Geometric Knowledge Graph Embeddings meet Probabilistic Circuits

- **Today:** Intro to Probabilistic Circuits and Knowledge Graph Embeddings
- **Tomorrow morning:** Intro by Lorenzo Loconte on "How to turn your knowledge graph embeddings into generative models" (NeurIPS'22)
- **After that:** coding, hacking, brainstorming

Slides: `https://github.com/smatmo/CoE-Summer-School`

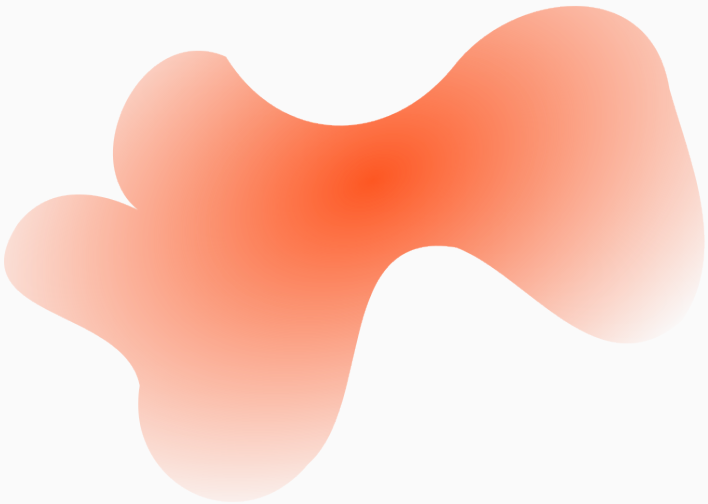
Probabilistic Circuits

Robert Peharz
Graz University of Technology

Bilateral AI Summer School
Klagenfurt, 8th July 2025

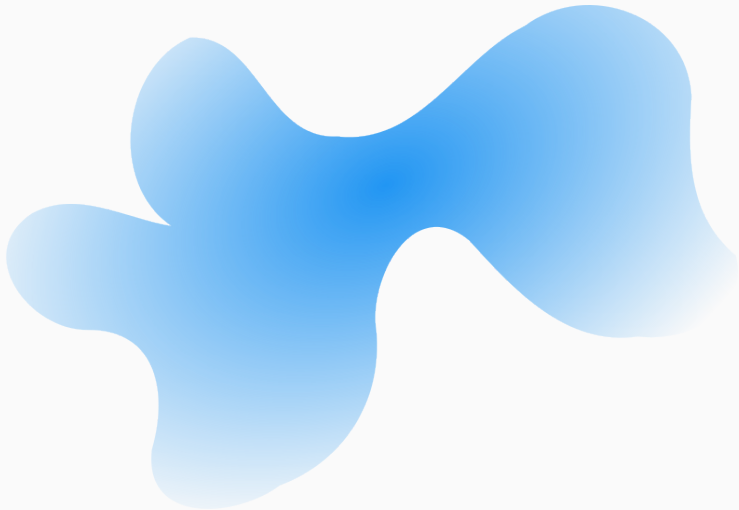
Probabilistic Modelling in a Nutshell

Model Distribution $\mathbb{P}_\theta$, p_θ



$\approx$

True Distribution $\mathbb{P}^*$, p^*

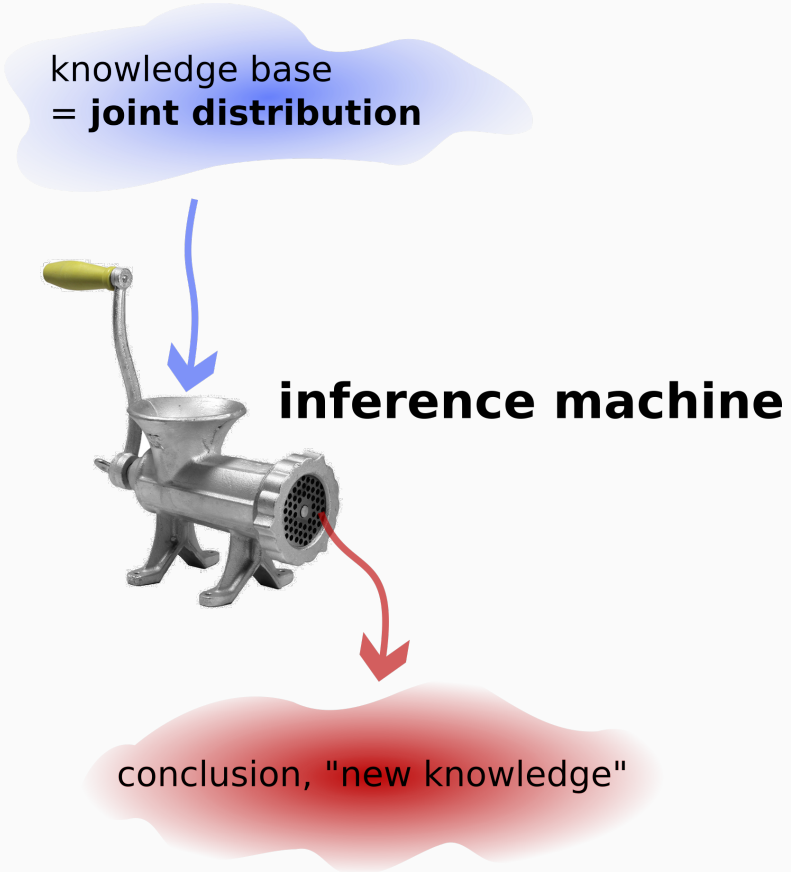


Why would we want to do that?

Generative Modelling

A cute kitty riding a unicycle while juggling with chainsaws

Reasoning under Uncertainty



Cat vs. Dog classifier

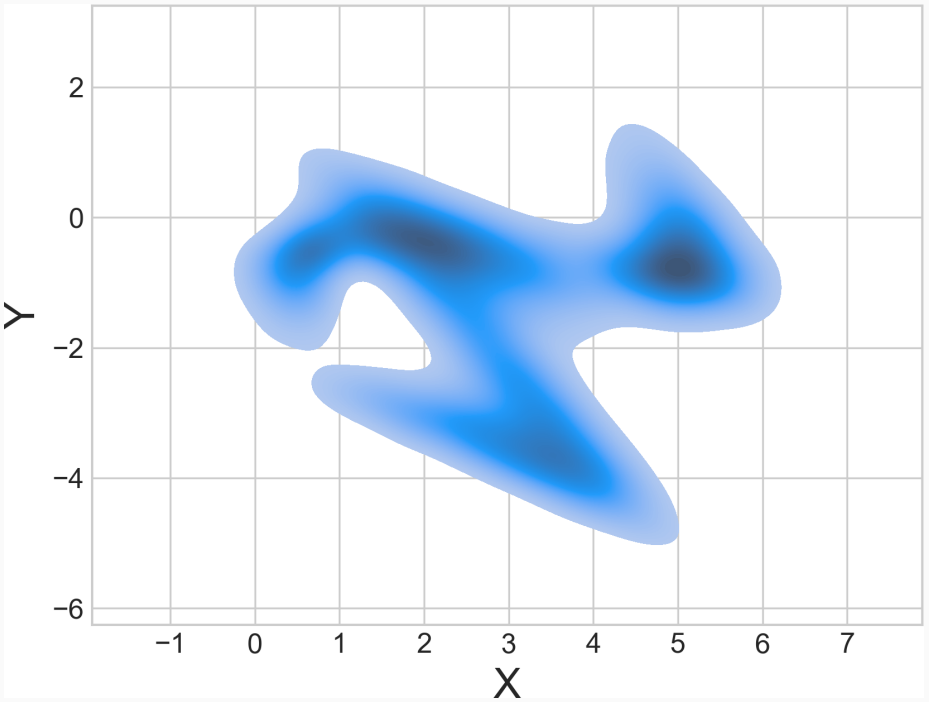
If you know the true distribution $p^*(\boldsymbol{x}, y)$ producing i.i.d. samples of $\boldsymbol{x}$ (image) and y (label), the **Bayes optimal classifier** is

$$\hat{y} = \arg\max_y p^*(y \mid \boldsymbol{x}) = \arg\max_y p^*(y, \boldsymbol{x})$$

No classifier has a higher accuracy.

Regression

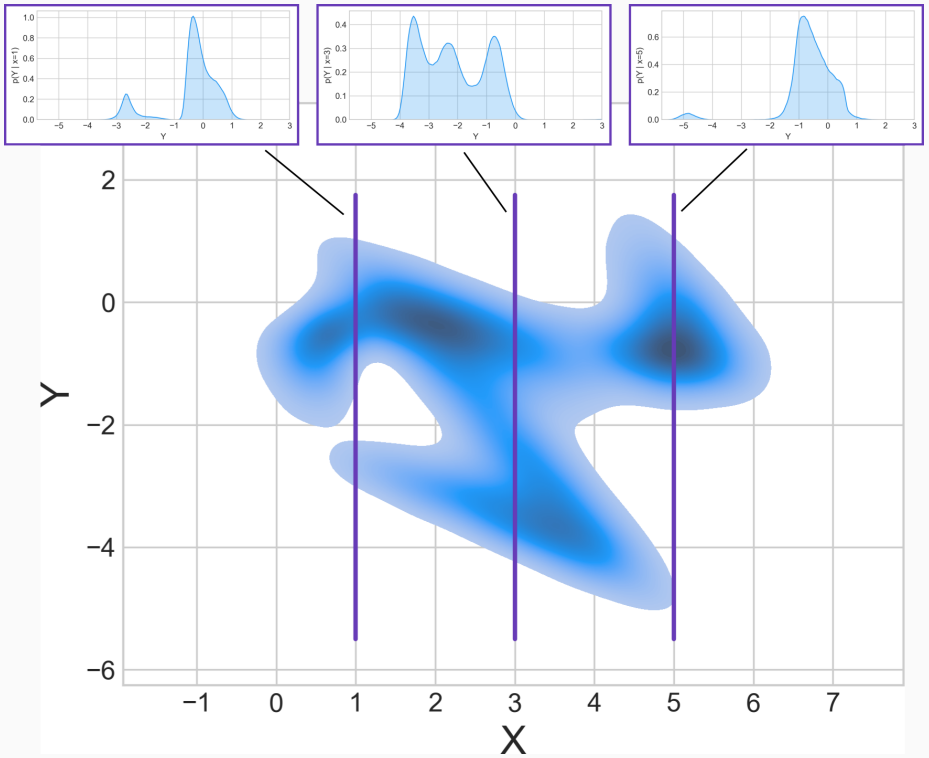
Say you want to predict Y from X, and you know the true distribution $p^*(y, x)$ producing i.i.d. samples of X, Y.



You might compute the **conditional distribution**

$$p^*(y \mid x) = \frac{p^*(y, x)}{\underbrace{p^*(x)}_{= \int p^*(x,y)\,\mathrm{d}y}}$$

for all values of x.

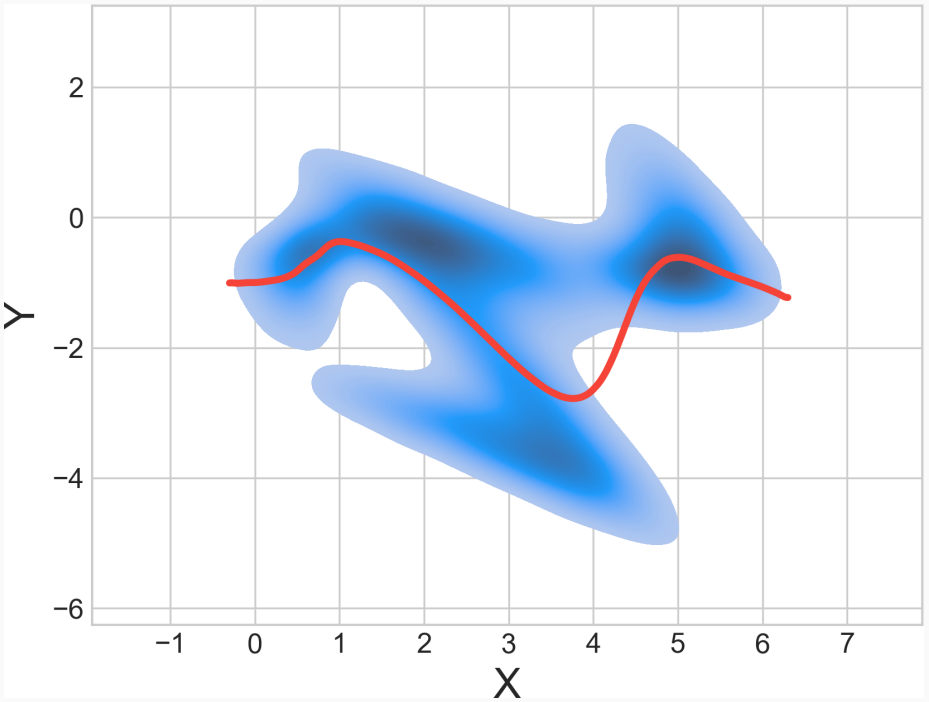


Then you can construct the **true regression function**

$$f(x) := \mathbb{E}[Y \mid x]$$

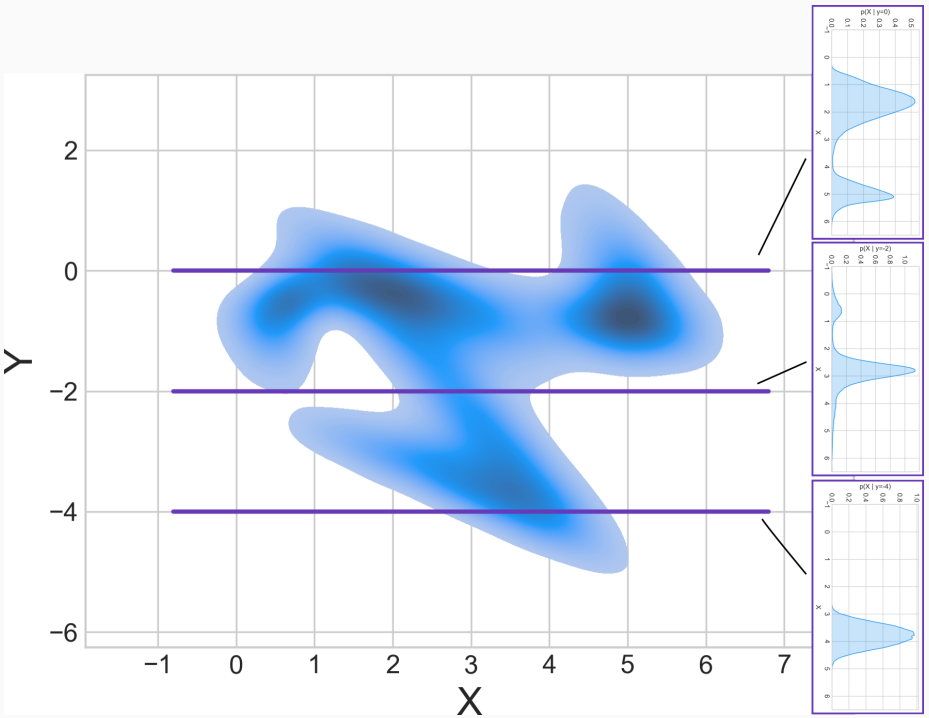
which is just the **expectation** of Y given x

No function has lower squared loss.



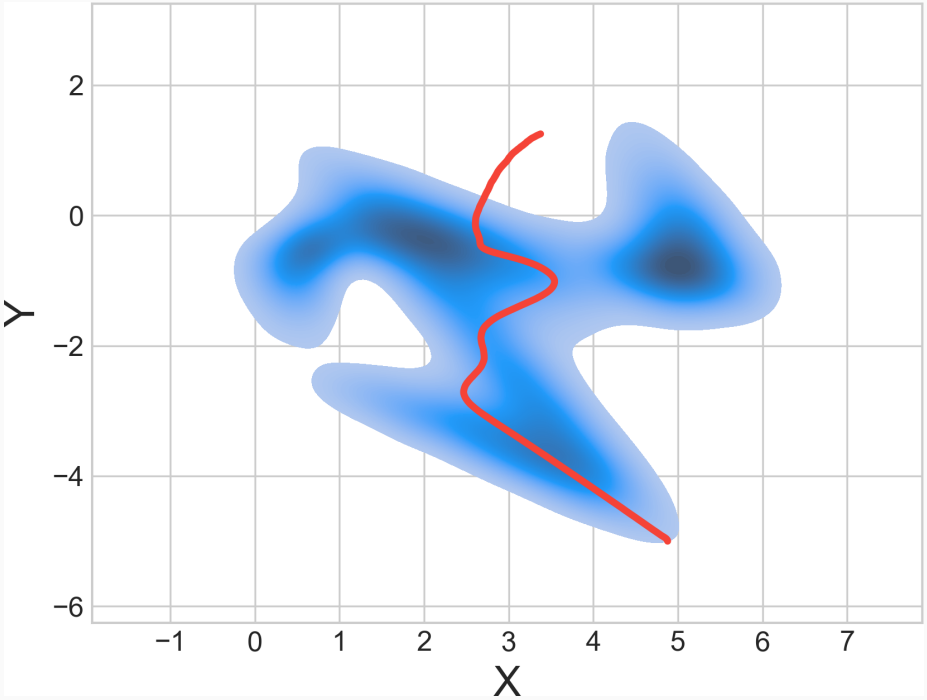
Of course, that Y is output and X is input was arbitrary, so the whole "trick" works also in the other direction:

$$p^*(x \mid y) = \frac{p^*(y,x)}{\underbrace{p^*(y)}_{=\int p^*(x,y)\,\mathrm{d}x}}$$



Of course, that Y is output and X is input was arbitrary, so the whole "trick" works also in the other direction:

$$f(y) := \mathbb{E}[X \mid y]$$



What is probabilistic reasoning?

