Probabilistic Circuits: Tractable Representations for Learning and Reasoning

Lecture 1 – Intro to Probabilistic Circuits

Robert Peharz,[1] **Antonio Vergari**[2]
[1]Graz University of Technology
[2]University of Edinburgh

European Summer School on Artificial Intelligence
Athens, 15th July 2024

What this course is about

- **probabilistic circuits (PCs)**, a framework for

  tractable probabilistic (generative) modelling

- PCs allow to do what probabilistic models should do:

  answer queries of interest exactly and efficiently

- most of the current probabilistic models (VAEs, GANs, Diffusion-based, Flows, etc.) ignore tractability and require approximations, with little to no guarantees

Goals and Prerequisites

- **Goal: Make you see AI through PC glasses**
  - introduce PCs and their working
  - how to use and combine PCs with other approaches
  - neurosymbolic ML with PCs
- this course is an Advanced Course, continuing our Introductory Course from least year's ESSAI:
  `https://tinyurl.com/PCs-ESSAI`
- **prior knowledge about PCs is not necessary but helpful**
- however, a fair knowledge about statistics or probabilistic ML will be required

Schedule

- **Monday: Intro to Probabilistic Circuits** (Robert)
- **Tuesday: Implementation and Algorithms** (Antonio)
- **Wednesday: Mixing PCs with other Models** (Robert)
- **Thursday: PCs for Neurosymbolic ML** (Antonio)
- **Friday: PCs Everywhere!** (Antonio and Robert)

Notation

- **random variables** (RVs) are written with uppercase letters, potentially with subscripts, e.g. X, Y_i, Z_k
- **values** or **states** of RVs are written with lowercase letters, e.g. x, y, z
- boldface font denotes "many", so that $\boldsymbol{X}$, $\boldsymbol{Y}_j$, $\boldsymbol{Z}_l$ are **sets of random variables** (**random vectors**)
- boldface lowercase letters are then corresponding **joint states**, e.g. $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$
- **(joint) distribution functions** are written as
  - $p(x)$, $p(y)$, $p(z)$
  - $p(\boldsymbol{x}) = p(x_1, x_2, x_3)$ (where $\boldsymbol{X} = (X_1, X_2, X_3)$)
  - $p(\boldsymbol{z}) = p(\boldsymbol{x}, \boldsymbol{y})$ (where $\boldsymbol{Z} = \boldsymbol{X} \cup \boldsymbol{Y}$)

Notation

- in math, one uses subscripts to indicate which density belongs to which RV, e.g. p_X is the distribution of X, and $p_X(x)$ is the evaluation for state x
- the subscript seems redundant, thus in ProbML one simply drops it, and writes $p(x)$ for the evaluation
- to distinguish the distribution from its evaluation, one writes $p(X)$ instead of p_X
- one might also interpret $p(X)$ as "$p(x)$ for all possible x"

Probability Mass Functions

A **probability mass function** (PMF) over **discrete** RVs $\boldsymbol{X}$ assigns to each state $\boldsymbol{x}$ the probability that $\boldsymbol{X} = \boldsymbol{x}$. Example for some $\boldsymbol{X}$ containing three **binary** variables:

$\boldsymbol{x}$	$p(\boldsymbol{x})$
$(0, 0, 0)$	0.2475
$(0, 0, 1)$	0.0025
$(0, 1, 0)$	0.125
$(0, 1, 1)$	0.125
$(1, 0, 0)$	0.125
$(1, 0, 1)$	0.125
$(1, 1, 0)$	0
$(1, 1, 1)$	0.25

A valid PMF must be **non-negative and sum to** 1**.**

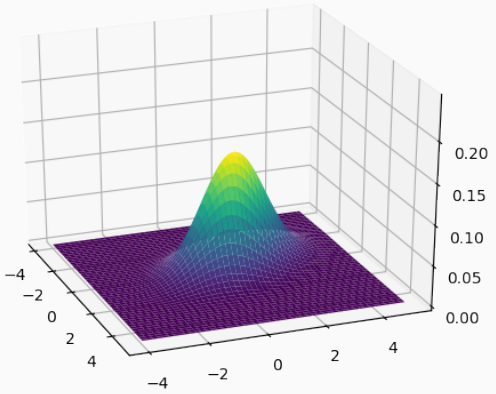
Probability Densities

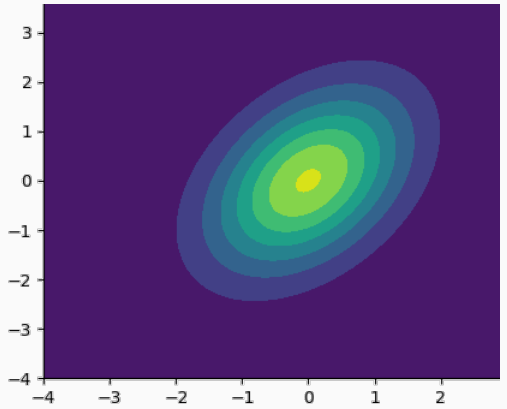
Continuous RVs are described by **probability density functions** (PDFs). One of the simplest examples is the **Gaussian**:

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

$\boldsymbol{\mu}$: **mean vector** (D-dimensional)

$\boldsymbol{\Sigma}$: **covariance matrix** ($D \times D$-positive definite)





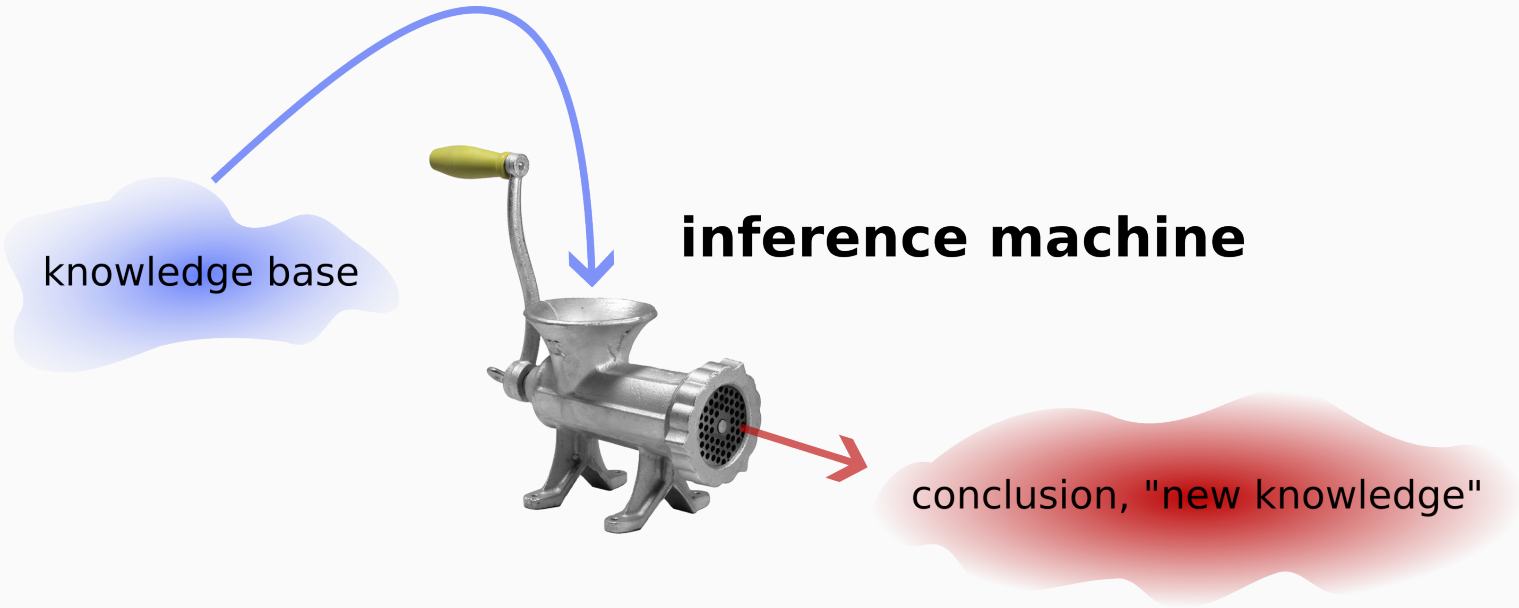
$p(\boldsymbol{X})$ must be non-negative, but might well be larger than 1! Only integrating $p(\boldsymbol{X})$ over some set $A \subset \mathbb{R}^D$ gives the probability that $\boldsymbol{X} \in A$. Thus, the total area under the density must be 1.

