*

Choi et al., "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling", 2020

decomposable + smooth PCs = ...

MAR ***sufficient*** *and* ***necessary*** *conditions for computing any marginal* $\int p(\mathbf{z}, \mathbf{y})\, d\mathbf{Z}$

CON ***sufficient*** *and* ***necessary*** *conditions for any conditional* $\frac{\int p(\mathbf{z},\mathbf{y},\mathbf{h})\, d\boldsymbol{H}}{\int\int p(\mathbf{z},\mathbf{y},\mathbf{h})\, d\mathbf{H}\, d\mathbf{Y}}$

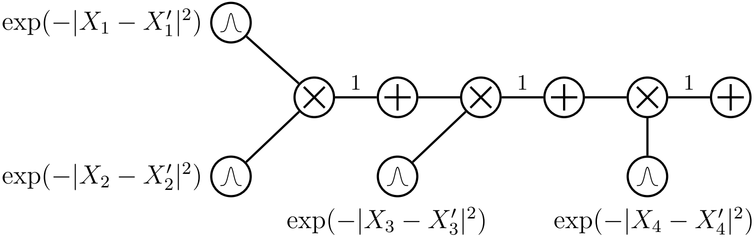
? *What about the* ***expected kernel*** $\mathbb{E}_{\mathbf{x}\sim\mathbf{p},\mathbf{x}'\sim\mathbf{q}}[\mathbf{k}(\mathbf{x}, \mathbf{x}')]$*?*

Choi et al., "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling", 2020

Can we represent kernels as circuits to characterize tractability of its queries?

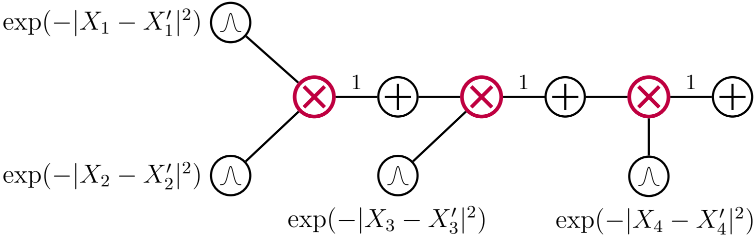
Kernel Circuits (KCs)

Exa. Radial basis function (RBF) kernel $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \exp\left(-\sum_{i=1}^{4} \mid X_i - X_i' \mid^2\right)$



Kernel Circuits (KCs)

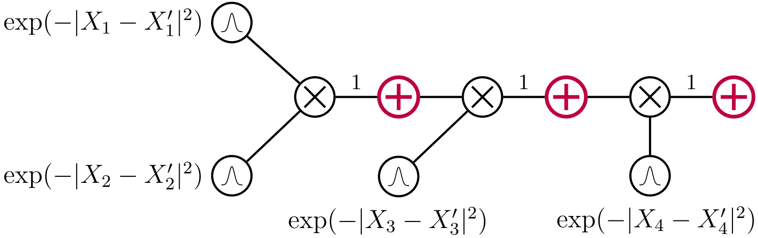
Exa. Radial basis function (RBF) kernel $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \exp\left(-\sum_{i=1}^{4} \mid X_i - X_i' \mid^2\right)$



decomposable if all inputs of product units depend on disjoint sets of variables

Kernel Circuits (KCs)

Exa. Radial basis function (RBF) kernel $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \exp\left(-\sum_{i=1}^{4} \mid X_i - X_i' \mid^2\right)$



decomposable if all inputs of product units depend on disjoint sets of variables

smooth if all inputs of sum units depend of the same variable sets

Kernel Circuits (KCs)

Common kernels can be compactly represented as decomposable + smooth KCs:

RBF, (exponentiated) Hamming, polynomial ...

Expected Kernel

tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

Expected Kernel

tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**

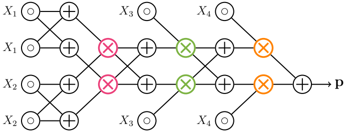
$\Longrightarrow$ *decompose in the same way*

Expected Kernel

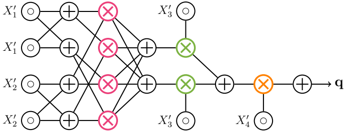
tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

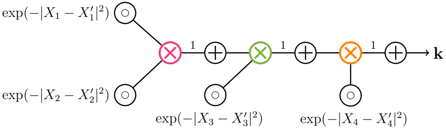
ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**