Knowledge Base, Probabilistic Model

- **joint distribution** $p(x_1, \ldots, x_D)$

Inference

- **marginals** ("Ignore", "Account for Unknowns")

$$p(\boldsymbol{y}) = \int_{\boldsymbol{\mathcal{Z}}} p(\boldsymbol{y}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z}$$

- **conditionals** ("Observe", "Inject Information")

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \frac{p(\boldsymbol{y}, \boldsymbol{z})}{p(\boldsymbol{z})} = \frac{p(\boldsymbol{y}, \boldsymbol{z})}{\int_{\boldsymbol{\mathcal{Y}}} p(\boldsymbol{y}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{y}}$$

- **sampling** $\boldsymbol{x} \sim p_{\boldsymbol{X}}$
- **compute density** $p(\boldsymbol{x})$
- **expectations** $\mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \int p(\boldsymbol{x}) \, f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$
- **maximization** $\arg\max_{\boldsymbol{x}} p(\boldsymbol{x})$

The Problem

However, probabilistic inference is **hard** 

	GANs	VAEs	DBMs	Flows	ARMs
sampling	✓	✓	✓	✓	✓
density	✗	✗	✓	✓	✓
marginals	✗	✗	✗	✗	✗
condition	✗	✗	✗	✗	✗
moments	✗	✗	✗	✗	✗
max (MAP)	✗	✗	✗	✗	✗
$\mathbb{E}$	✗	✗	✗	✗	✗

The Problem

However, probabilistic inference is **hard**  . . . except for PCs!

	GANs	VAEs	DBMs	Flows	ARMs	PCs
sampling	✓	✓	✓	✓	✓	✓
density	✗	✗	✓	✓	✓	✓
marginals	✗	✗	✗	✗	✗	✓
condition	✗	✗	✗	✗	✗	✓
moments	✗	✗	✗	✗	✗	✓
max (MAP)	✗	✗	✗	✗	✗	✓ (✗)
$\mathbb{E}$	✗	✗	✗	✗	✗	✓ (✗)

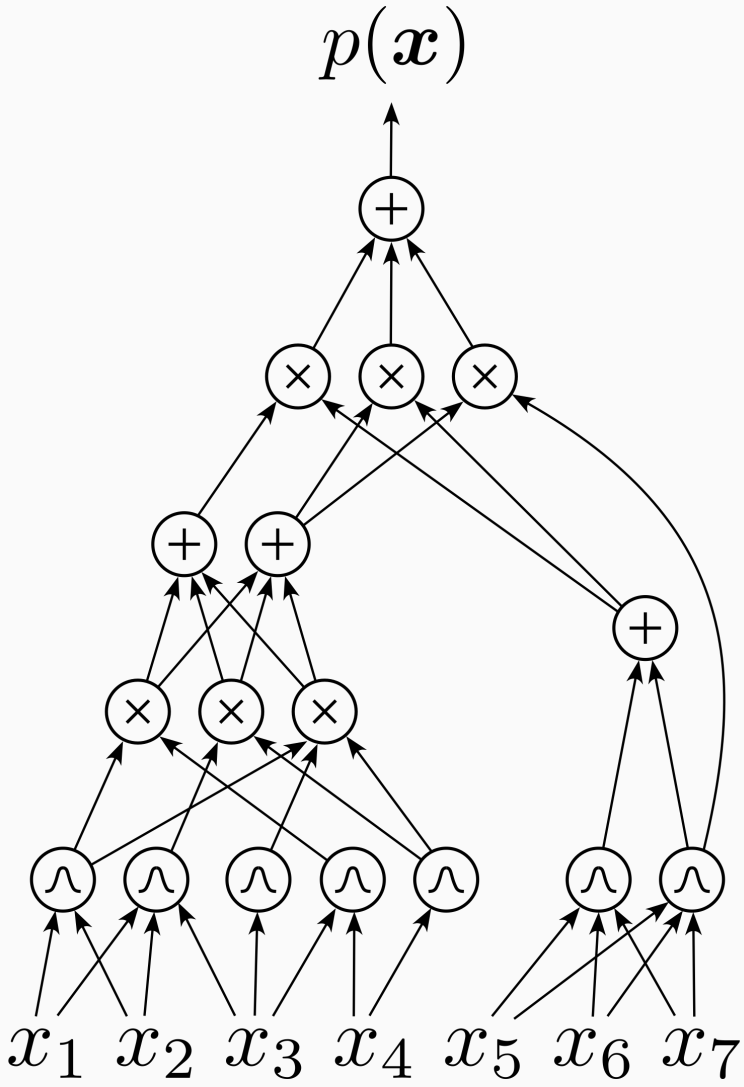
Probabilistic Circuits

A **probabilistic circuit (PC)** is a computational graph (neural network) containing three types of nodes:

- **distributions**
- **products**
- **sums**

Input: values $\boldsymbol{x} = (x_1, \ldots, x_D)$ for RVs $\boldsymbol{X}$ we want to model

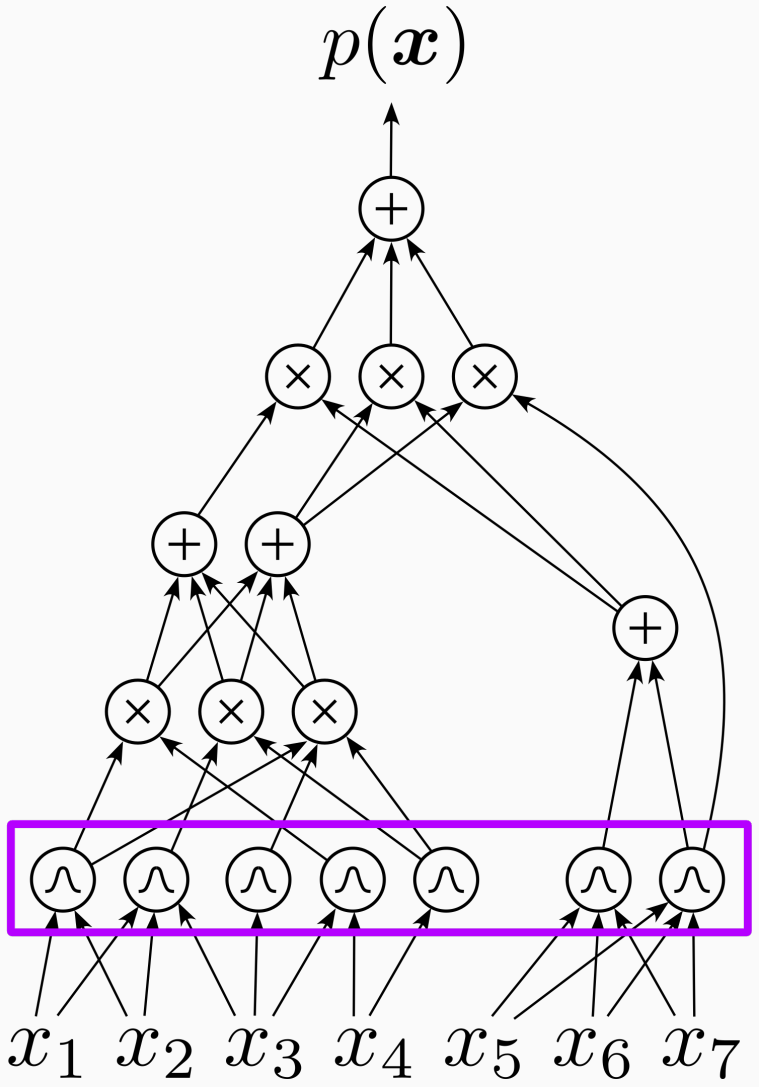
Output: joint density $p(\boldsymbol{x})$ evaluated at $\boldsymbol{x}$ (perhaps unnormalized)



We will explain three types of nodes in the following.

But first note that

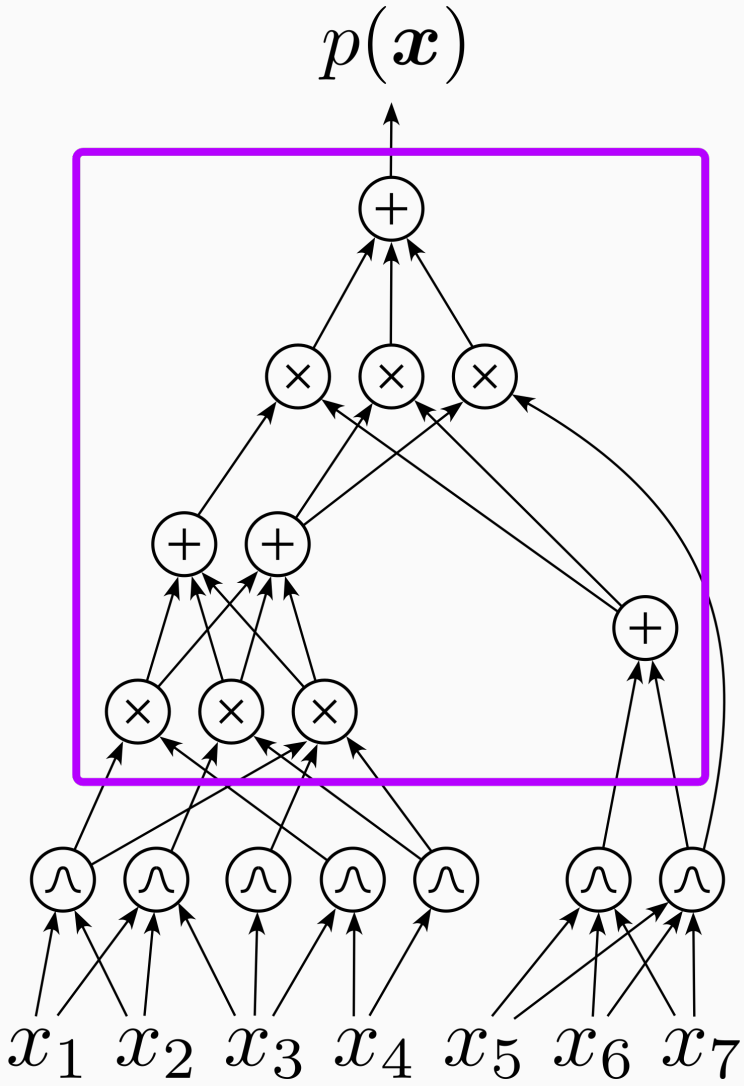
- all nodes on the first layer are distributions 



We will explain three types of nodes in the following.

But first note that

- all nodes on the first layer are distributions
- all nodes on higher layers are sums or products

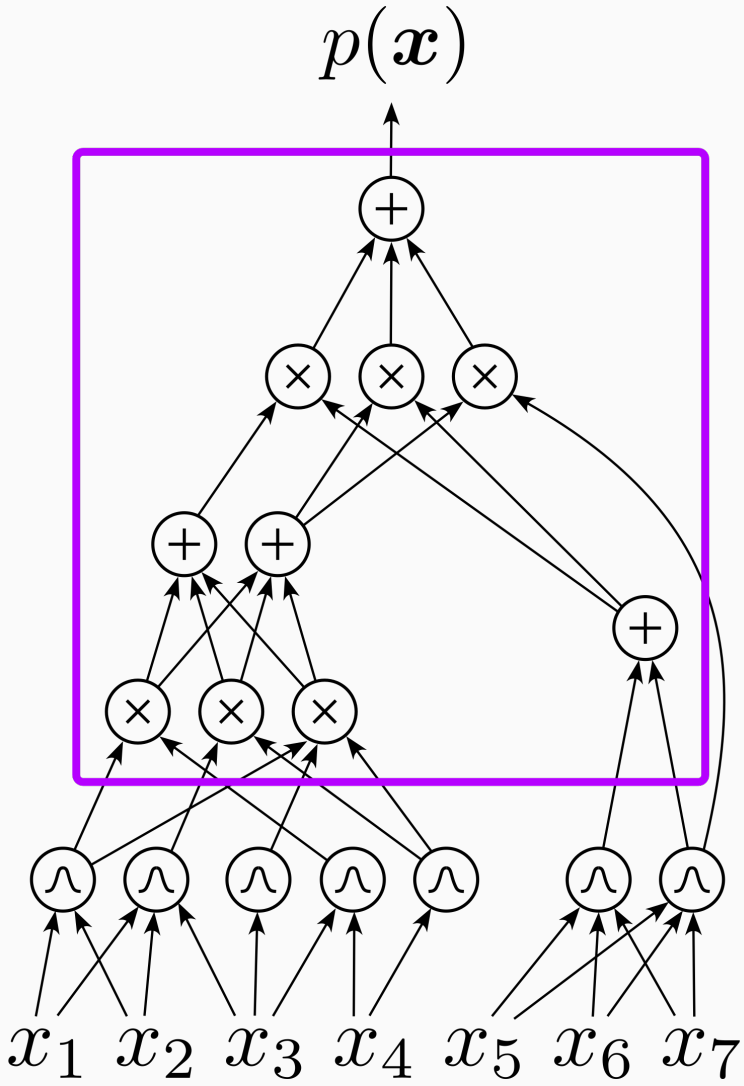


We will explain three types of nodes in the following.

But first note that

- all nodes on the first layer are distributions 
- all nodes on higher layers are sums or products 

This is indeed a requirement.

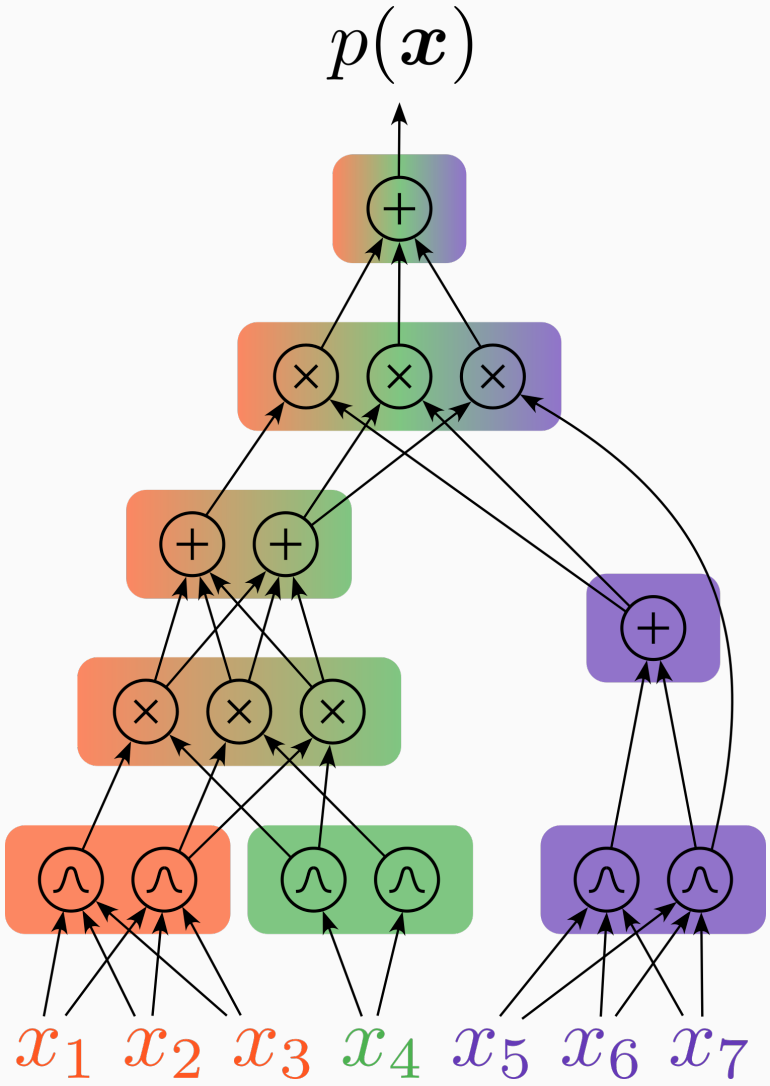


Furthermore, note that **each node depends only on a subset of the inputs**, i.e. each node has a restricted **receptive field** or **scope**.

scope $sc(N) \subseteq \boldsymbol{X}$

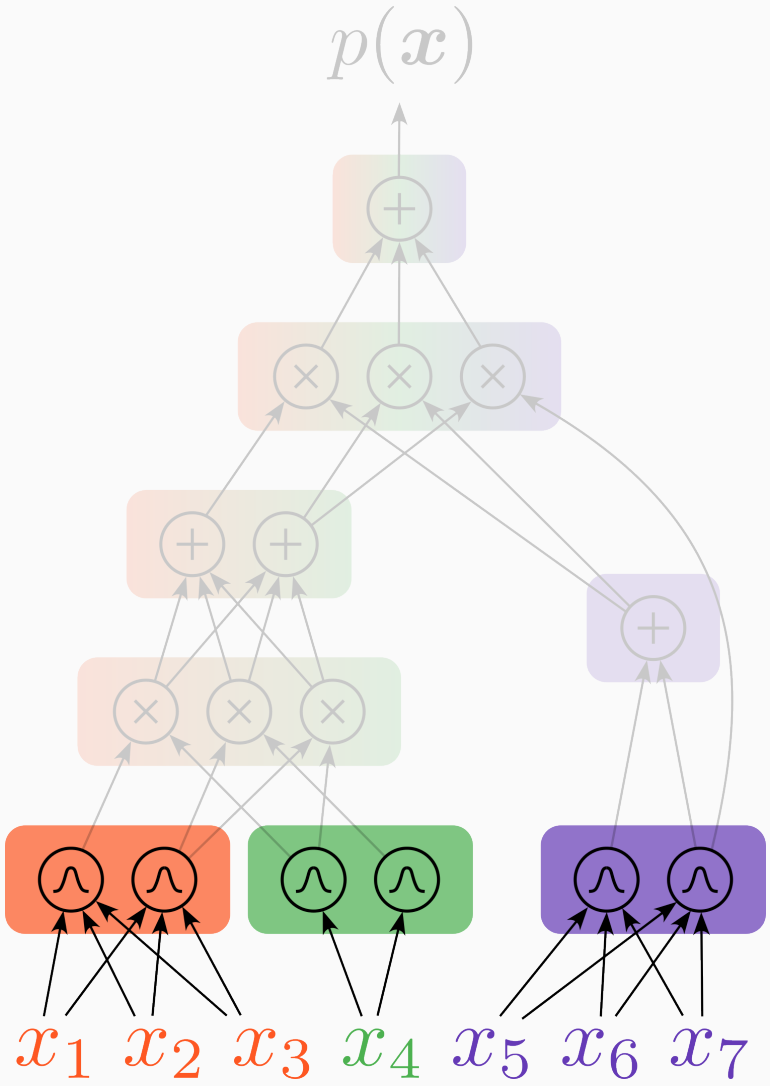
The scope of a node N is defined as the set of RVs it depends on. The **output node** of a PC has always full scope $\boldsymbol{X}$.