Joint Densities

- we will often just say "density" and mean **both** probability densities and probability mass functions, because
  - it's easier
  - there is actually no difference in a measure theoretic treatment (just a different base measure)
- we will often also say "distribution" instead of density
- **note:** a joint distribution $p(\boldsymbol{X})$ over $\boldsymbol{X}$ describes
  - all dependencies between the individual RVs in $\boldsymbol{X}$
  - the uncertainty about $\boldsymbol{X}$
- **probabilistic (generative) model**: a description of a joint distribution over some $\boldsymbol{X}$ (possibly conditional on other stuff)

Probabilistic Inference

Reasoning



- the ability to reason is central to any form of intelligence
- for AI, this process should be **automatizable**
- examples: classical logic, equations, etc.
- in ProbML:

  joint distribution = knowledge base + uncertainty

- what is the inference machine?

Marginal Distribution

Let p be the distribution of $\boldsymbol{X}$, and let $\boldsymbol{Y}$ and $\boldsymbol{Z} = \{Z_1, \ldots, Z_K\}$ be any partition of $\boldsymbol{X}$, i.e. $\boldsymbol{Y} \cap \boldsymbol{Z} = \emptyset$ and $\boldsymbol{Y} \cup \boldsymbol{Z} = \boldsymbol{X}$. Then

$$p(\boldsymbol{Y}) = \int_{\mathcal{Z}} p(\boldsymbol{Y}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = \int_{\mathcal{Z}_1} \cdots \int_{\mathcal{Z}_K} p(\boldsymbol{Y}, z_1, \ldots, z_K) \, \mathrm{d}z_1 \ldots \mathrm{d}z_K$$

is the **marginal** distribution of $\boldsymbol{Y}$. We say the $\boldsymbol{Z}$'s are marginalized from the model. For discrete RVs, integrals are replaced by sums.

Semantics of marginalization: "ignore", "account for unknowns"

When $\boldsymbol{Z}$ contains many variables, marginalization is in general computationally hard. But it is conceptually simple and the right thing to do.

Conditional Distribution

Let $p(\boldsymbol{Y}, \boldsymbol{Z})$ the **joint distribution** of $\boldsymbol{Y}$ and $\boldsymbol{Z}$. The **conditional distribution** is defined as (for $p(\boldsymbol{z}) > 0$):

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

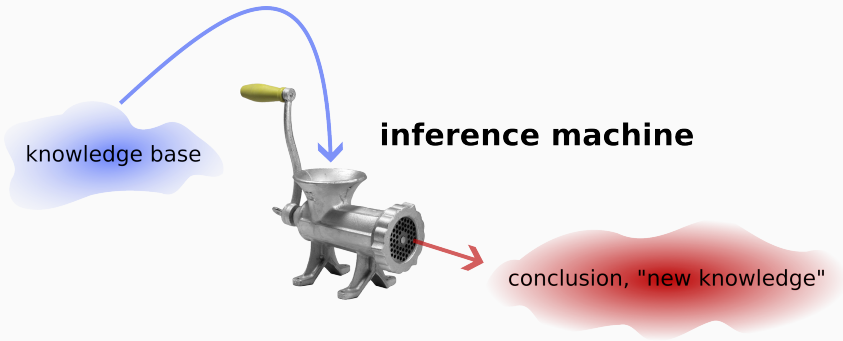
Conversely, if **marginal** $p(\boldsymbol{Z})$ and **conditional** $p(\boldsymbol{Y} \mid \boldsymbol{Z})$ are given, the **joint distribution** is given as

$$p(\boldsymbol{Y}, \boldsymbol{Z}) = p(\boldsymbol{Y} \mid \boldsymbol{Z}) \, p(\boldsymbol{Z})$$

Semantics of conditioning: "observe", "inject information"

Conditioning uses marginalization as a "subroutine".

Probabilistic Inference



marginalization

ignoring, accounting for unknown quantities.

conditioning

observing, introducing evidence.

These two are the most important reasoning tools in probabilistic modeling!

Further primitives: **sampling**, **compute density**, **expectations**, **maximization**, . . .

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)	✗	✗	✗	✗	✗	✓ (✗)
$\mathbb{E}$	✗	✗	✗	✗	✗	✓ (✗)