$\{X_1\}\{X_2\}$



$\{X'_1\}\{X'_2\}$



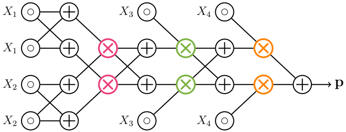
$\{(X_1, X'_1)\}\{(X_2, X'_2)\}$

Expected Kernel

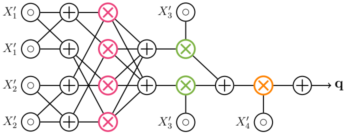
tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

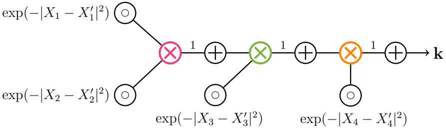
ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**



$\{X_1, X_2\}\{X_3\}$



$\{X_1', X_2'\}\{X_3'\}$



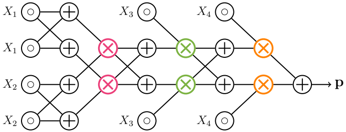
$\{(X_1, X_1'), (X_2, X_2')\}\{(X_3, X_3')\}$

Expected Kernel

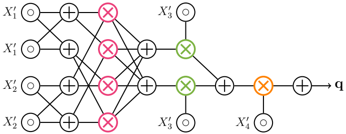
tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

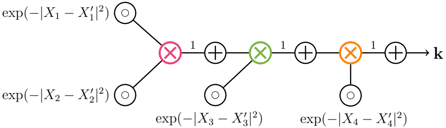
ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**



$\{X_1, X_2, X_3\}\{X_4\}$



$\{X_1', X_2', X_3'\}\{X_4'\}$



$\{(X_1, X_1'), (X_2, X_2'), (X_3, X_3')\}\{(X_4, X_4')\}$

Expected Kernel

tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**

Expected Kernel

tractable computation via circuit operations

i) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **decomposable** + **smooth**

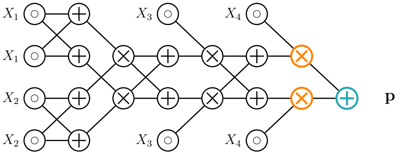
ii) PCs $\mathbf{p}$ and $\mathbf{q}$, and KC $\mathbf{k}$ are **compatible**

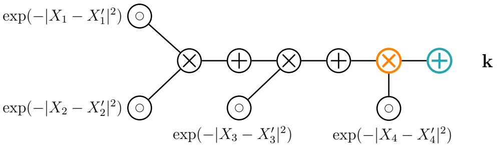
Then computing expected kernels can be done ***tractably*** by a forward pass

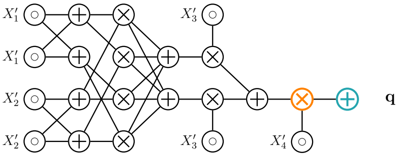
$\Longrightarrow$ *product of the sizes of each circuit!*

smooth + **decomposable** + **compatible** = **tractable E[k]**

[Sum Nodes] $\mathbf{p}(\mathbf{X}) = \sum_i w_i \mathbf{p}_i(\mathbf{X})$, $\mathbf{q}(\mathbf{X}') = \sum_j w'_j \mathbf{q}_j(\mathbf{X}')$, and kernel $\mathbf{k}(\mathbf{X}, \mathbf{X}') = \sum_l w''_l \mathbf{k}_l(\mathbf{X}, \mathbf{X}')$:

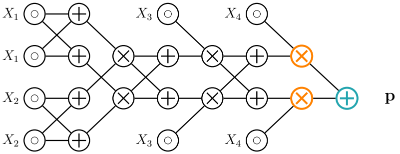


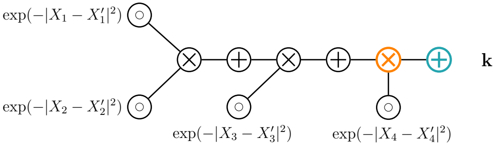


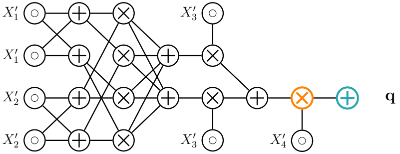


smooth + decomposable + compatible = tractable E[k]

[Sum Nodes] $\mathbf{p}(\mathbf{X}) = \sum_i w_i \mathbf{p}_i(\mathbf{X})$, $\mathbf{q}(\mathbf{X}') = \sum_j w'_j \mathbf{q}_j(\mathbf{X}')$, and kernel $\mathbf{k}(\mathbf{X}, \mathbf{X}') = \sum_l w''_l \mathbf{k}_l(\mathbf{X}, \mathbf{X}')$:





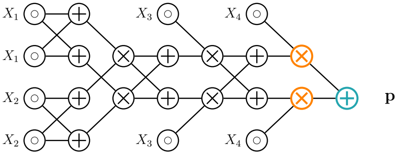


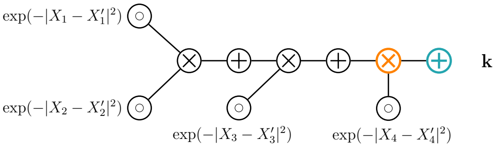
$$\sum_{\mathbf{x},\mathbf{x}'} \mathbf{p}(\mathbf{x})\mathbf{q}(\mathbf{x}')\mathbf{k}(\mathbf{x}, \mathbf{x}')$$

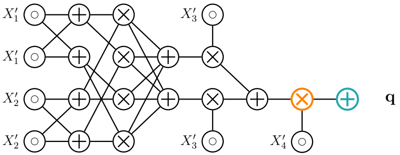
$$= \sum_{i,j,l} w_i w'_j w''_l \mathbf{p}_i(\mathbf{x})\mathbf{q}_j(\mathbf{x})\mathbf{k}_l(\mathbf{x}, \mathbf{x}')$$

smooth + **decomposable** + **compatible** = **tractable E[k]**

[Sum Nodes] $\mathbf{p}(\mathbf{X}) = \sum_i w_i \mathbf{p}_i(\mathbf{X})$, $\mathbf{q}(\mathbf{X}') = \sum_j w'_j \mathbf{q}_j(\mathbf{X}')$, and kernel $\mathbf{k}(\mathbf{X}, \mathbf{X}') = \sum_l w''_l \mathbf{k}_l(\mathbf{X}, \mathbf{X}')$:





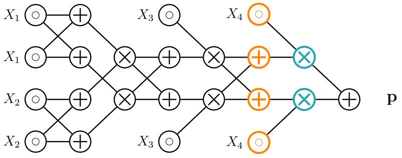


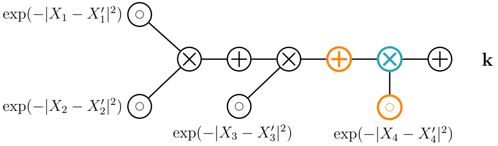
$$\mathbb{E}_{\mathbf{p},\mathbf{q}}[\mathbf{k}(\mathbf{x}, \mathbf{x}')] = \sum_{i,j,l} w_i w'_j w''_l \mathbb{E}_{\mathbf{p}_i,\mathbf{q}_j}[\mathbf{k}_l(\mathbf{x}, \mathbf{x}')]$$

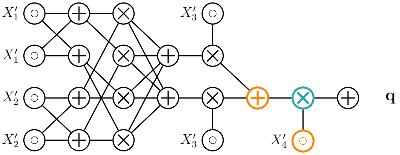
$\Longrightarrow$ *expectation is "pushed down" to children*

smooth + **decomposable** + **compatible** = **tractable E[k]**

[Product Nodes] $\mathbf{p}_\times(\mathbf{X}) = \prod_i \mathbf{p}_i(\mathbf{X}_i)$, $\mathbf{q}_\times(\mathbf{X}') = \prod_i \mathbf{q}_j(\mathbf{X}'_i)$, and kernel $\mathbf{k}_\times(\mathbf{X}, \mathbf{X}') = \prod_i \mathbf{k}_i(\mathbf{X}_i, \mathbf{X}'_i)$:

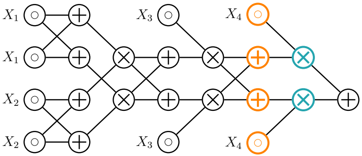




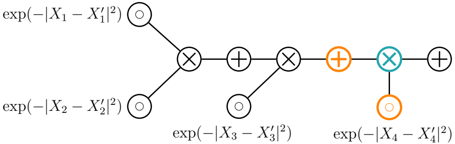


smooth + **decomposable** + **compatible** = **tractable E[k]**

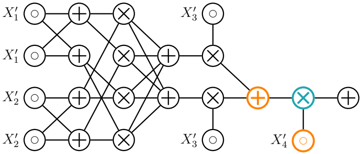
[Product Nodes] $\mathbf{p}_\times(\mathbf{X}) = \prod_i \mathbf{p}_i(\mathbf{X}_i)$, $\mathbf{q}_\times(\mathbf{X}') = \prod_i \mathbf{q}_j(\mathbf{X}'_i)$, and kernel $\mathbf{k}_\times(\mathbf{X}, \mathbf{X}') = \prod_i \mathbf{k}_i(\mathbf{X}_i, \mathbf{X}'_i)$:



$\mathbf{p}$



$\mathbf{k}$

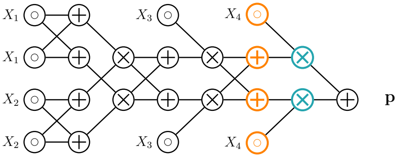


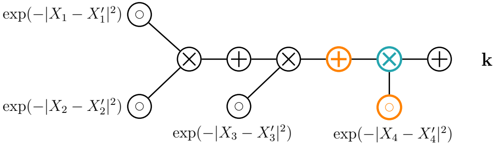
$\mathbf{q}$

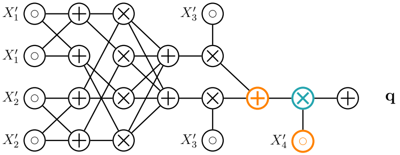
$$
\begin{aligned}
&\sum_{\mathbf{x},\mathbf{x}'} \mathbf{p}_\times(\mathbf{x})\mathbf{q}_\times(\mathbf{x}')\mathbf{k}_\times(\mathbf{x}, \mathbf{x}') \\
&= \sum_{\mathbf{x},\mathbf{x}'} \prod_i \mathbf{p}(\mathbf{x}_i)\mathbf{q}(\mathbf{x}_i)\mathbf{k}_i(\mathbf{x}_i, \mathbf{x}'_i) \\
&= \prod_i \Big(\sum_{\mathbf{x}_i,\mathbf{x}'_i} \mathbf{p}(\mathbf{x}_i)\mathbf{q}(\mathbf{x}_i)\mathbf{k}_i(\mathbf{x}_i, \mathbf{x}'_i)\Big)
\end{aligned}
$$

smooth + *decomposable* + *compatible* = *tractable E[k]*

[Product Nodes] $\mathbf{p}_\times(\mathbf{X}) = \prod_i \mathbf{p}_i(\mathbf{X}_i)$, $\mathbf{q}_\times(\mathbf{X}') = \prod_i \mathbf{q}_j(\mathbf{X}'_i)$, and kernel $\mathbf{k}_\times(\mathbf{X}, \mathbf{X}') = \prod_i \mathbf{k}_i(\mathbf{X}_i, \mathbf{X}'_i)$:







$$\mathbb{E}_{\mathbf{p}_\times, \mathbf{q}_\times}[\mathbf{k}_\times(\mathbf{x}, \mathbf{x}')] = \prod_i \mathbb{E}_{\mathbf{p},\mathbf{q}}[\mathbf{k}(\mathbf{x}_i, \mathbf{x}'_i)]$$

$\Longrightarrow$ *expectation decomposes into easier ones*

smooth + **decomposable** + **compatible** = **tractable E[k]**

Algorithm 1 $\mathbb{E}_{\mathbf{p}_n,\mathbf{q}_m}[\mathbf{k}_l]$ — Computing the expected kernel

Input: Two compatible PCs $\mathbf{p}_n$ and $\mathbf{q}_m$, and a KC $\mathbf{k}_l$ that is kernel-compatible with the PC pair $\mathbf{p}_n$ and $\mathbf{q}_m$.

1: **if** m, n, l are ***input*** nodes **then**
2: **return** $\mathbb{E}_{\mathbf{p}_n,\mathbf{q}_m}[\mathbf{k}_l]$
3: **else if** m, n, l are ***sum*** nodes **then**
4: **return** $\sum_{i\in\mathsf{in}(n),j\in\mathsf{in}(m),c\in\mathsf{in}(l)} w_i w'_j w''_c \, \mathbb{E}_{\mathbf{p}_i,\mathbf{q}_j}[\mathbf{k}_c]$
5: **else if** m, n, l are ***product*** nodes **then**
6: **return** $\mathbb{E}_{\mathbf{p}_{n_\mathsf{L}},\mathbf{q}_{m_\mathsf{L}}}[\mathbf{k}_\mathsf{L}] \cdot \mathbb{E}_{\mathbf{p}_{n_\mathsf{R}},\mathbf{q}_{m_\mathsf{R}}}[\mathbf{k}_\mathsf{R}]$

Computation can be done in one forward pass!

smooth + decomposable + compatible = tractable E[k]

Algorithm 2 $\mathbb{E}_{\mathbf{p}_n,\mathbf{q}_m}[\mathbf{k}_l]$ — Computing the expected kernel

Input: Two compatible PCs $\mathbf{p}_n$ and $\mathbf{q}_m$, and a KC $\mathbf{k}_l$ that is kernel-compatible with the PC pair $\mathbf{p}_n$ and $\mathbf{q}_m$.