The notion of scope will turn out to be very important!



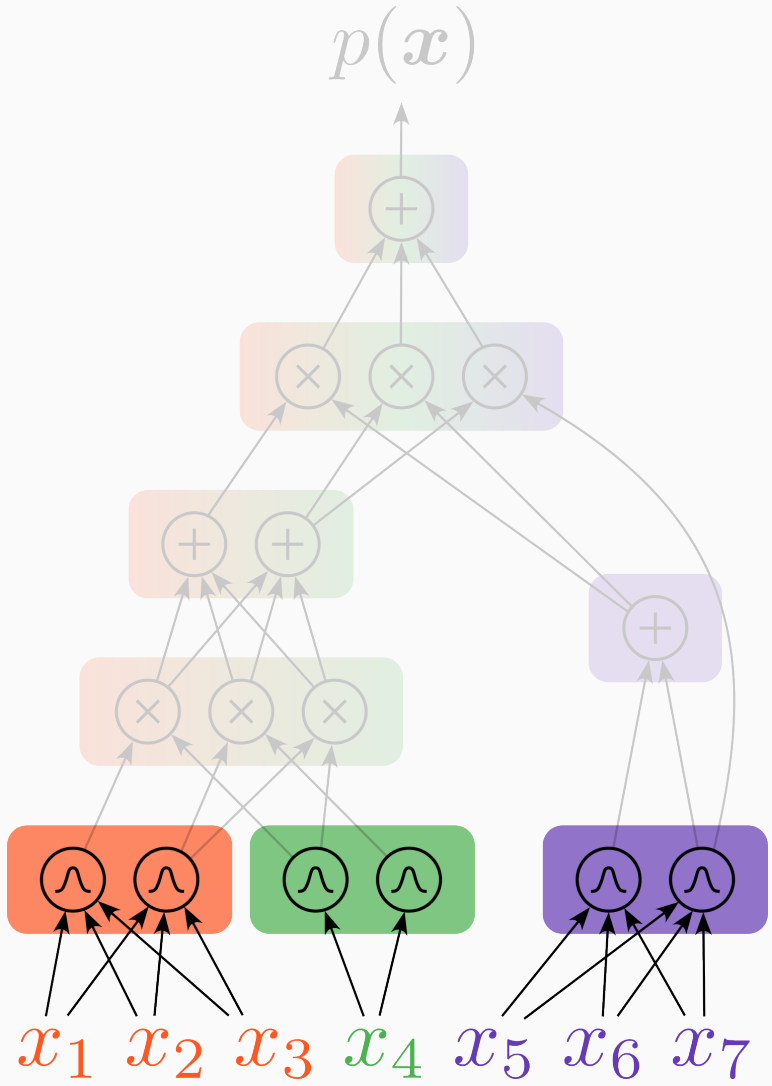
A **distribution (node)**  is just **some** density over its scope. In the **forward pass** it evaluates this density, which serves as input to the next nodes.

Every distribution node typically has its **own parameters**, hence **two distribution nodes over the same scope will represent different distributions**.



On the right, we see

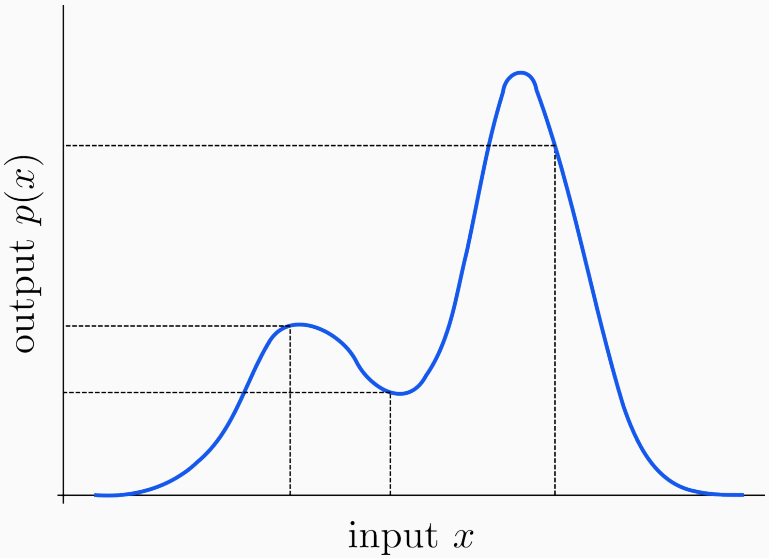
- 2 (joint) distributions over $\{X_1, X_2, X_3\}$ (orange, left)
- 2 distributions over $\{X_4\}$ (green, center)
- 2 (joint) distributions over $\{X_5, X_6, X_7\}$ (purple, right)



Common Pitfall 1

"So, the distribution nodes output samples?" 

No, in the forward pass they output a density! For example, if a distribution node over $\{X\}$ represents the following density, it will really just evaluate the density at any provided x and return $p(x)$:



Protip: Think about densities like non-linearities in neural nets.

Common Pitfall 2

"Oh, the symbol  you use for distribution shows a Gaussian, so all these distribution nodes are Gaussian!" 

No, it's just a symbol! Distribution nodes can be **any** distribution, including

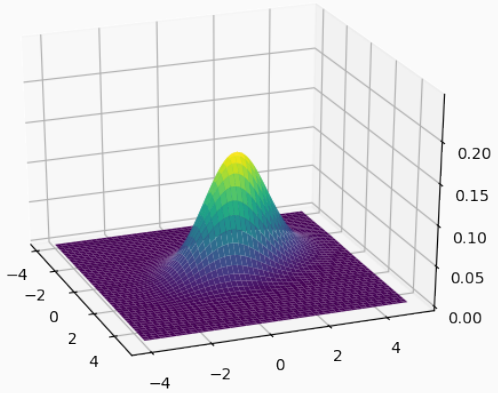
- probability densities and mass functions (can model both continuous **and** discrete RVs)
- whole other generative models (e.g. VAEs, Gaussian processes)
- distributions without density (probability measures)

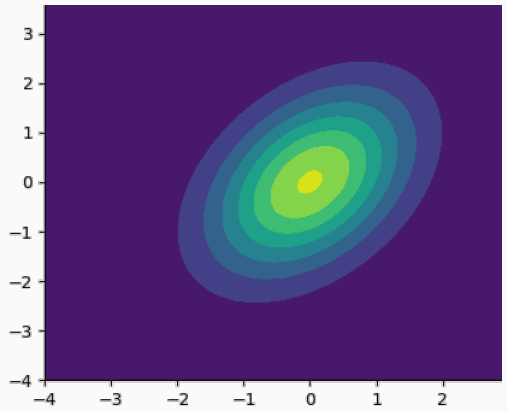
However, Gaussians are indeed often used as distribution nodes.

Gaussian Distribution Nodes

The promise of PCs is **tractable inference**, especially marginals and conditionals. To this end, distribution nodes need to be tractable themselves. **Gaussians are a great example of this.**

$$p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \, |\boldsymbol{\Sigma}|}} \, e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$





Gaussian Marginals

Marginals of Gaussians are again Gaussian, where parameters of the marginalized RVs are simply discarded.

For a Gaussian over 5 random variables X_1, X_2, X_3, X_4, X_5, the marginal over X_1, X_4, X_5 (**query** RVs) is Gaussian with parameters $\boldsymbol{\mu}_q$ and Σ_{qq} obtained by deleting rows and columns of X_2, X_3:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \cancel{\mu_2} \\ \cancel{\mu_3} \\ \mu_4 \\ \mu_5 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} v_{1,1} & \cancel{v_{1,2}} & \cancel{v_{1,3}} & v_{1,4} & v_{1,5} \\ \cancel{v_{2,1}} & \cancel{v_{2,2}} & \cancel{v_{2,3}} & \cancel{v_{2,4}} & \cancel{v_{2,5}} \\ \cancel{v_{3,1}} & \cancel{v_{3,2}} & \cancel{v_{3,3}} & \cancel{v_{3,4}} & \cancel{v_{3,5}} \\ v_{4,1} & v_{4,2} & v_{4,3} & v_{4,4} & v_{4,5} \\ v_{5,1} & v_{5,2} & v_{5,3} & v_{5,4} & v_{5,5} \end{pmatrix}$$

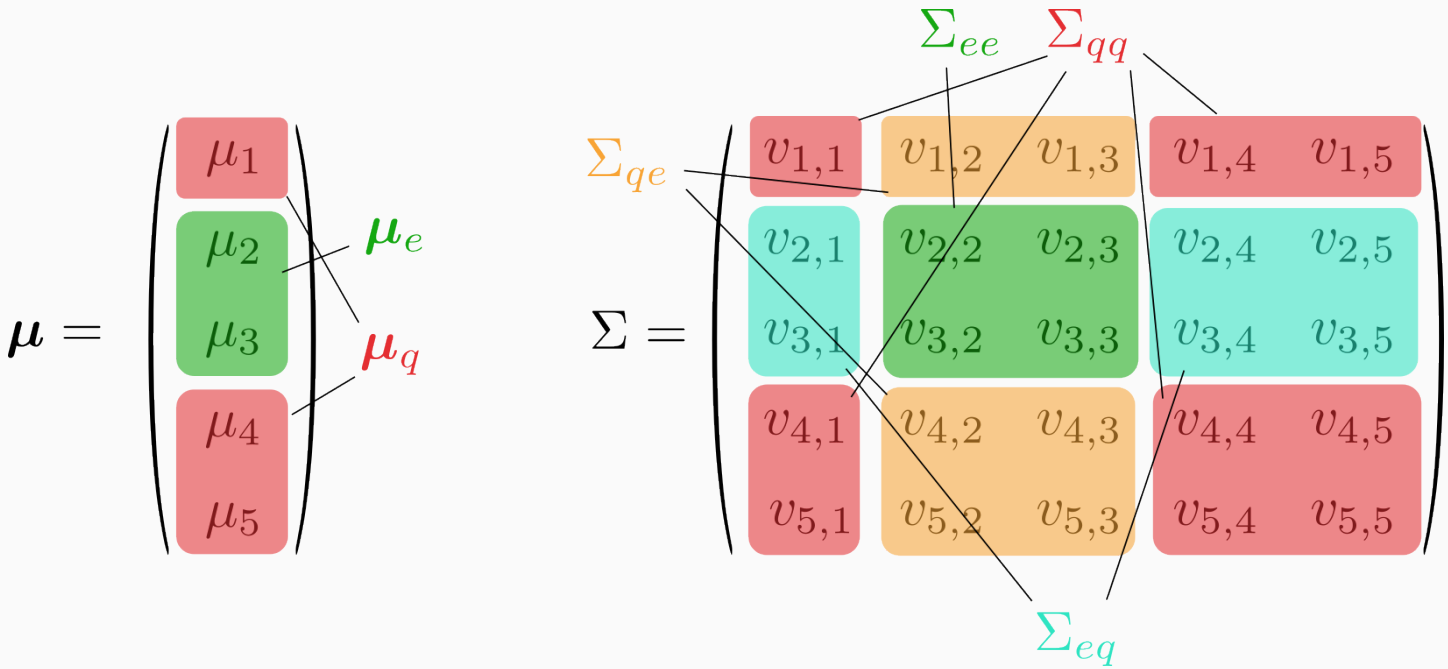
$$\boldsymbol{\mu}_q = \begin{pmatrix} \mu_1 \\ \mu_4 \\ \mu_5 \end{pmatrix} \qquad \Sigma_{qq} = \begin{pmatrix} v_{1,1} & v_{1,4} & v_{1,5} \\ v_{4,1} & v_{4,4} & v_{4,5} \\ v_{5,1} & v_{5,4} & v_{5,5} \end{pmatrix}$$

Gaussian Conditional Distributions

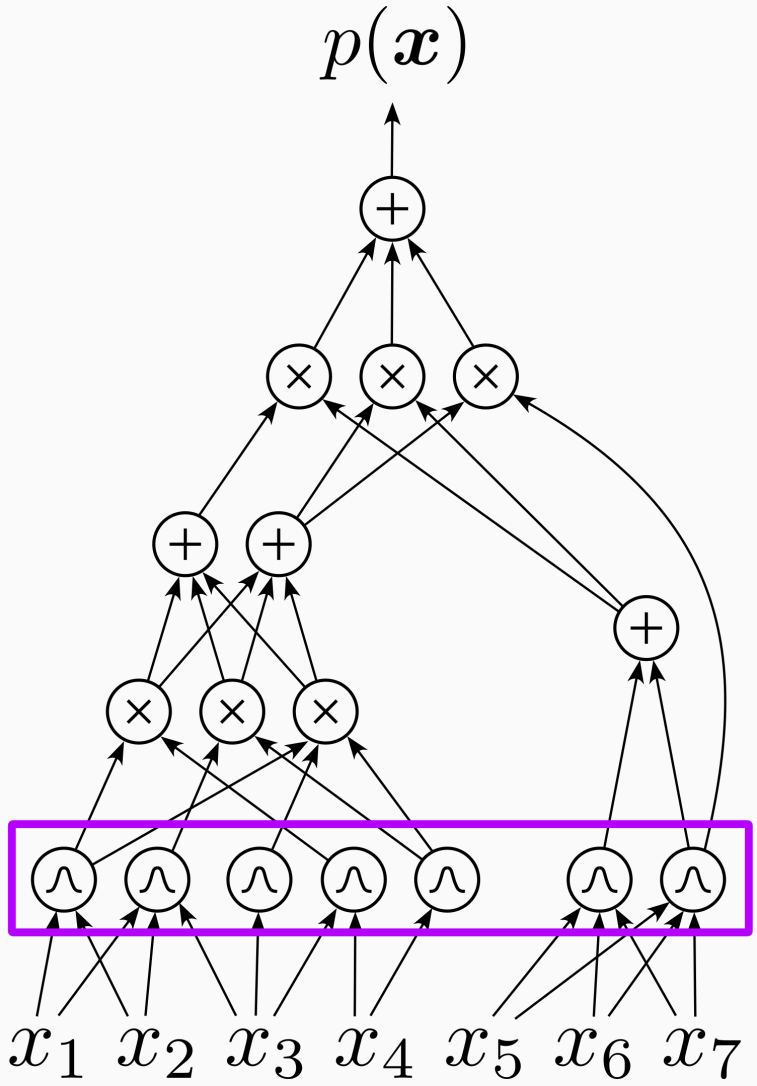
In Gaussians with parameters $\boldsymbol{\mu}$ and Σ, any conditional distribution $p(\boldsymbol{x}_q \mid \boldsymbol{x}_e)$ is again Gaussian with **closed form parameters**:

$$\boldsymbol{\mu}_{q|e} = \boldsymbol{\mu}_q + \Sigma_{qe}\Sigma_{ee}^{-1}(\boldsymbol{x}_e - \boldsymbol{\mu}_e)$$
$$\Sigma_{q|e} = \Sigma_{qq} - \Sigma_{qe}\Sigma_{ee}^{-1}\Sigma_{eq}$$

For **query** RVs X_1, X_4, X_5 and **evidence** RVs X_2, X_3:



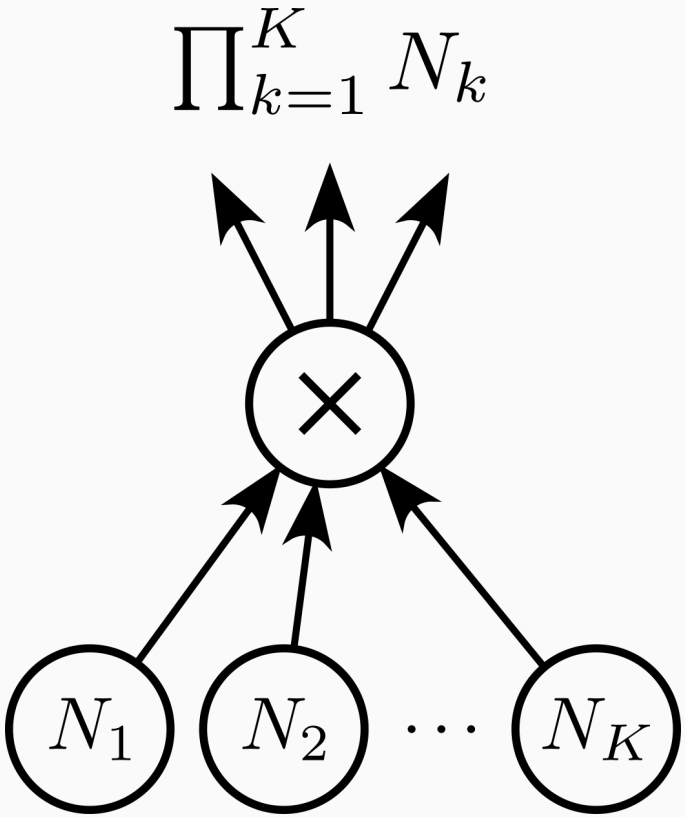
- **distribution nodes** are in the first layer of PCs (hence, sometimes called **input distributions** or **leaves**)
- they represent a density over their **scope**
- each distribution node has its own parameters
- should allow **tractable inference** (**marginalization**, **conditioning**)
- Gaussians are common, but any density can be used



- a **product node** P simply computes the product of its inputs
- if P is a product node with inputs $N_1, N_2, \ldots, N_K$, it computes

$$P := \prod_{k=1}^{K} N_k$$

- easy!