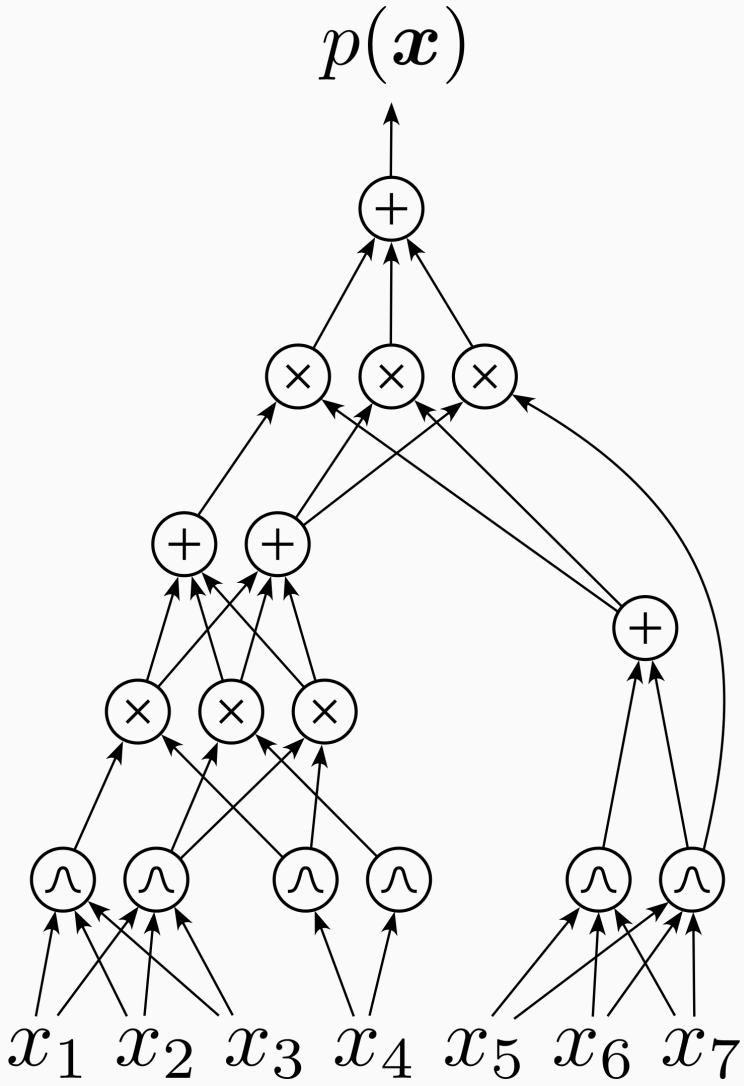
Probabilistic Circuits

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

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

Input: values $\boldsymbol{x}$ for RVs $\boldsymbol{X}$ we want to model

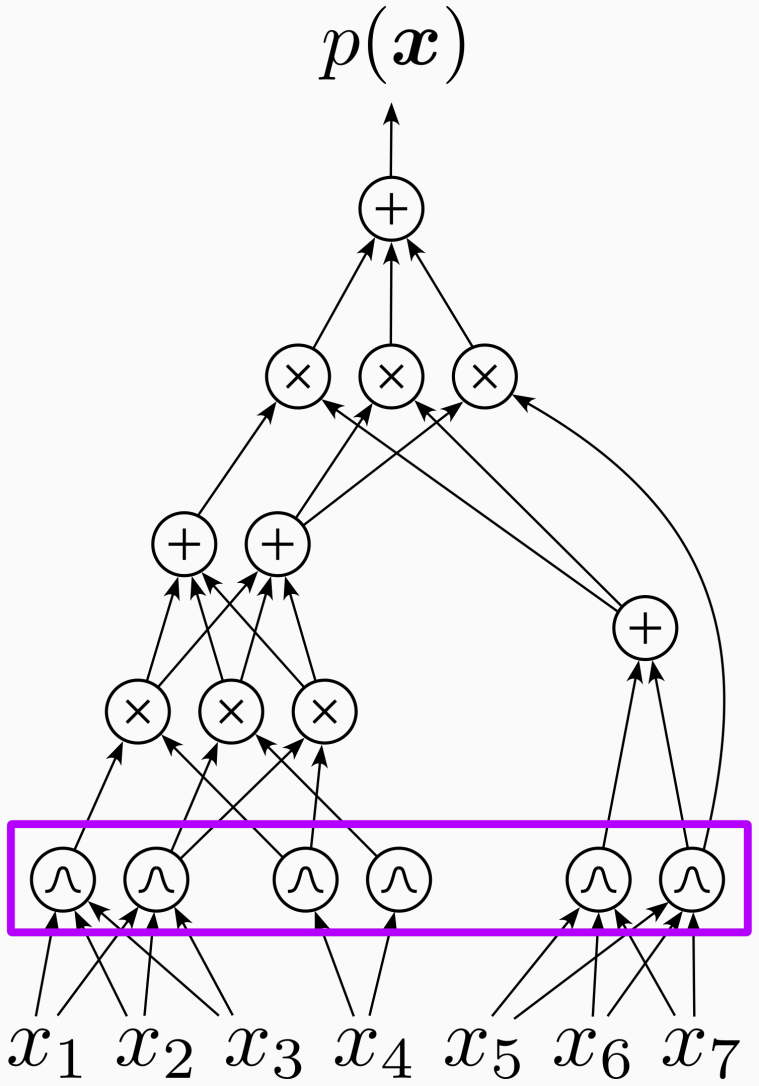
Output: joint density $p(\boldsymbol{x})$ evaluated at $\boldsymbol{x}$ (perhaps unnormalized)



We will explain three types of nodes in the following.