1: **if** m, n, l are ***input*** nodes **then**
2: **return** $\mathbb{E}_{\mathbf{p}_n,\mathbf{q}_m}[\mathbf{k}_l]$
3: **else if** m, n, l are ***sum*** nodes **then**
4: **return** $\sum_{i\in\mathsf{in}(n),j\in\mathsf{in}(m),c\in\mathsf{in}(l)} w_i w'_j w''_c \, \mathbb{E}_{\mathbf{p}_i,\mathbf{q}_j}[\mathbf{k}_c]$
5: **else if** m, n, l are ***product*** nodes **then**
6: **return** $\mathbb{E}_{\mathbf{p}_{n_\mathsf{L}},\mathbf{q}_{m_\mathsf{L}}}[\mathbf{k}_\mathsf{L}] \cdot \mathbb{E}_{\mathbf{p}_{n_\mathsf{R}},\mathbf{q}_{m_\mathsf{R}}}[\mathbf{k}_\mathsf{R}]$

Computation can be done in one forward pass!

$\Longrightarrow$ *squared maximum mean discrepancy* $MMD[\mathbf{p}, \mathbf{q}]$ *[Gretton et al. 2012]*

$\Longrightarrow$ + *determinism, kernelized discrete Stein discrepancy (KDSD) [Yang et al. 2018]*

Applications

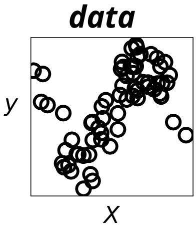
- ***Support vector regression with missing features***

Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

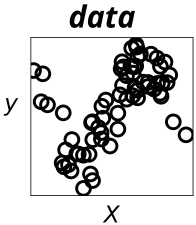
Support vector regression with missing features

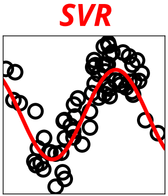
- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.



Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

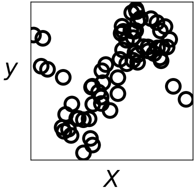




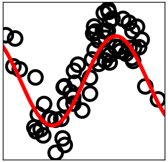
Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

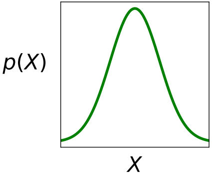
data



SVR

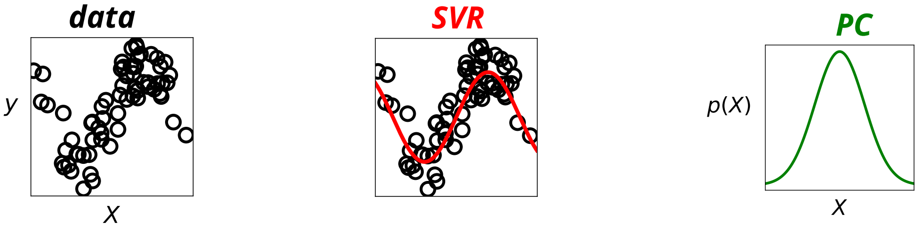


PC



Support vector regression with missing features

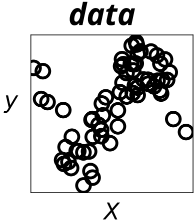
- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

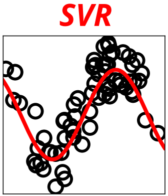


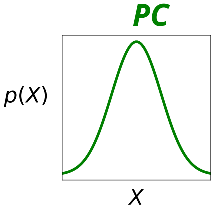
At deployment time, what happen if we observe partial features and some are missing?

Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.







At deployment time, what happen if we observe partial features and some are missing?

$\Longrightarrow$ ***Expected prediction!***

Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

At deployment time, in the case when only features $\mathbf{X}_o = \mathbf{x}_o$ are *observed* and features $\mathbf{X}_m$ are *missing*, with $\mathbf{X} = (\mathbf{X}_o, \mathbf{X}_m)$, the expected prediction is

Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

At deployment time, in the case when only features $\mathbf{X}_o = \mathbf{x}_o$ are *observed* and features $\mathbf{X}_m$ are *missing*, with $\mathbf{X} = (\mathbf{X}_o, \mathbf{X}_m)$, the expected prediction is

$$\mathbb{E}_{\mathbf{x}_m \sim \mathbf{p}(\mathbf{X}_m | \mathbf{x}_o)} [f(\mathbf{x}_o, \mathbf{x}_m)]$$