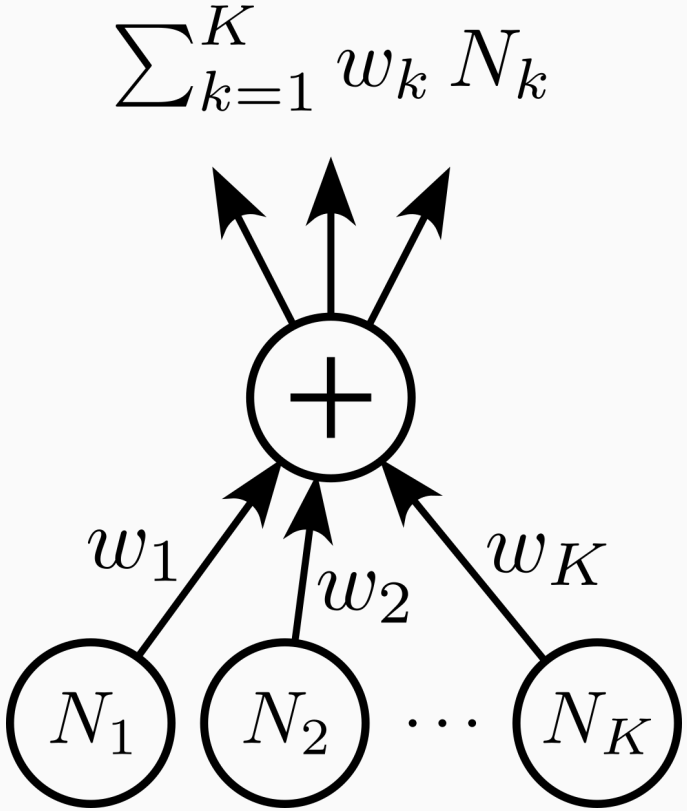


- a **sum node** S computes a weighted sum of its inputs (**linear unit**)
- if S is a sum node with inputs $N_1, N_2, \ldots, N_K$, it computes

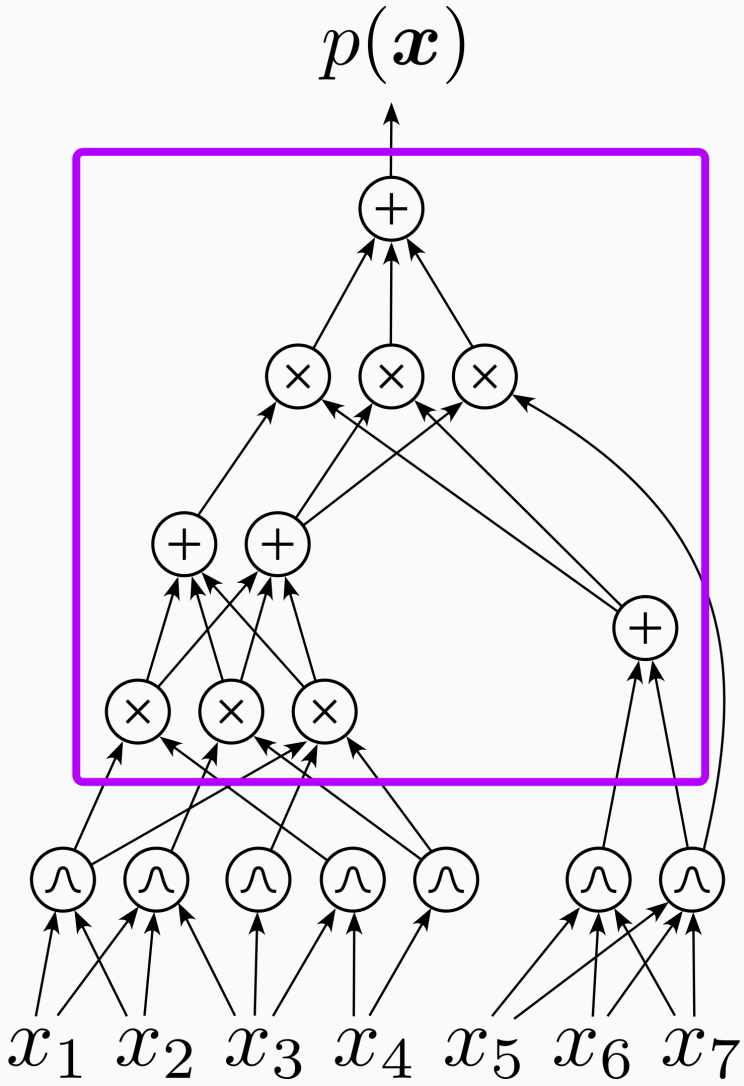
$$S := \sum_{k=1}^{K} w_k \, N_k$$

- the **weights** w_k are parameters of the PC and need to satisfy
  - $w_k \geq 0$
  - $\sum_k w_k = 1$

  meaning S computes a **convex combination** of inputs



- not much surprise here!
- **product nodes** compute products of their inputs
- **sum nodes** compute weighted sums of their intputs
- sum weights are parameters
- they should be non-negative and sum to one (**convex combination**)
- when drawing PCs, weights are typically omitted

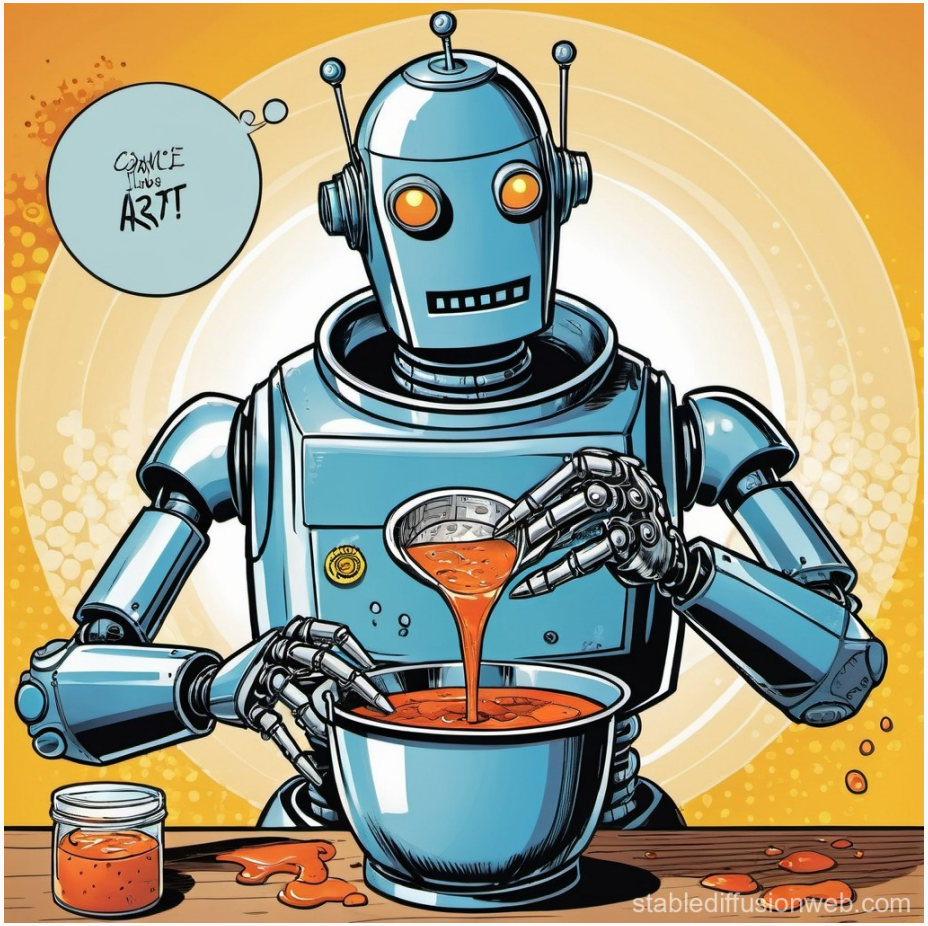


Tractability?

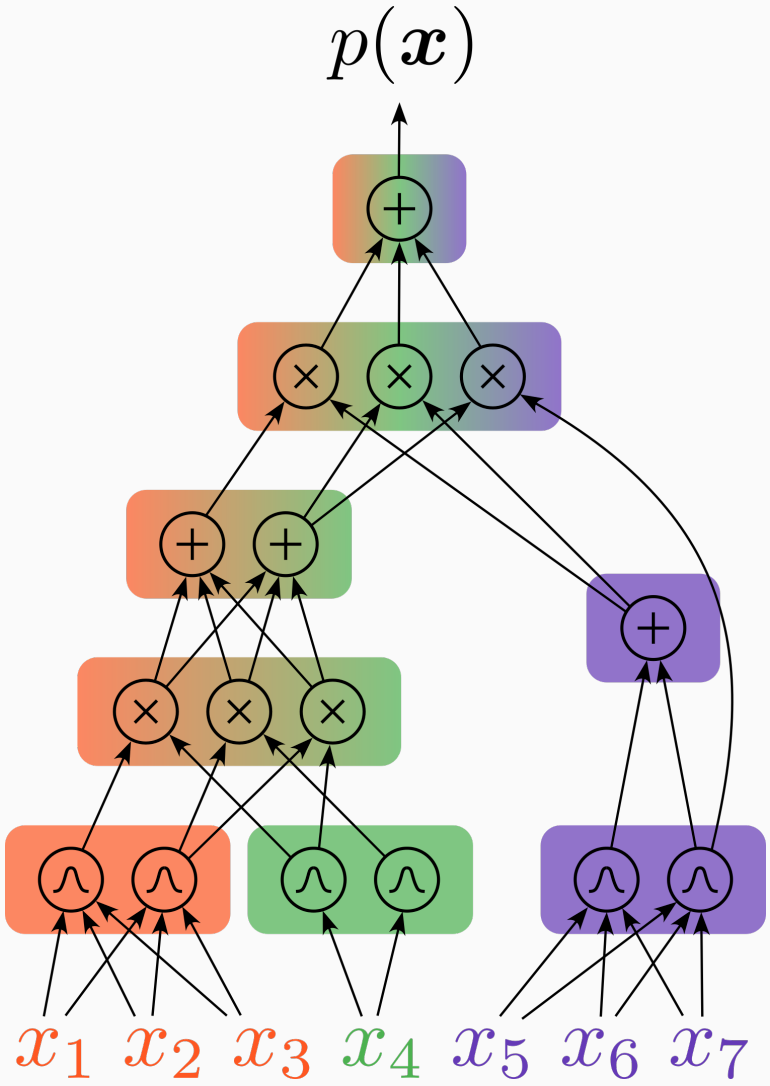
PCs, as discussed so far, do define some (potentially unnormalized) distribution $p(\boldsymbol{X})$.

However, they do not allow to compute marginals or conditionals.

For tractability, we are still missing a secret sauce—**constraints**.



Recall that every node in a PC has a scope (receptive field), which is the subset of $\boldsymbol{X}$ it "sees".



Smoothness

A **sum node** is **smooth**, if all its inputs have the **same scope**. A PC is smooth, if all sum nodes in it are smooth.

Decomposability

A **product node** is **decomposable**, if all its inputs have **disjoint scope**. A PC is decomposable, if all product nodes in it are decomposable.

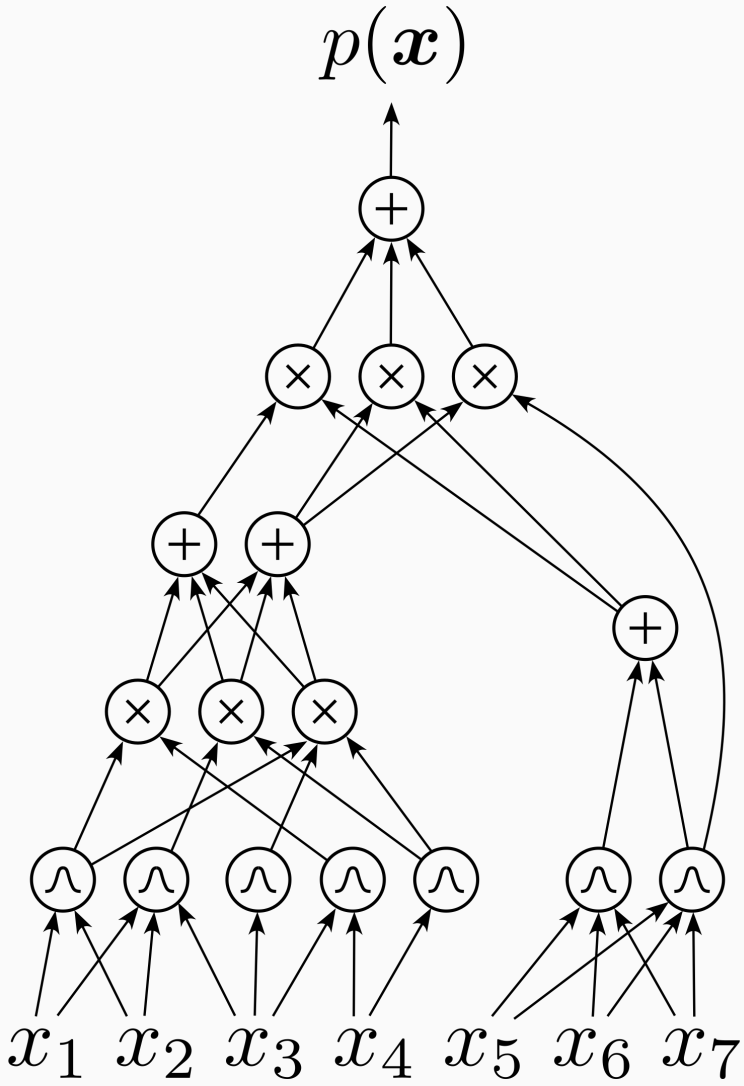
Test Time

Is this PC smooth?

(for each **sum**, all inputs have **same** scope)

Is it decomposable?

(for each **product**, all inputs have **disjoint** scope)



Test Time

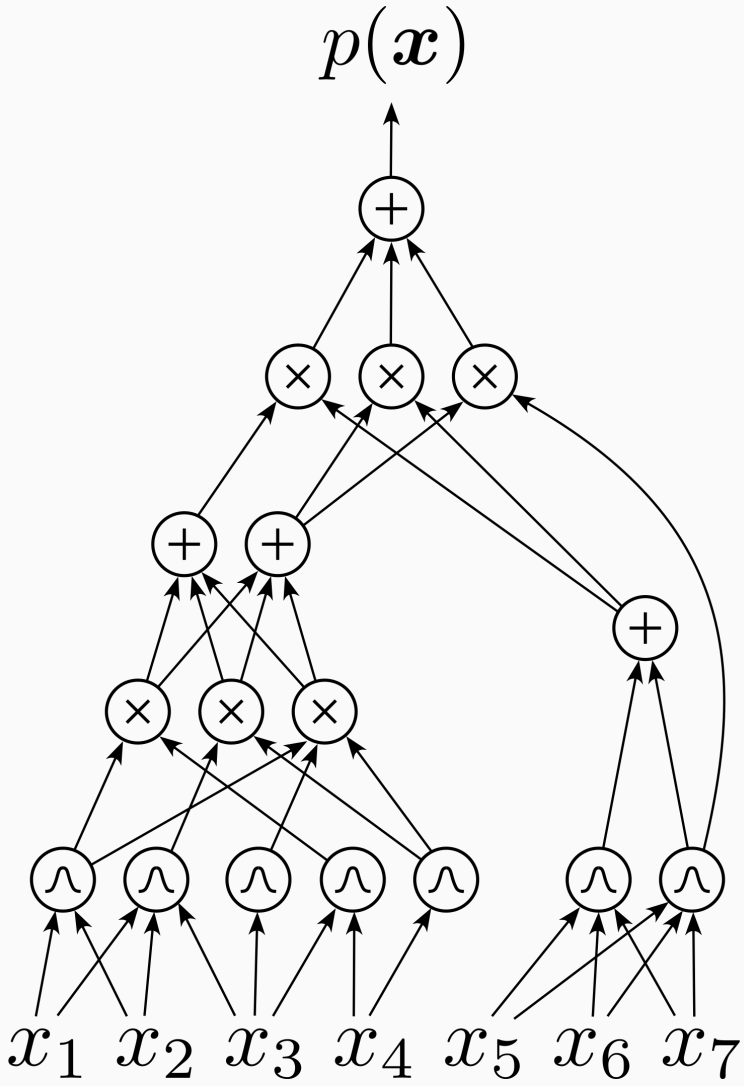
Is this PC smooth?