But first note that

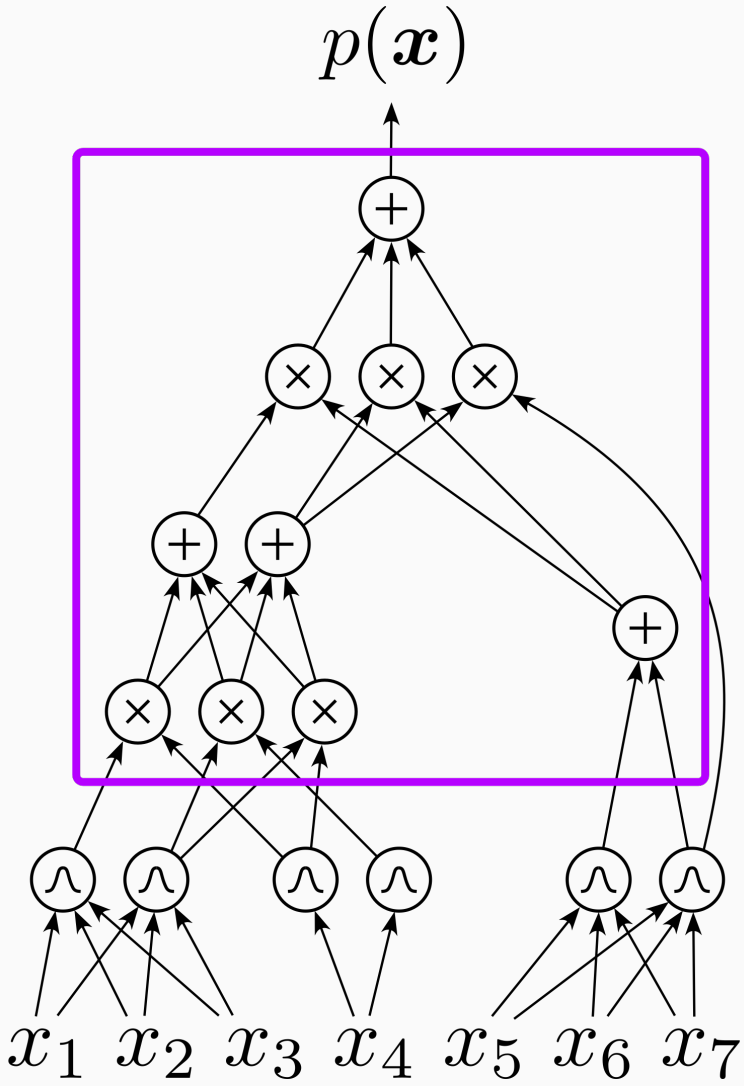
- all nodes on the first layer are distributions