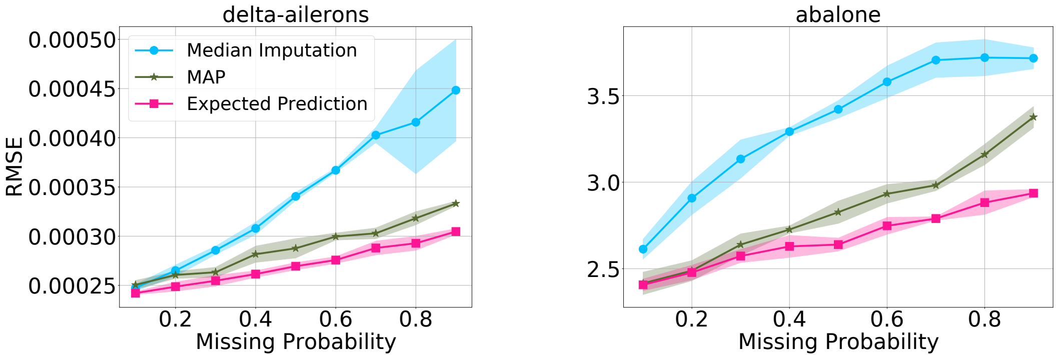
Support vector regression with missing features

- Given training data,
- we can learn a *support vector regression (SVR) model* $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}) + b$;
- also we can learn a *generative model* for features in PC $\mathbf{p}(\mathbf{X})$.

At deployment time, in the case when only features $\mathbf{X}_o = \mathbf{x}_o$ are *observed* and features $\mathbf{X}_m$ are *missing*, with $\mathbf{X} = (\mathbf{X}_o, \mathbf{X}_m)$, the expected prediction is

$$\mathbb{E}_{\mathbf{x}_m \sim \mathbf{p}(\mathbf{X}_m|\mathbf{x}_o)}[f(\mathbf{x}_o, \mathbf{x}_m)] = \sum_{i=1}^{m} w_i \mathbb{E}_{\mathbf{x}_m \sim \mathbf{p}(\mathbf{X}_m|\mathbf{x}_o)}[\mathbf{k}(\mathbf{x}_i, (\mathbf{x}_o, \mathbf{x}_m))] + b$$

Support vector regression with missing features



$\Longrightarrow$ *Expected prediction improves over the baselines*

Applications

- ***Support vector regression with missing features***
- ***Collapsed black-box importance sampling***

Recap **Black-box Importance Sampling** [Liu et al. 2016]

- Empirical KDSD $\mathbb{S}(\{\underbrace{w^{(i)}}_{\text{weights}}, \underbrace{\mathbf{x}^{(i)}}_{\text{samples}}\}_{i=1}^{n} \parallel \mathbf{p})$

$$\mathbb{S}^2(\{w^{(i)}, \mathbf{x}^{(i)}\}_{i=1}^{n} \parallel \mathbf{p}) = \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w}, \text{ with } [\boldsymbol{K}_p]_{ij} = k_p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Given a distribution $\mathbf{p}$, and samples $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$, the black-box importance sampling obtains weights by solving optimization problem

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Recap Black-box Importance Sampling [Liu et al. 2016]

- Empirical KDSD $\mathbb{S}(\{\underbrace{w^{(i)}}_{\text{weights}}, \underbrace{\mathbf{x}^{(i)}}_{\text{samples}}\}_{i=1}^{n} \parallel \mathbf{p})$

$$\mathbb{S}^2(\{w^{(i)}, \mathbf{x}^{(i)}\}_{i=1}^{n} \parallel \mathbf{p}) = \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w}, \ \text{ with } \ [\boldsymbol{K}_p]_{ij} = k_p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Given a distribution $\mathbf{p}$, and samples $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$, the black-box importance sampling obtains weights by solving optimization problem

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1, \ w_i \geq 0 \right\}$$

Recap Black-box Importance Sampling [Liu et al. 2016]

- Empirical KDSD $\mathbb{S}(\{\underbrace{w^{(i)}}_{\text{weights}}, \underbrace{\mathbf{x}^{(i)}}_{\text{samples}}\}_{i=1}^{n} \parallel \mathbf{p})$

$$\mathbb{S}^2(\{w^{(i)}, \mathbf{x}^{(i)}\}_{i=1}^{n} \parallel \mathbf{p}) = \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w}, \text{ with } [\boldsymbol{K}_p]_{ij} = k_p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Given a distribution $\mathbf{p}$, and samples $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$, the black-box importance sampling obtains weights by solving optimization problem

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left\{ \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Can we use less samples but maintain the same or even better performance?