(for each **sum**, all inputs have **same** scope)

Is it decomposable?

(for each **product**, all inputs have **disjoint** scope)

Yes, it is smooth and decomposable



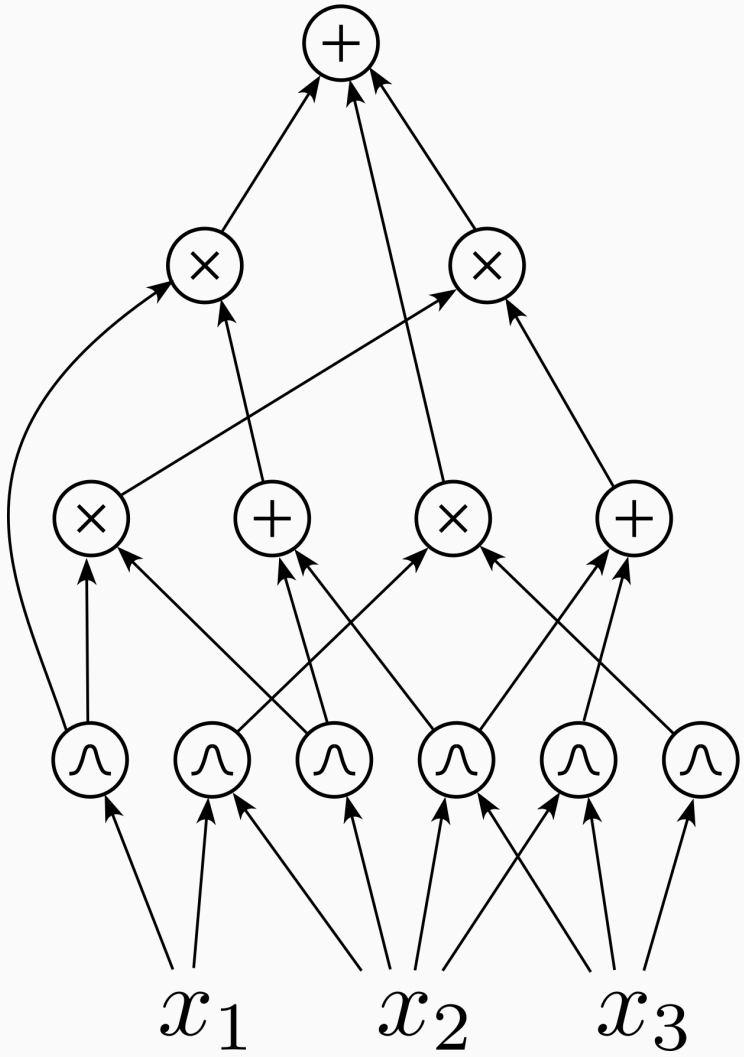
Test Time

Is this PC smooth?

(for each **sum**, all inputs have **same** scope)

Is it decomposable?

(for each **product**, all inputs have **disjoint** scope)



Test Time

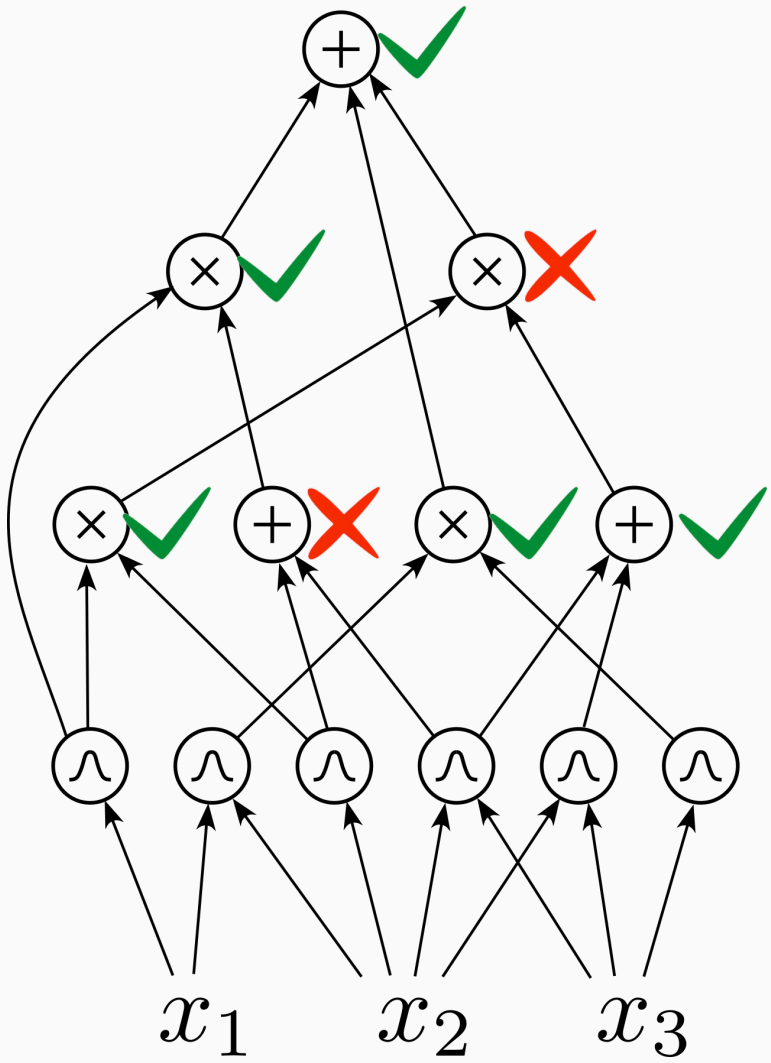
Is this PC smooth?

(for each **sum**, all inputs have **same** scope)

Is it decomposable?

(for each **product**, all inputs have **disjoint** scope)

Neither. There is 1 non-smooth sum and 1 non-decomposable product



What do these constraints mean?

A **product node** P computes the product of its inputs

$$P := \prod_{k=1}^{K} N_k$$

When the N_k's are distributions and the product is **decomposable**, this just means that the product node is a

factorized distribution

which represents **independence** between the N_k's.

A **sum node** S computes a convex combination of its inputs

$$S := \sum_{k=1}^{K} w_k \, N_k$$

where $w_k \geq 0$, $\sum_k w_k = 1$. When the N_k's are distributions and the sum is smooth, this just means that the sum node is a

mixture distribution

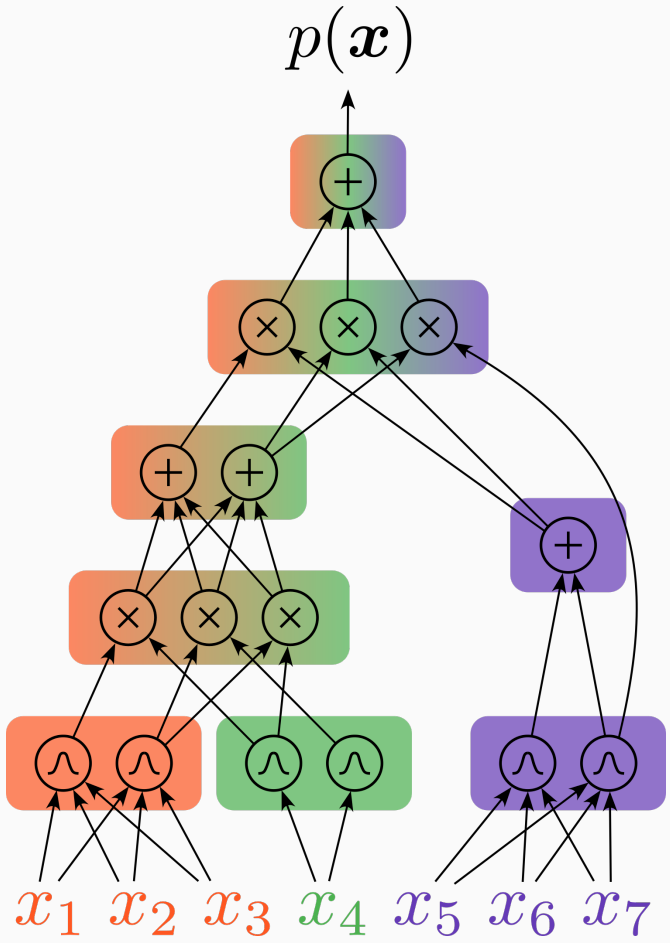
of the distributions represented by the N_k's.

Corollary: All nodes are normalized distributions

Any node in a smooth and decomposable PC is a properly normalized distribution over its scope.

Proof by induction:

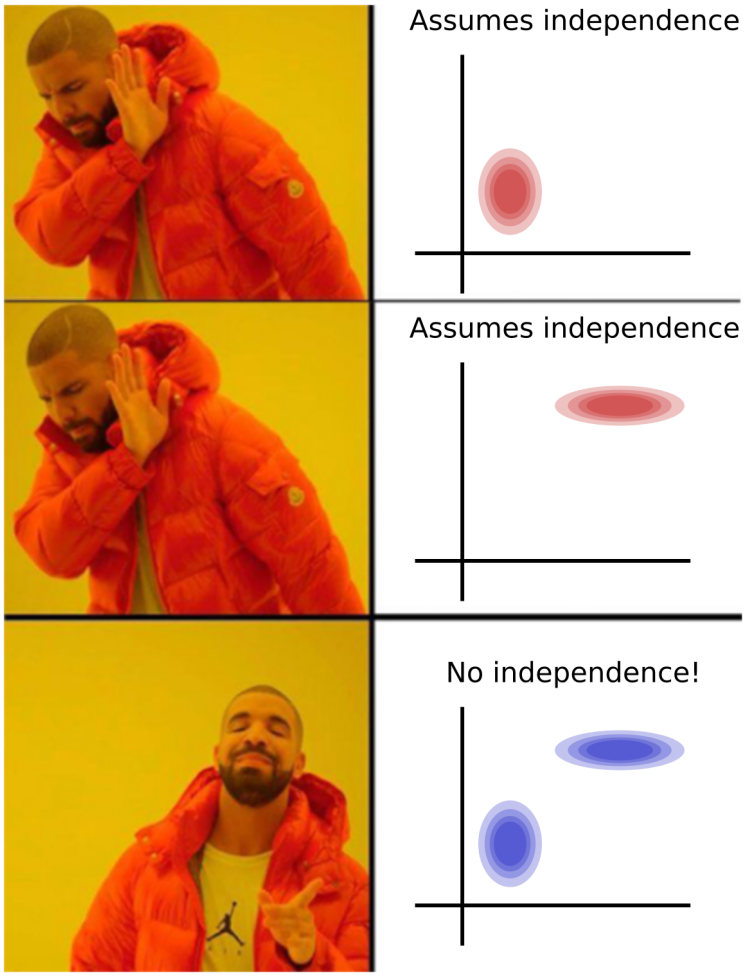
- Distribution nodes (induction basis): by definition
- Products (induction step): the decomposable product of normalized distributions is again a normalized distribution.
- Sums (induction step): a mixture of normalized distributions is again a normalized distribution.



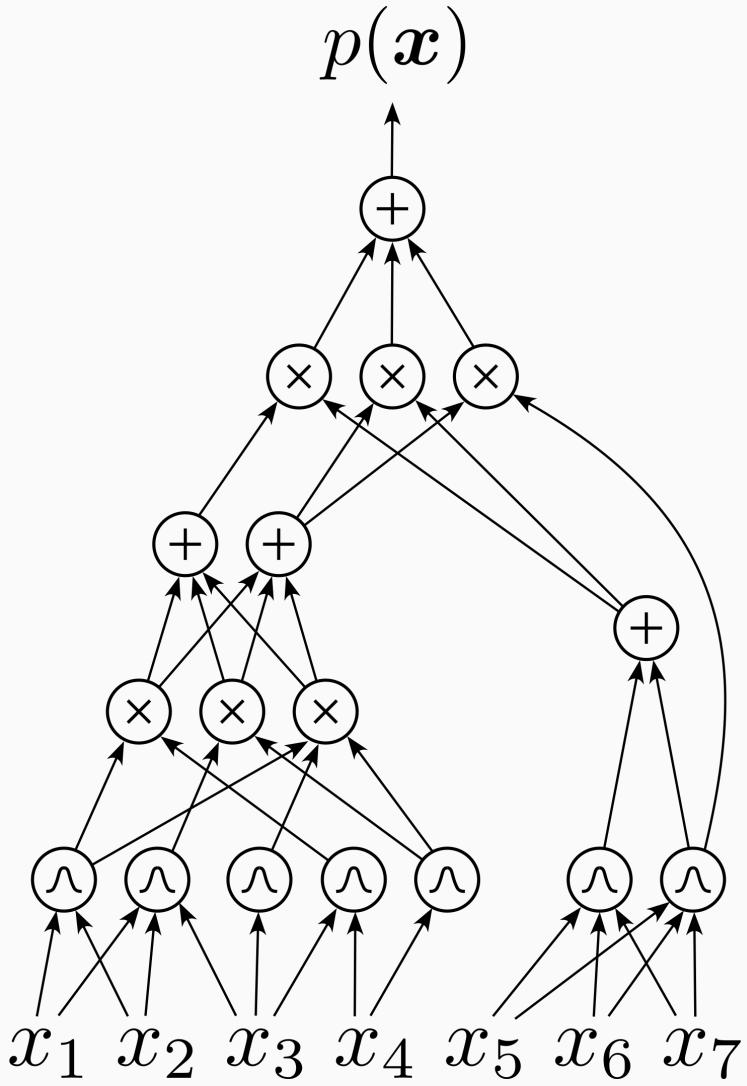
Common Pitfall 3

"The product nodes assume independence among their input distributions! Isn't this a strong assumption?" 🤔

No, not really, since the product nodes are usually input to sum nodes above them. **A mixture of factorized distributions does not factorize!** That is, the mixture do not assume independence postulated by the product nodes.



- computational graph containing
  - **distribution nodes**
  - **product nodes**
  - **sum nodes**
- **smoothness**: inputs of sum nodes have all same scope
  mixture of distributions
- **decomposability**: inputs of product nodes have disjoint scopes
  factorization of distributions



Tractable Inference

Smoothness and decomposability lead to a natural interpretation of PCs as **hierarchical mixture models**.

Crucially, however, they unlock tractable inference in PCs, in particular **marginalization** and **conditioning**.

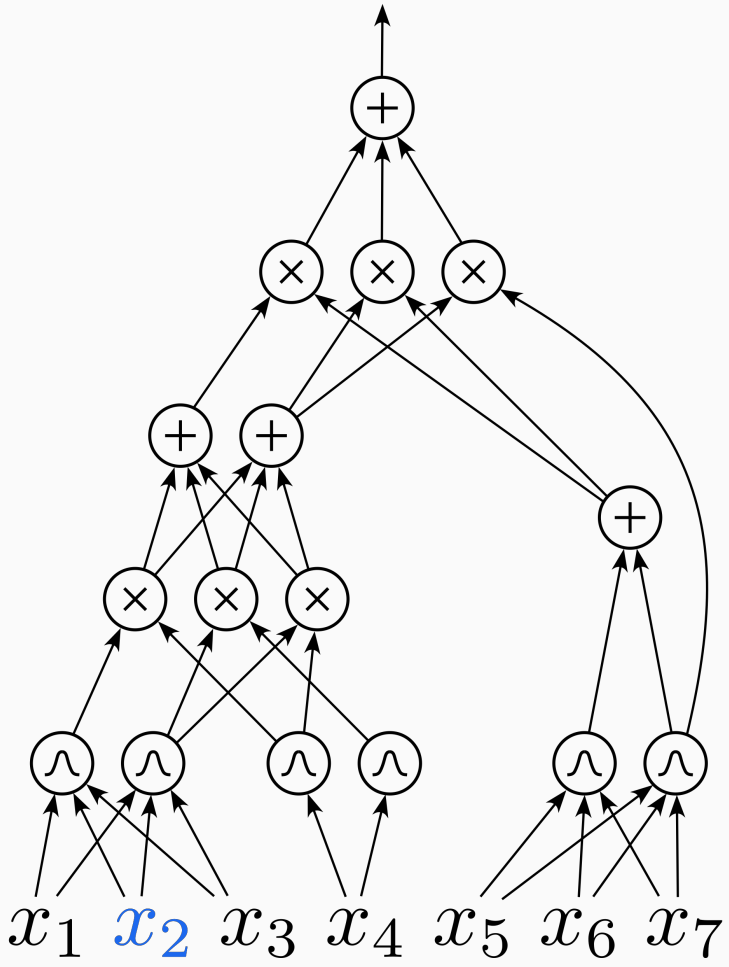
Key requirement: all input distributions allow marginalization and conditioning (e.g. Gaussians). A smooth and decomposable PC essentially inherits this property from its input distributions.