Recap Black-box Importance Sampling [Liu et al. 2016]

- Empirical KDSD $\mathbb{S}(\{\underbrace{w^{(i)}}_{\text{weights}}, \underbrace{\mathbf{x}^{(i)}}_{\text{samples}}\}_{i=1}^{n} \parallel \mathbf{p})$

$$\mathbb{S}^2(\{w^{(i)}, \mathbf{x}^{(i)}\}_{i=1}^{n} \parallel \mathbf{p}) = \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w}, \ \text{with} \ [\boldsymbol{K}_p]_{ij} = k_p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Given a distribution $\mathbf{p}$, and samples $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$, the black-box importance sampling obtains weights by solving optimization problem

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left\{ \boldsymbol{w}^\top \boldsymbol{K_p} \boldsymbol{w} \;\middle|\; \sum_{i=1}^{n} w_i = 1, \ w_i \geq 0 \right\}$$

Can we use less samples but maintain the same or even better performance?

$\Longrightarrow$ ***Collapsed samples!***

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^n$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathbf{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs $\mathbf{p}_i$ by *knowledge compilation* *[Shen et al. 2016]*
- Compile the kernel function $\mathbf{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC $\mathbf{k}$
- Empirical KDSD between collapsed samples and the target distribution $\mathbf{p}$

$$\mathbb{S}_{\mathbf{s}}^2(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathbf{p}_i, \mathbf{x}_\mathbf{c}' \sim \mathbf{p}_j} \left[\mathbf{k}_p(\mathbf{x}, \mathbf{x}')\right]$

- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \;\middle|\; \sum_{i=1}^n w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^{n}$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathrm{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs p_i by *knowledge compilation* *[Shen et al. 2016]*
- Compile the kernel function $\mathrm{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC k
- Empirical KDSD between collapsed samples and the target distribution p

$$\mathbb{S}^2_{\mathbf{s}}(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

  with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathrm{p}_i, \mathbf{x}'_\mathbf{c} \sim \mathrm{p}_j} \left[\mathrm{k}_p(\mathbf{x}, \mathbf{x}')\right]$
- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^{n}$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathbf{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs $\mathbf{p}_i$ by *knowledge compilation* *[Shen et al. 2016]*
- Compile the kernel function $\mathbf{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC $\mathbf{k}$
- Empirical KDSD between collapsed samples and the target distribution $\mathbf{p}$

$$\mathbb{S}^2_{\mathbf{s}}(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathbf{p}_i, \mathbf{x}'_\mathbf{c} \sim \mathbf{p}_j}\left[\mathbf{k}_p(\mathbf{x}, \mathbf{x}')\right]$

- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^{n}$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathbf{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs $\mathbf{p}_i$ by *knowledge compilation* *[Shen et al. 2016]*
- Compile the kernel function $\mathbf{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC $\mathbf{k}$
- Empirical KDSD between collapsed samples and the target distribution $\mathbf{p}$

$$\mathbb{S}^2_{\mathbf{s}}(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathbf{p}_i, \mathbf{x}'_\mathbf{c} \sim \mathbf{p}_j} [\mathbf{k}_p(\mathbf{x}, \mathbf{x}')]$

- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^{n}$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathbf{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs $\mathbf{p}_i$ by *knowledge compilation* [Shen et al. 2016]
- Compile the kernel function $\mathbf{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC $\mathbf{k}$
- Empirical KDSD between collapsed samples and the target distribution $\mathbf{p}$

$$\mathbb{S}^2_{\mathbf{s}}(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathbf{p}_i, \mathbf{x}'_\mathbf{c} \sim \mathbf{p}_j} [\mathbf{k}_p(\mathbf{x}, \mathbf{x}')]$

- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \,\middle|\, \sum_{i=1}^{n} w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling

- Given partial samples $\{\mathbf{x_s}^{(i)}\}_{i=1}^n$, with $(\mathbf{X_s}, \mathbf{X_c})$ a partition of $\mathbf{X}$,
- Represent the conditional distributions $\mathbf{p}(\mathbf{X_c} \mid \mathbf{x_s}^{(i)})$ as PCs $\mathbf{p}_i$ by *knowledge compilation* *[Shen et al. 2016]*
- Compile the kernel function $\mathbf{k}(\mathbf{X_c}, \mathbf{X_c}')$ as KC $\mathbf{k}$
- Empirical KDSD between collapsed samples and the target distribution $\mathbf{p}$

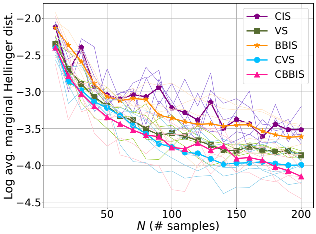
$$\mathbb{S}^2_{\mathbf{s}}(\{\mathbf{x_s}^{(i)}, w_i\} \parallel p) = \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w}$$

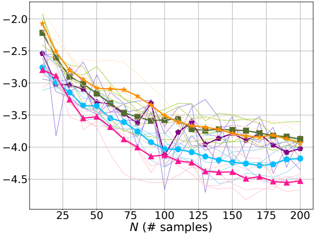
with $[\boldsymbol{K}_{p,\mathbf{s}}]_{ij} = \mathbb{E}_{\mathbf{x_c} \sim \mathbf{p}_i, \mathbf{x}'_\mathbf{c} \sim \mathbf{p}_j} [\mathbf{k}_p(\mathbf{x}, \mathbf{x}')]$

- Finally, obtain the importance weights $\boldsymbol{w}$ by solving

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left\{ \boldsymbol{w}^\top \boldsymbol{K}_{p,\mathbf{s}} \boldsymbol{w} \;\middle|\; \sum_{i=1}^n w_i = 1,\ w_i \geq 0 \right\}$$

Collapsed Black-box Importance Sampling





$\Longrightarrow$ *methods with collapsed samples all outperform their non-collapsed counterparts*

$\Longrightarrow$ *CBBIS performs equally well or better than other baselines*

Friedman and Broeck, "Approximate Knowledge Compilation by Online Collapsed Importance Sampling", 2018

Liu and Lee, "Black-box importance sampling", 2016

Applications

- *Support vector regression with missing features*
- *Collapsed black-box importance sampling*

Conclusion

Takeaways

#1: you can be both tractable and expressive

#2: circuits are a foundation for tractable inference over kernels

What else?

What other applications would benefit from the tractable computation of the expected kernels?

More on circuits ...

Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models

`starai.cs.ucla.edu/papers/ProbCirc20.pdf`

Probabilistic Circuits: Representations, Inference, Learning and Theory

`youtube.com/watch?v=2RAG5-L9R70`

Probabilistic Circuits

`arranger1044.github.io/probabilistic-circuits/`

Foundations of Sum-Product Networks for probabilistic modeling

`tinyurl.com/w65po5d`

Questions?

References I

- Chow, C and C Liu (1968). “Approximating discrete probability distributions with dependence trees”. In: *IEEE Transactions on Information Theory* 14.3, pp. 462–467.
- Rabiner, Lawrence and Biinghwang Juang (1986). “An introduction to hidden Markov models”. In: *ieee assp magazine* 3.1, pp. 4–16.
- Bach, Francis R. and Michael I. Jordan (2001). “Thin Junction Trees”. In: *Advances in Neural Information Processing Systems 14*. MIT Press, pp. 569–576.
- Darwiche, Adnan and Pierre Marquis (2002). “A knowledge compilation map”. In: *Journal of Artificial Intelligence Research* 17, pp. 229–264.
- Jaeger, Manfred (2004). “Probabilistic decision graphs—combining verification and AI techniques for probabilistic inference”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 12.supp01, pp. 19–42.
- Kisa, Doga, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche (July 2014). “Probabilistic sentential decision diagrams”. In: *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. Vienna, Austria. URL: `http://starai.cs.ucla.edu/papers/KisaKR14.pdf`.
- Liu, Qiang and Jason D Lee (2016). “Black-box importance sampling”. In: *arXiv preprint arXiv:1610.05247*.
- Friedman, Tal and Guy Van den Broeck (Dec. 2018). “Approximate Knowledge Compilation by Online Collapsed Importance Sampling”. In: *Advances in Neural Information Processing Systems 31 (NeurIPS)*. URL: `http://starai.cs.ucla.edu/papers/FriedmanNeurIPS18.pdf`.
- Peharz, Robert, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani (2019). “Random sum-product networks: A simple but effective approach to probabilistic deep learning”. In: *UAI*.
- Choi, YooJung, Antonio Vergari, and Guy Van den Broeck (2020). “Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling”. In:
- Dang, Meihua, Antonio Vergari, and Guy Van den Broeck (2020). “Strudel: Learning Structured-Decomposable Probabilistic Circuits”. In: *PGM* abs/2007.09331.

References II

Peharz, Robert, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani (2020). "Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits". In: *International Conference of Machine Learning*.