Marginal of PC

Example

- say we want to marginalize X_2

$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$

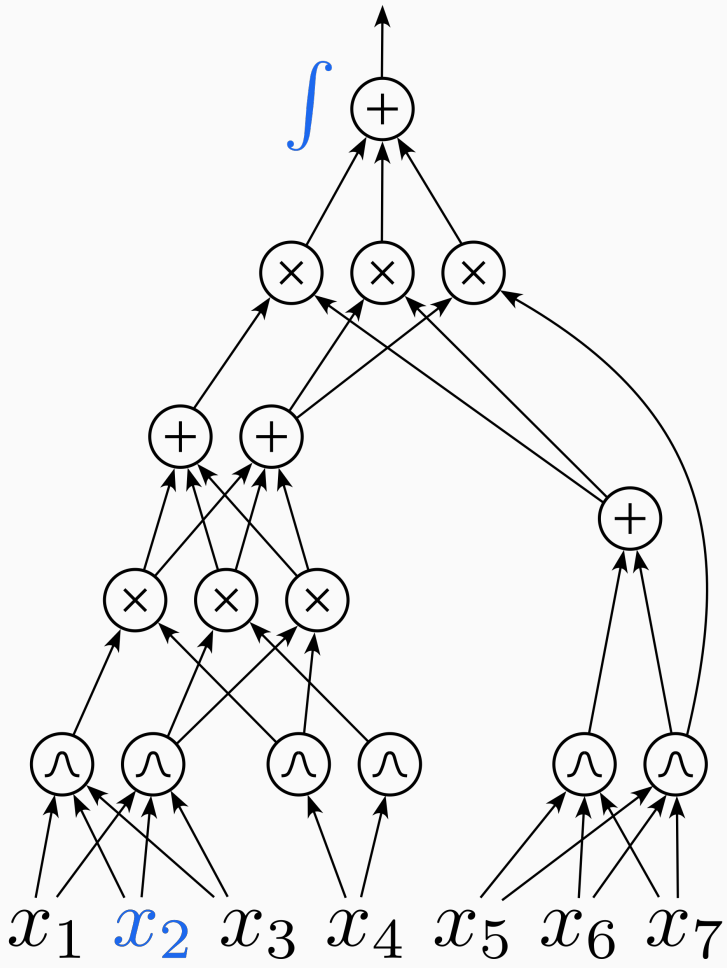


Marginal of PC

Example

- say we want to marginalize X_2
- the integral over x_2 is applied to the output node

$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$



Marginal of PC

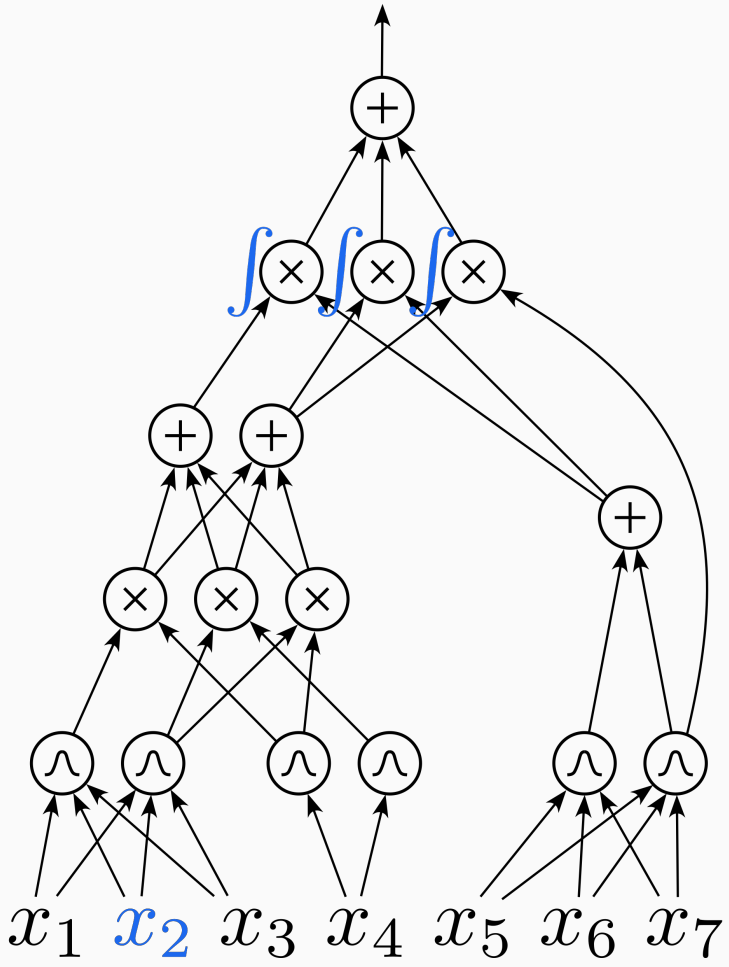
Example

- say we want to marginalize X_2
- the integral over x_2 is applied to the output node
- a sum node! Integrals and sums commute

$$\int \sum_k \dots \mathrm{d}x_2 = \sum_k \int \dots \mathrm{d}x_2$$

- hence we can push the integral to the inputs of the sum node

$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$



Marginal of PC

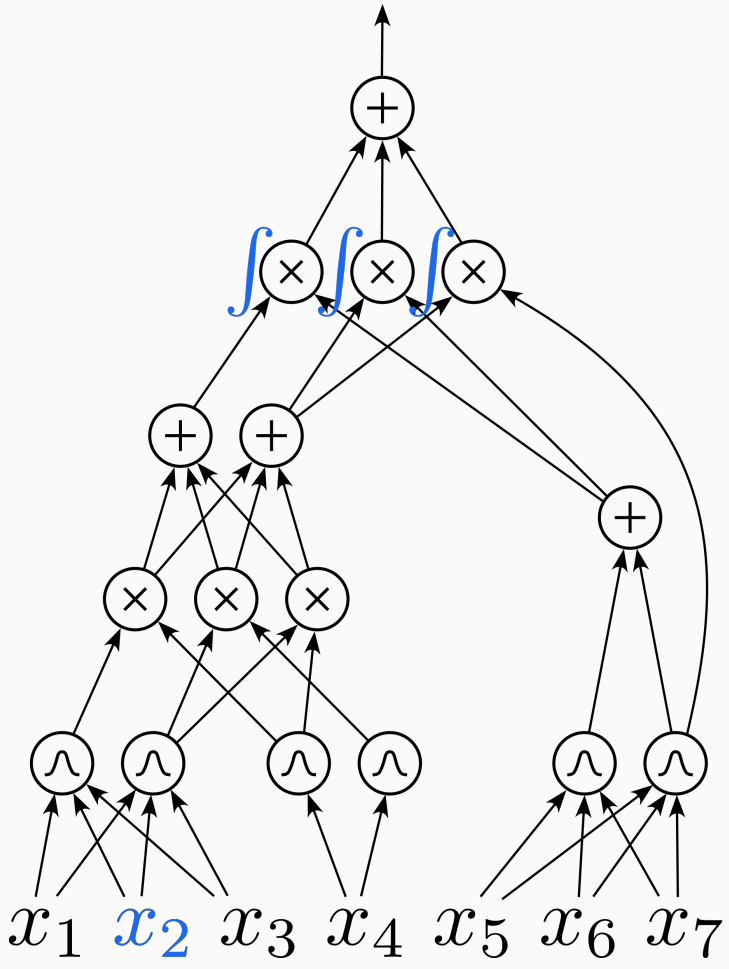
Example

- say we want to marginalize X_2
- the integral over x_2 is applied to the output node
- a sum node! Integrals and sums commute

$$\int \sum_k \dots \mathrm{d}x_2 = \sum_k \int \dots \mathrm{d}x_2$$

- hence we can push the integral to the inputs of the sum node
- now the problem is to compute the integrals of the nodes below (all products in this case)

$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$



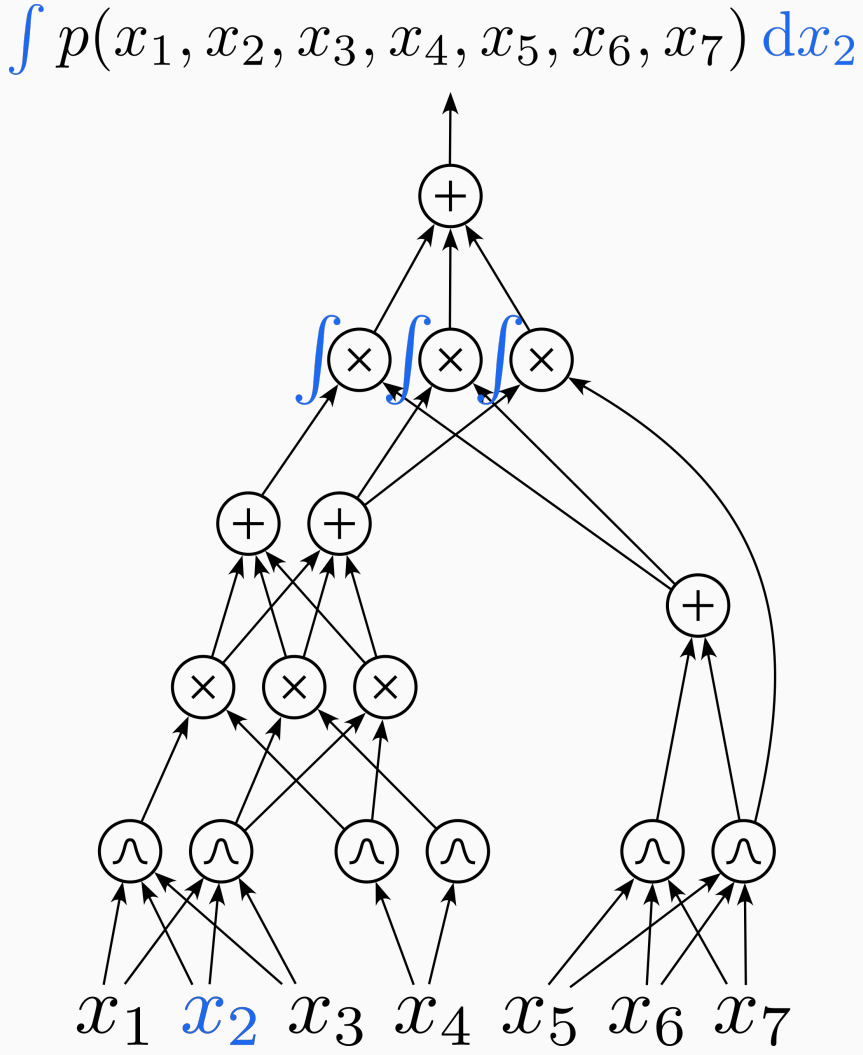
Marginal of PC

Example

- we assume **decomposability**, which means X_2 only appears in one input of each product!
- the other inputs are just a multiplicative constant for the integral over x_2:

$$\int c\, f(x_2, \dots)\, \mathrm{d}x_2 =$$

$$c \int f(x_2, \dots)\, \mathrm{d}x_2$$



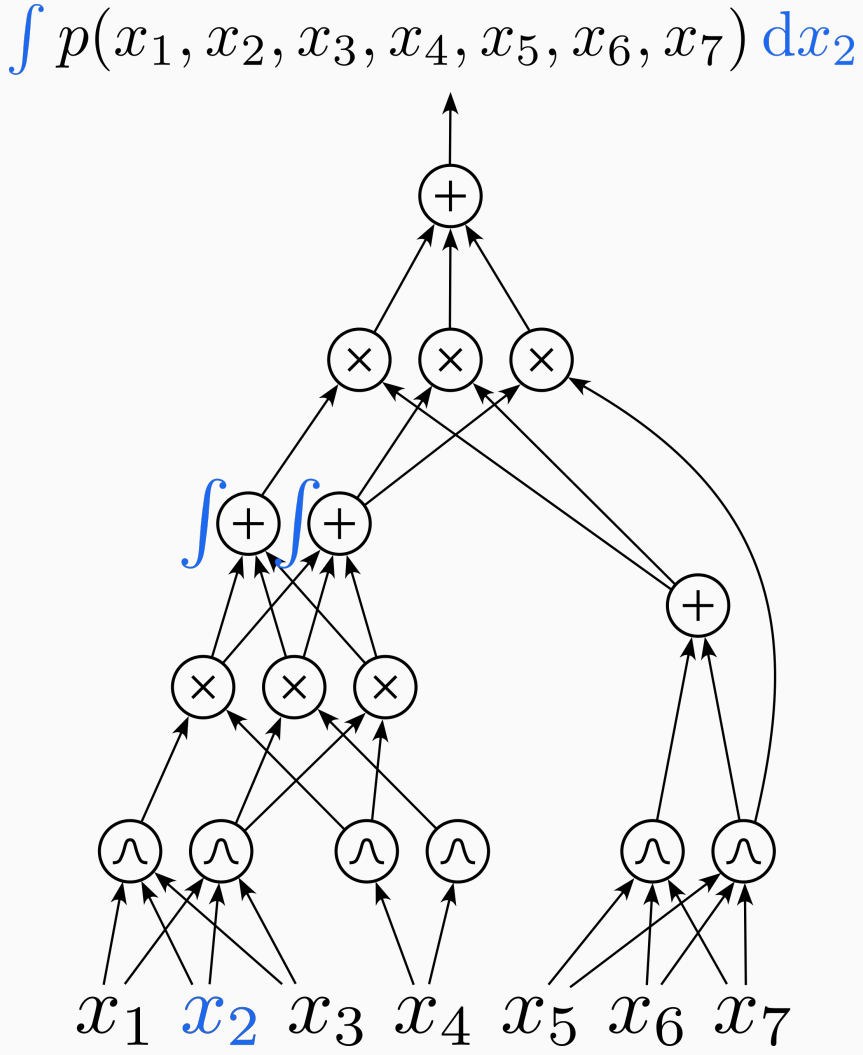
Marginal of PC

Example

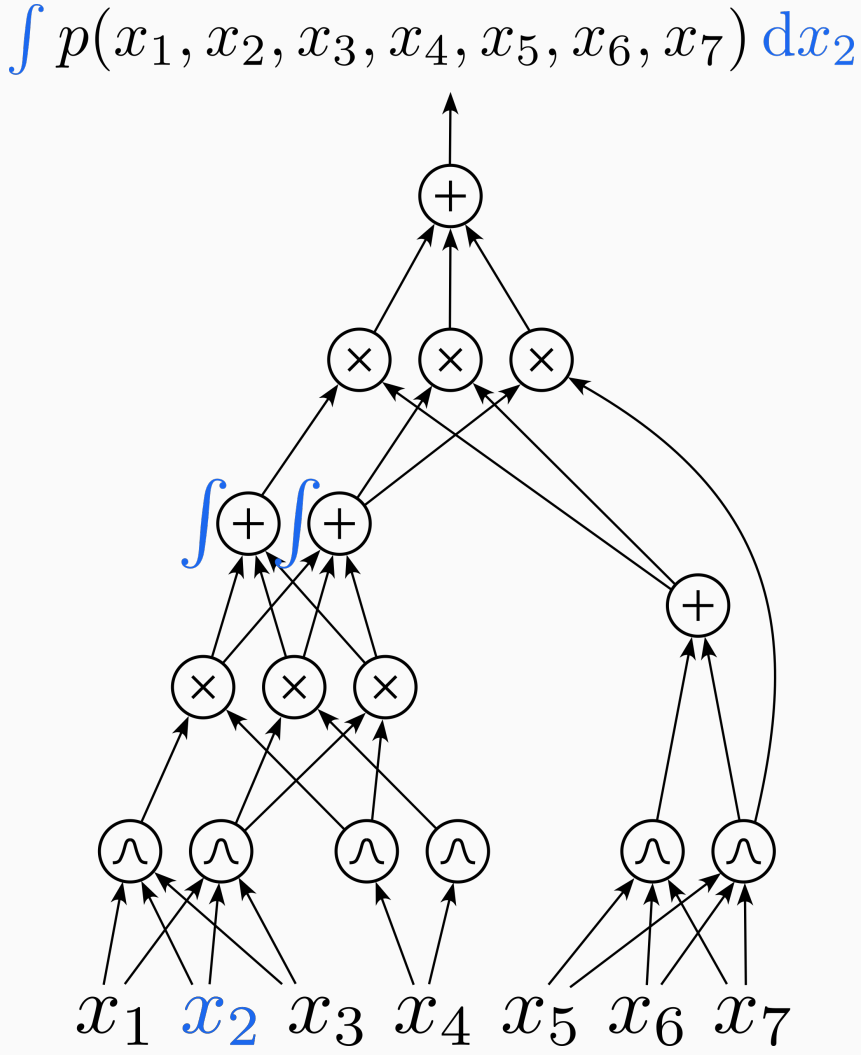
- we assume **decomposability**, which means X_2 only appears in one input of each product!
- the other inputs are just a multiplicative constant for the integral over x_2:

$$\int c\, f(x_2, \dots)\, \mathrm{d}x_2 = c \int f(x_2, \dots)\, \mathrm{d}x_2$$

- hence, we can push the integral over each product, to the respective node containing X_2



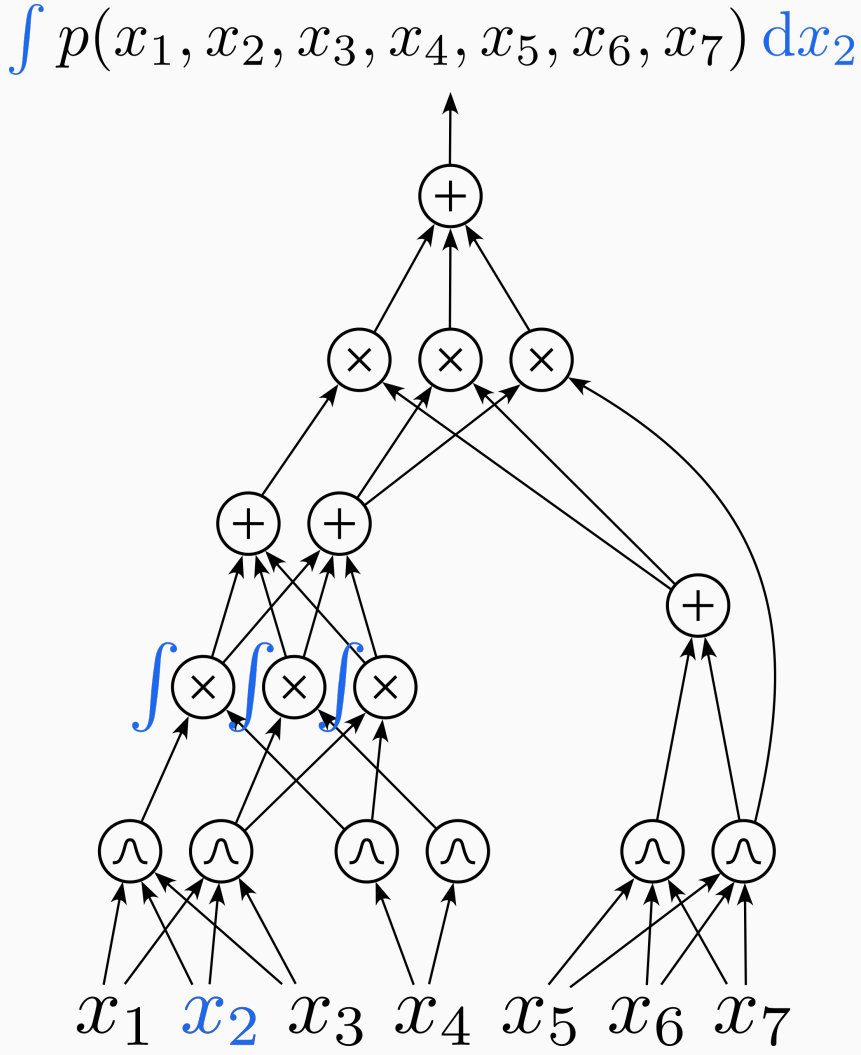
- these are sum nodes again



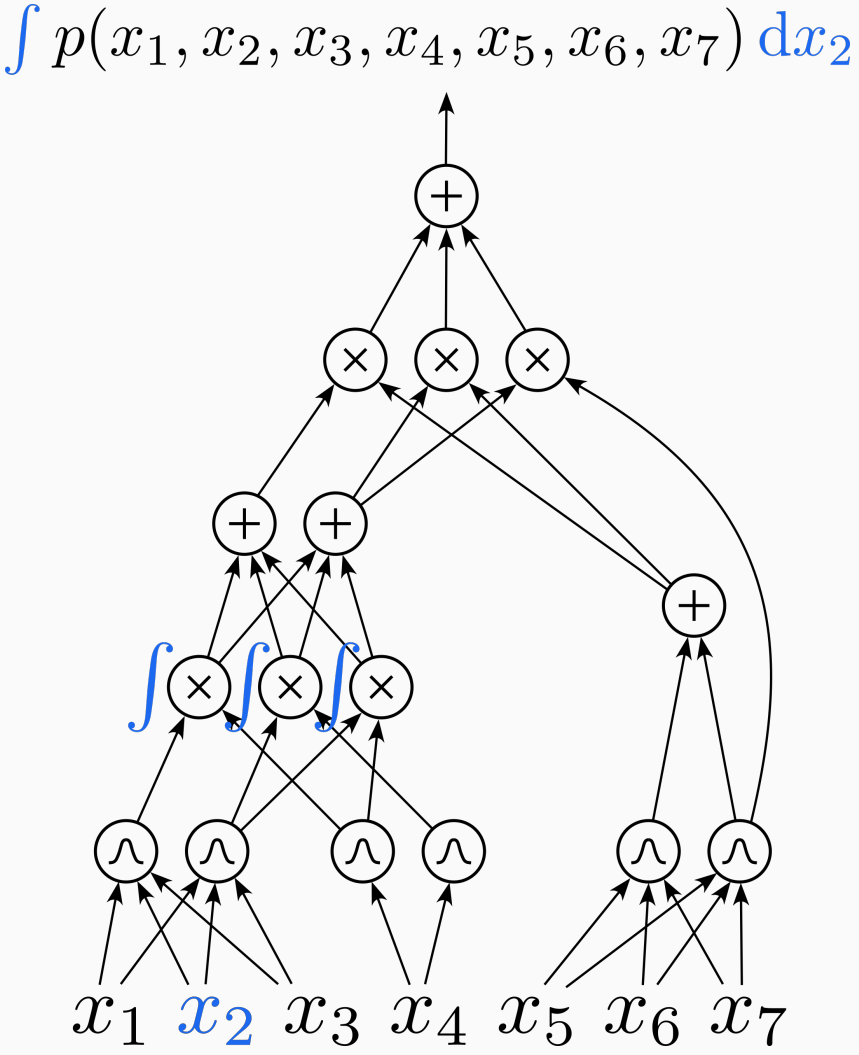
Marginal of PC

Example

- these are sum nodes again
- so we can push them down again



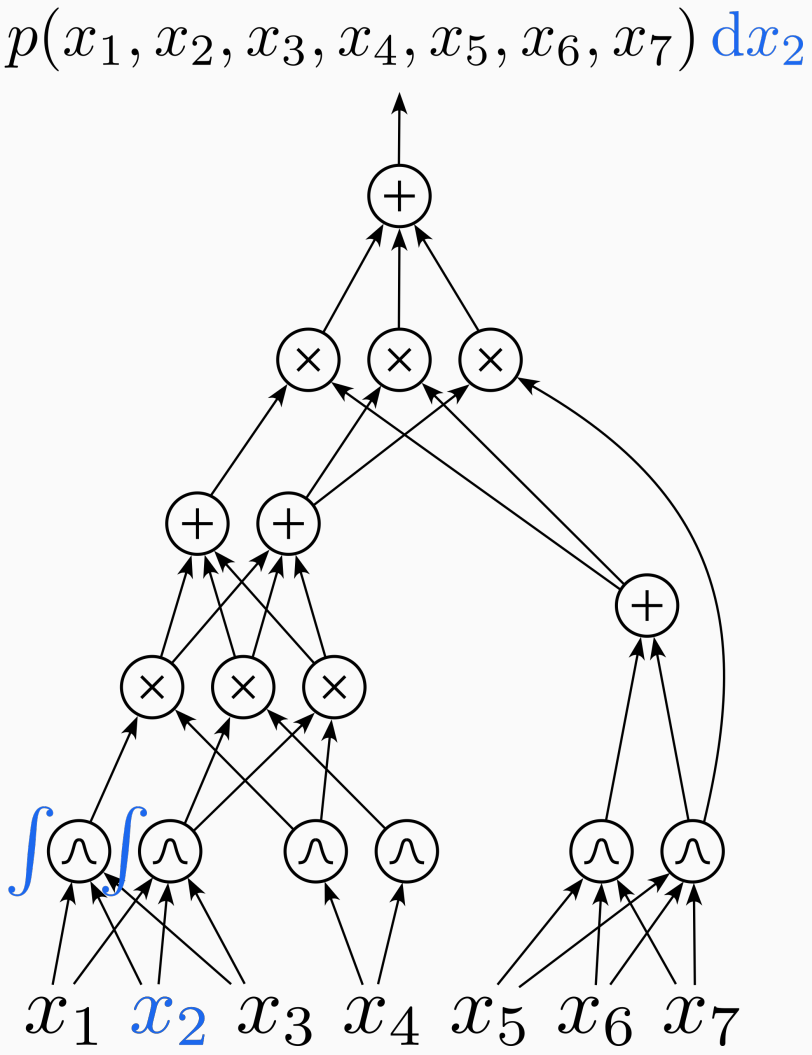
- decomposable products again



Marginal of PC

Example

- decomposable products again
- so we can push them down again

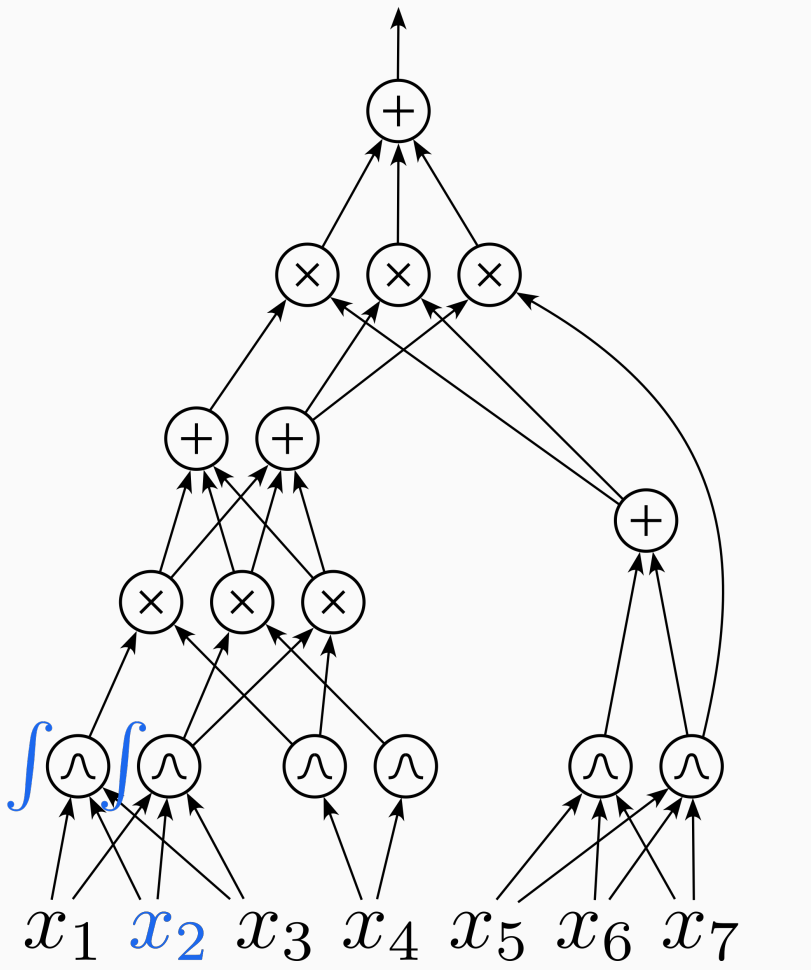


Marginal of PC

Example

- now, the integrals are at input distributions
- but we assumed that they allow tractable marginals!

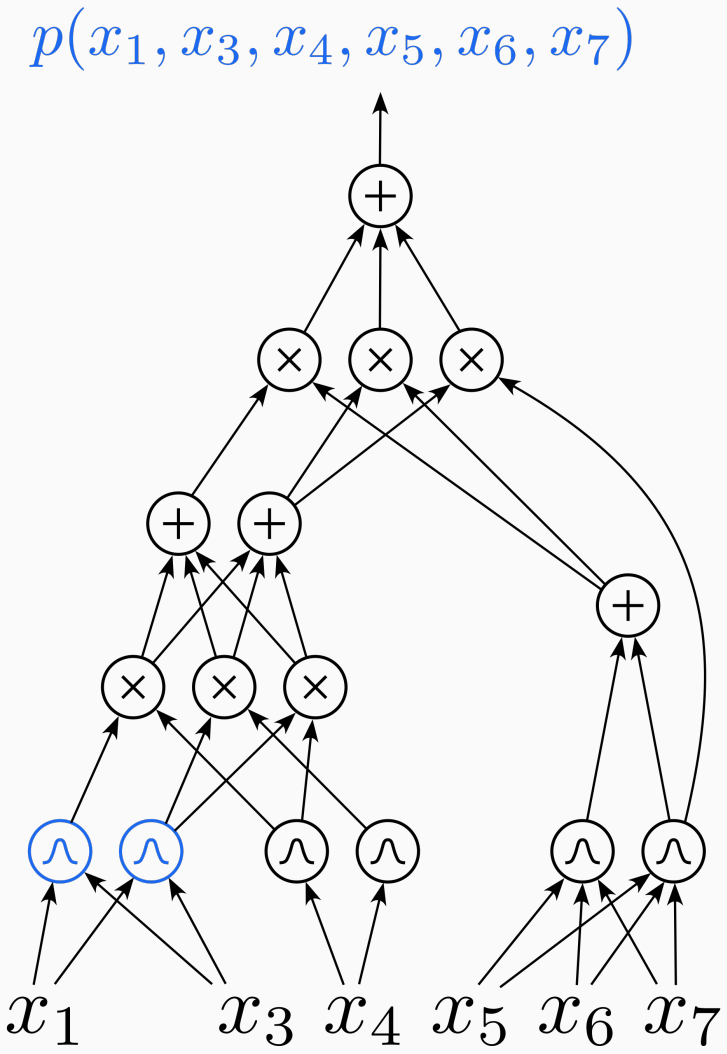
$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$



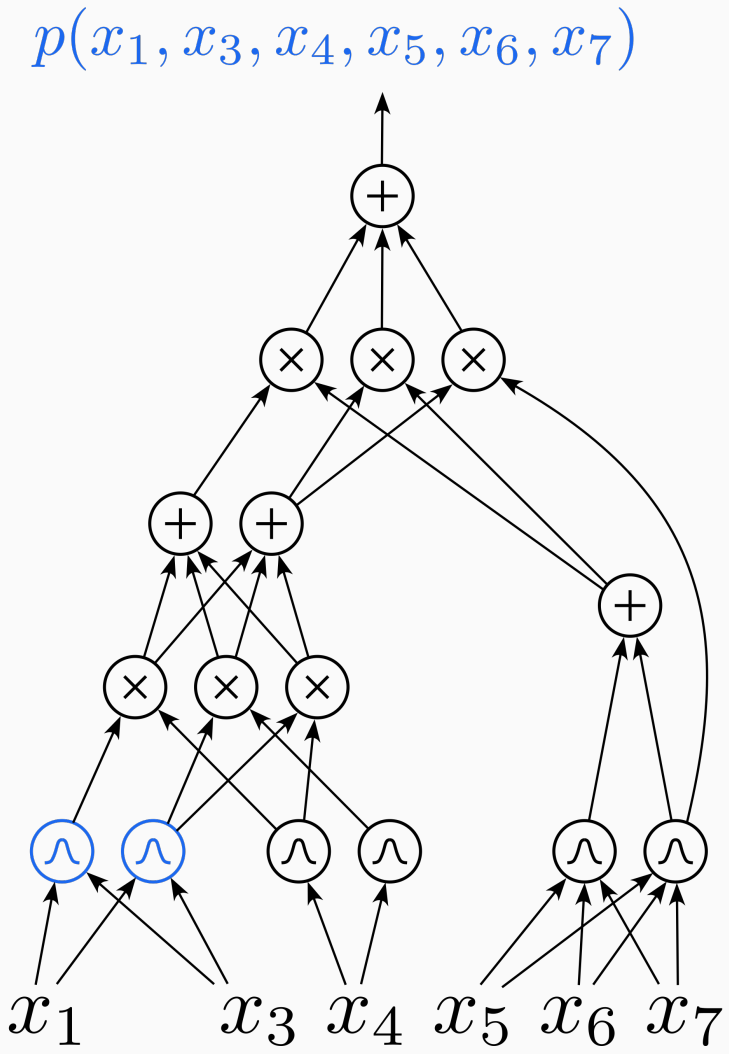
Marginal of PC

Example

- now, the integrals are at input distributions
- but we assumed that they allow tractable marginals!
- thus, just replace the input distributions containing X_2 with the marginals where X_2 is marginalized out
- this yields a new PC, representing the desired marginal $p(X_1, X_3, X_4, X_5, X_6, X_7)$



We showed marginalization for just one variable, but by repeating the argument, we can marginalize arbitrarily many RVs.



Conditional PCs

Let a PC be given which represents $p(\boldsymbol{x})$. Split $\boldsymbol{X}$ into $\boldsymbol{X}_e$ (**evidence**) and $\boldsymbol{X}_q$ (**query**). Upon observing that $\boldsymbol{X}_e = \boldsymbol{x}_e$, we want to derive the **conditional PC** representing

$$\overbrace{p(\boldsymbol{x}_q \mid \boldsymbol{x}_e)}^{\textbf{conditional}} = \frac{\overbrace{p(\boldsymbol{x}_q, \boldsymbol{x}_e)}^{\textbf{joint}}}{\underbrace{p(\boldsymbol{x}_e)}_{\textbf{marginal}}}$$

Like for marginalization, we will use the power of recursion—conditionals of nodes will be reduced to conditionals for their inputs.

Conditional of Product

Assume we have a decomposable product node

$$p(\boldsymbol{x}) = \prod_k p(\boldsymbol{x}_k)$$

Recall, decomposability means the blocks of RVs $\boldsymbol{X}_k$'s are non-overlapping.

Let's split for each block according to query and evidence:

$$\boldsymbol{X}_{q,k} := \boldsymbol{X}_q \cap \boldsymbol{X}_k$$
$$\boldsymbol{X}_{e,k} := \boldsymbol{X}_e \cap \boldsymbol{X}_k$$

so that we can write

$$p(\boldsymbol{x}) = \prod_k p(\underbrace{\boldsymbol{x}_{q,k}, \boldsymbol{x}_{e,k}}_{\boldsymbol{x}_k})$$

Conditional of Product cont'd

We have seen before that the marginal of a product is the the product of marginals, hence

$$p_{\text{prod}}(\boldsymbol{x}_e) = \prod_k p(\boldsymbol{x}_{e,k})$$

Thus,

$$p_{\text{prod}}(\boldsymbol{x}_q \mid \boldsymbol{x}_e) = \frac{\prod_k p(\boldsymbol{x}_{q,k}, \boldsymbol{x}_{e,k})}{\prod_k p(\boldsymbol{x}_{e,k})} = \prod_k p(\boldsymbol{x}_{q,k} \mid \boldsymbol{x}_{e,k})$$

The conditional of a product = product of the conditionals!

Conditional of Sum

The conditionals of sum nodes (mixture) are also easy. We use the fact that the conditional is proportional to the joint, $p(\boldsymbol{x}_q \mid \boldsymbol{x}_e) \propto p(\boldsymbol{x}_q, \boldsymbol{x}_e)$:

$$
\begin{aligned}
p_{\text{mix}}(\boldsymbol{x}_q \mid \boldsymbol{x}_e) &\propto \sum_{k=1}^{K} w_k \overbrace{p_k(\boldsymbol{x}_q, \boldsymbol{x}_e)}^{k^{\text{th}} \text{ input}} \\
&= \sum_{k=1}^{K} w_k \overbrace{p_k(\boldsymbol{x}_q \mid \boldsymbol{x}_e)\, p_k(\boldsymbol{x}_e)}^{\text{chain rule of probability}} \\
&= \sum_{k=1}^{K} \overbrace{w_k \underbrace{p_k(\boldsymbol{x}_e)}_{\text{marginal}}}^{\text{not normalized}} \overbrace{p_k(\boldsymbol{x}_q \mid \boldsymbol{x}_e)}^{\text{normalized}}
\end{aligned}
$$

By replacing $w_k\, p_k(\boldsymbol{x}_e)$ with $\tilde{w}_k = \frac{w_k\, p_k(\boldsymbol{x}_e)}{\sum_i w_i\, p_i(\boldsymbol{x}_e)}$, we can conclude that

$$
p_{\text{mix}}(\boldsymbol{x}_q \mid \boldsymbol{x}_e) = \sum_{k=1}^{K} \tilde{w}_k\, p_k(\boldsymbol{x}_q \mid \boldsymbol{x}_e)
$$

Conditional of Sum cont'd

The conditional of a mixture = a mixture of the conditionals!

. . . with modified mixture weights

$$\tilde{w}_k = \frac{w_k \, p_k(\boldsymbol{x}_e)}{\sum_i w_i \, p_i(\boldsymbol{x}_e)}$$

- **probabilistic circuits**: a framework for tractable probabilistic modelling
- computational graph of
  - **input distributions**
  - **product nodes** (factorizations)
  - **sum nodes** (mixtures)
- **smoothness** and **decomposability** are important structural properties, leading to an interpretation as hierarchical mixture model
- crucially, they are required for tractable marginals and conditionals
- stay tuned for
  - **determinism**: allows exact maximization
  - **structured decomposability** and **compatibility**: allows advanced reasoning tasks and neuro-symbolic tricks

Progress and Challenges

Progress: PCs at Speed and Scale

$100 - 200X$ slower than MLPs

Random Sum-Product Networks:
A Simple and Effective Approach to Probabilistic Deep Learning

---

$50X$ slower than MLPs

Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits

---

$5X$ slower than MLPs

Scaling Tractable Probabilistic Circuits: A Systems Perspective

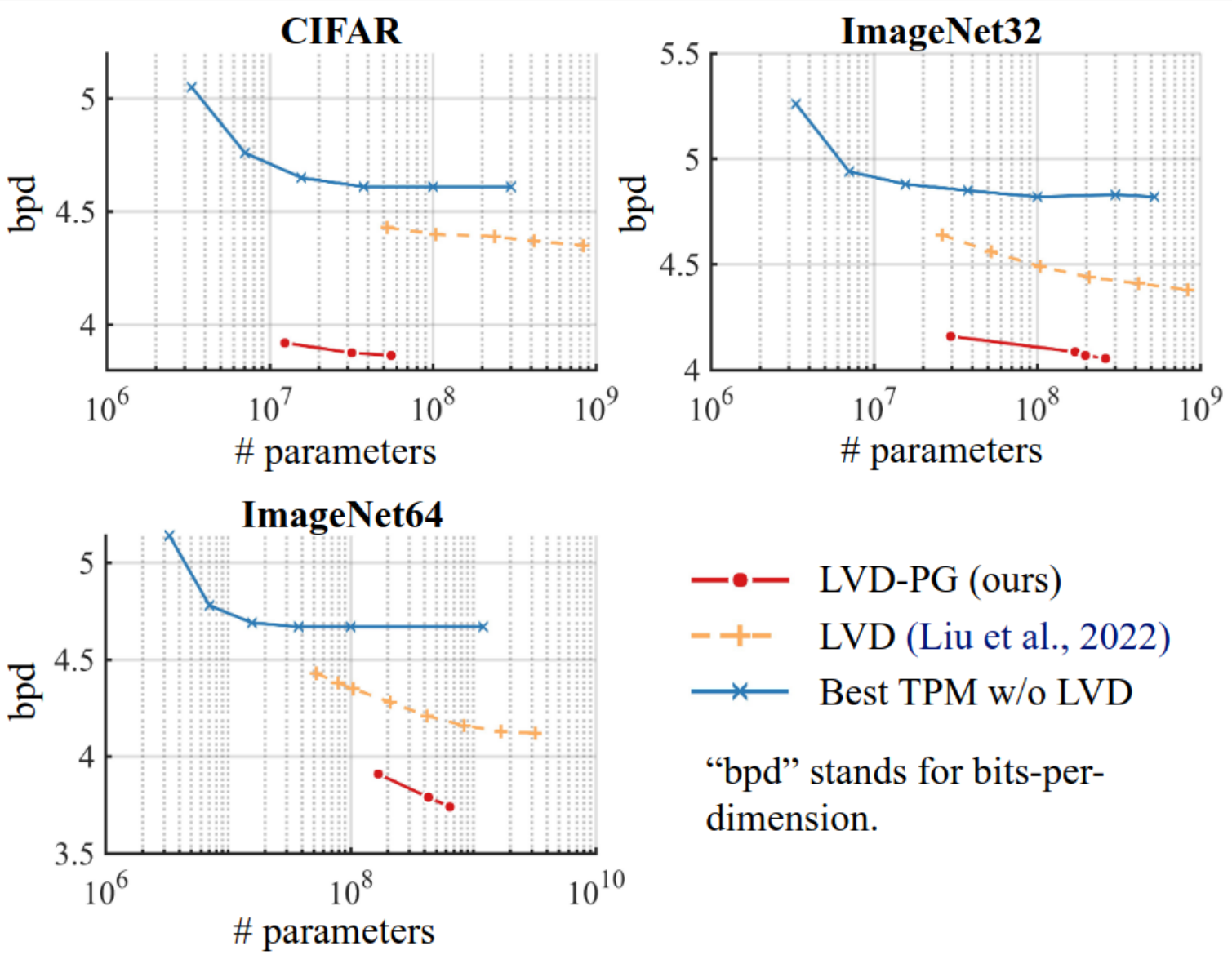
---

en par or faster than MLPs

Challenge: Expressiveness-vs-Tractability

- constraints are the price to pay for tractability
- constraints, however, limit the expressive power of an architecture (example: deep vs. shallow networks)
- hence, there is a **expressiveness-vs-tractability trade-off**
- for some classes of PCs, this trade-off is theoretically well understood (exponential separation)
- But, we can scale! This will surely save us, right?

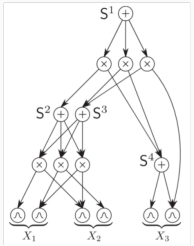
Challenge: Scale is not all we need

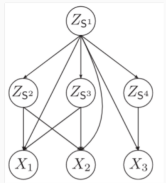


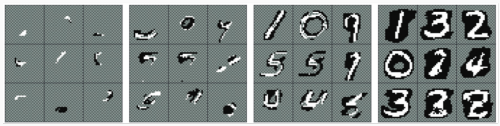
Liu et al., 2023

Progress: The many connections of PCs

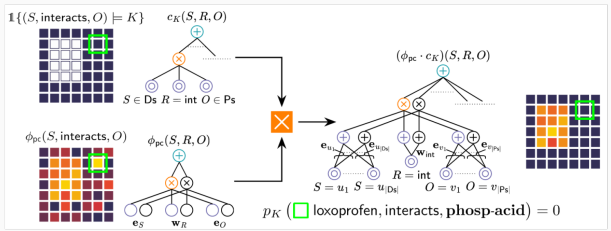
PCs as Latent Variable Models



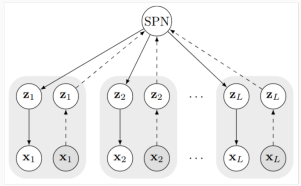


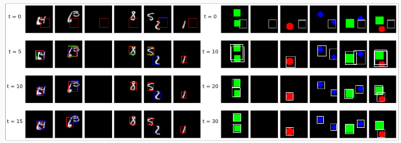


PCs x Knowledge Graphs

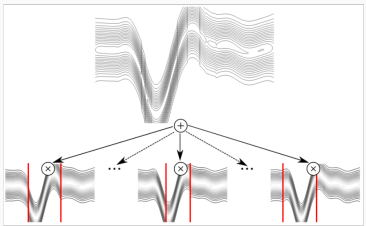


PCs x VAEs

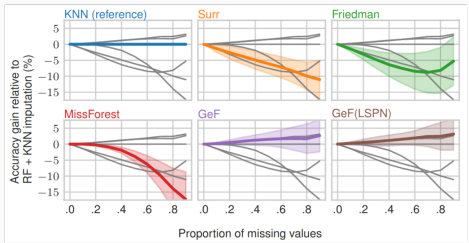




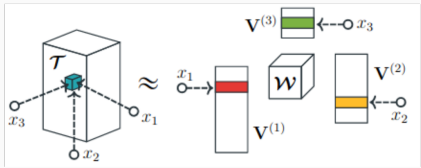
PCs x Gaussian Processes

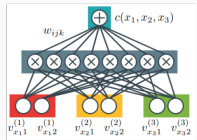


PCs x Decision Trees



PCs x Tensor Decompositions

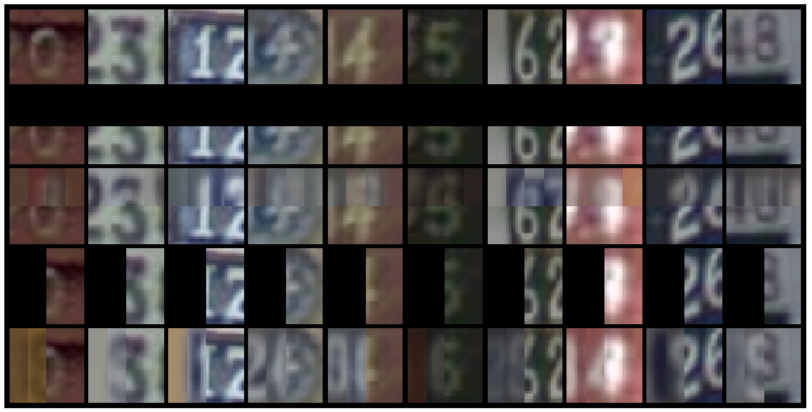




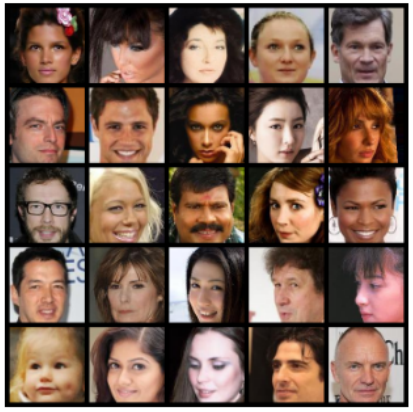
Challenge: Sample Quality (But, Exact Inpainting)

(a) Real SVHN images.

(b) EiNet SVHN samples.



(c) Real images (top), covered images, and EiNet reconstructions



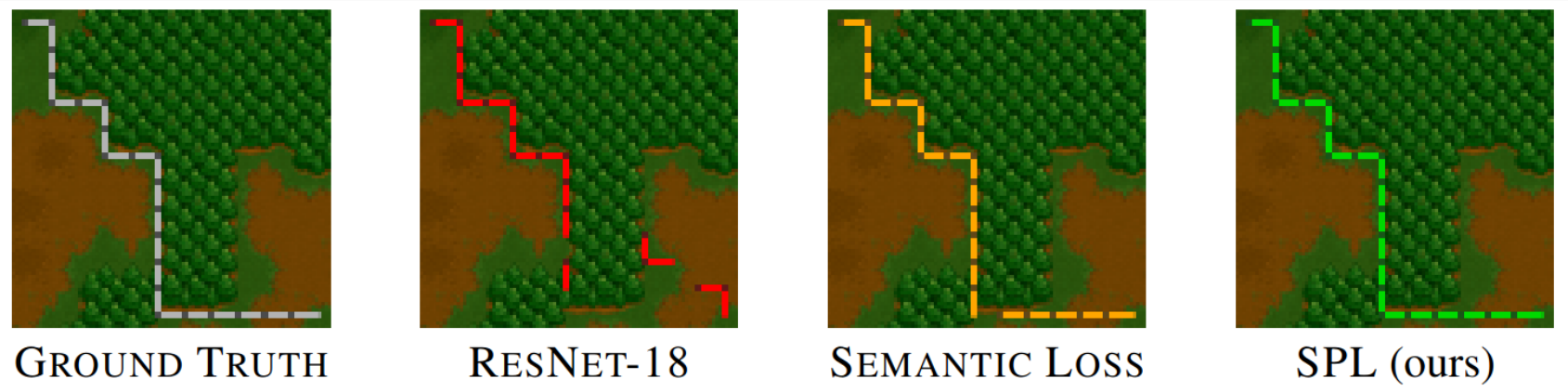
(d) Real CelebA samples.

(e) EiNet CelebA samples.

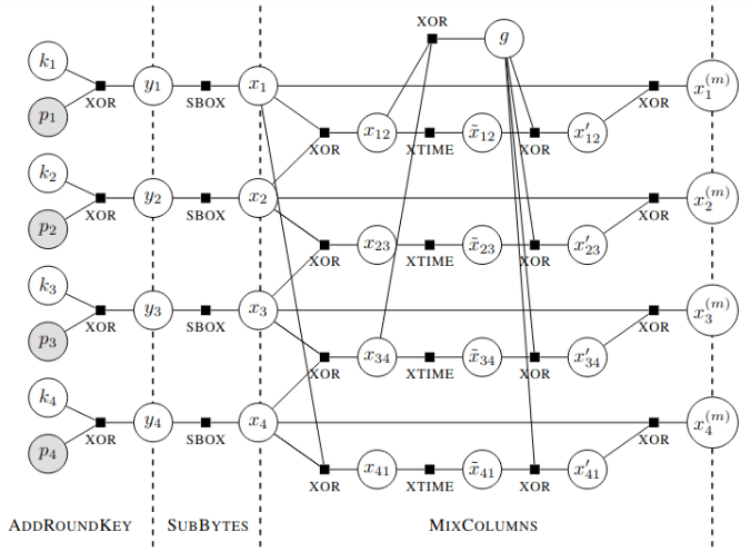
(f) Real images (top), covered images, and EiNet reconstructions

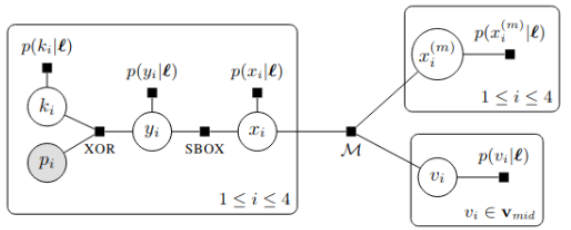
Peharz et al., 2020

Progress: Neurosymbolic ML



Ahmed et al., 2022





Inference Method	Top-1 Success Rate		
	$\varepsilon = 10^{-2}$	$\varepsilon = 10^{-8}$	$\varepsilon = 0$
Baseline	0.10%	0.10%	0.10%
SASCA (3 iters)	0.52%	0.52%	0.52%
SASCA (50 iters)	32.34%	33.81%	33.76%
SASCA (100 iters)	34.25%	36.04%	35.84%
ExSASCA + MAR	57.36%	67.37%	67.37%
ExSASCA + MPE	**57.51%**	**67.60%**	**67.61%**

Wedenig et al., 2024

References

Peharz et al., On the latent variable interpretation in sum-product networks, TPAMI 2017.
Peharz et al., Einsum networks: Fast and scalable learning of tractable probabilistic circuits, ICML 2020.
Vergari et al., Sum-product autoencoding: Encoding and decoding representations using sum-product networks, AAAI 2018.
Butz et al., Sum-product network decompilation, PGM 2020.
Loconte et al., How to turn your knowledge graph embeddings into generative models, NeurIPS 2023.
Stelzner et al., Faster attend-infer-repeat with tractable probabilistic models, ICML 2018.
Tan and Peharz, Hierarchical decompositional mixtures of variational autoencoders, ICML 2018.
Trapp et al., Deep structured mixtures of gaussian processes, AISTATS 2020.
Correia et al., Continuous mixtures of tractable probabilistic models, AAAI, 2023.
Gala et al., Probabilistic integral circuits, AISTATS 2024.
Correia et al., Joints in random forests, NeurIPS 2020.
Khosravi et al., Handling Missing Data in Decision Trees: A Probabilistic Approach, 2020
Ventola et al., Relational Decision Trees as Probabilistic Circuits, Springer 2021.
Loconte et al., What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)?, 2024
Ahmed et al., Semantic Probabilistic Layers for Neuro-Symbolic Learning, NeurIPS 2022.
Wedenig et al., Exact Soft Analytical Side-Channel Attacks using Tractable Circuits, ICML 2024.
Liu et al., Understanding the distillation process from deep generative models to tractable probabilistic circuits, ICML 2023.
Liu et al., Scaling Tractable Probabilistic Circuits: A Systems Perspective, ICML 2024.