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 

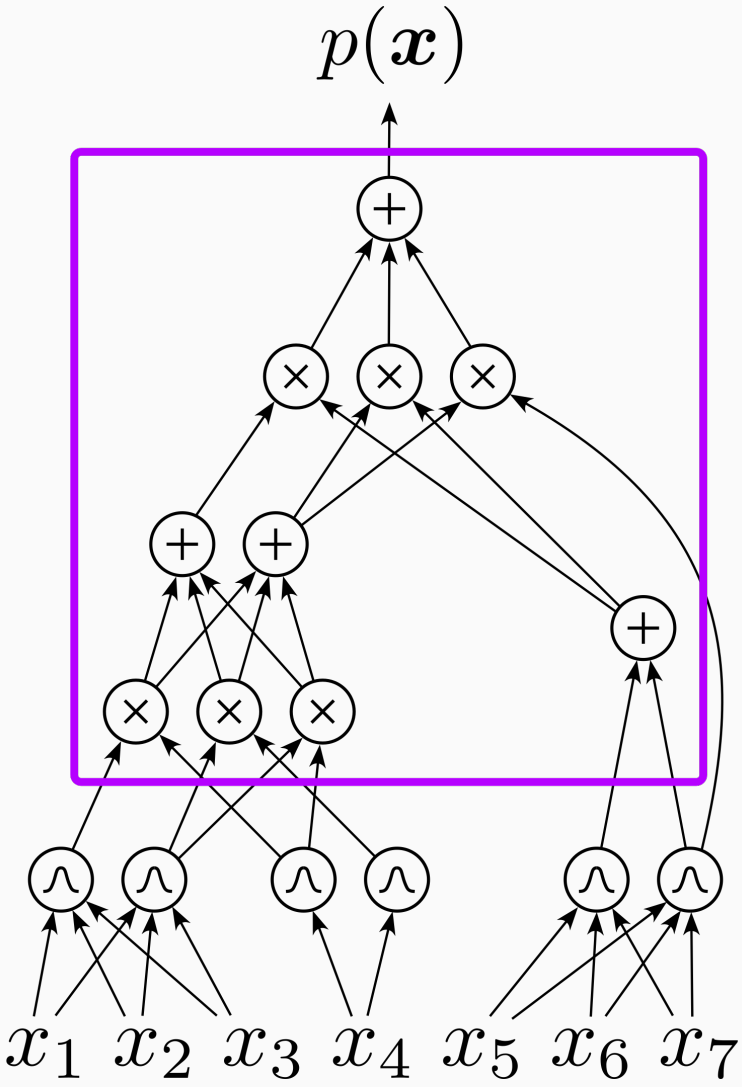


We will explain three types of nodes in the following.

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.

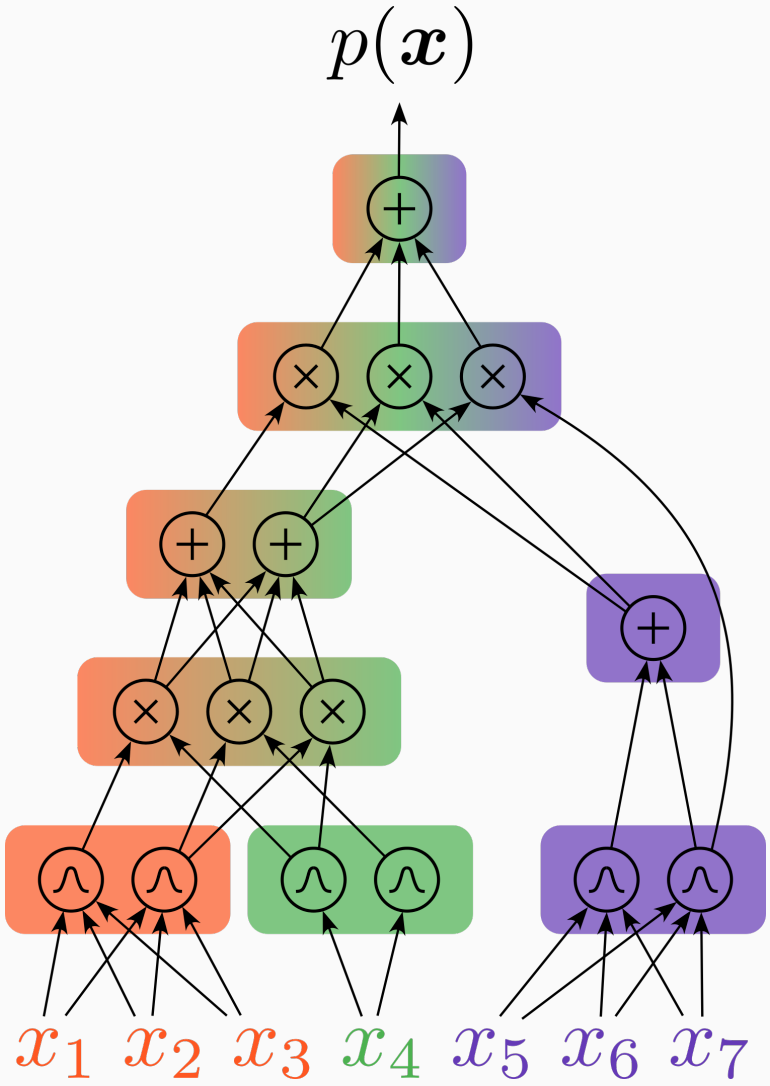


Furthermore, note that **each node depends only on a subset of the inputs**, i.e. each node has a restricted **receptive field** or **scope**.

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

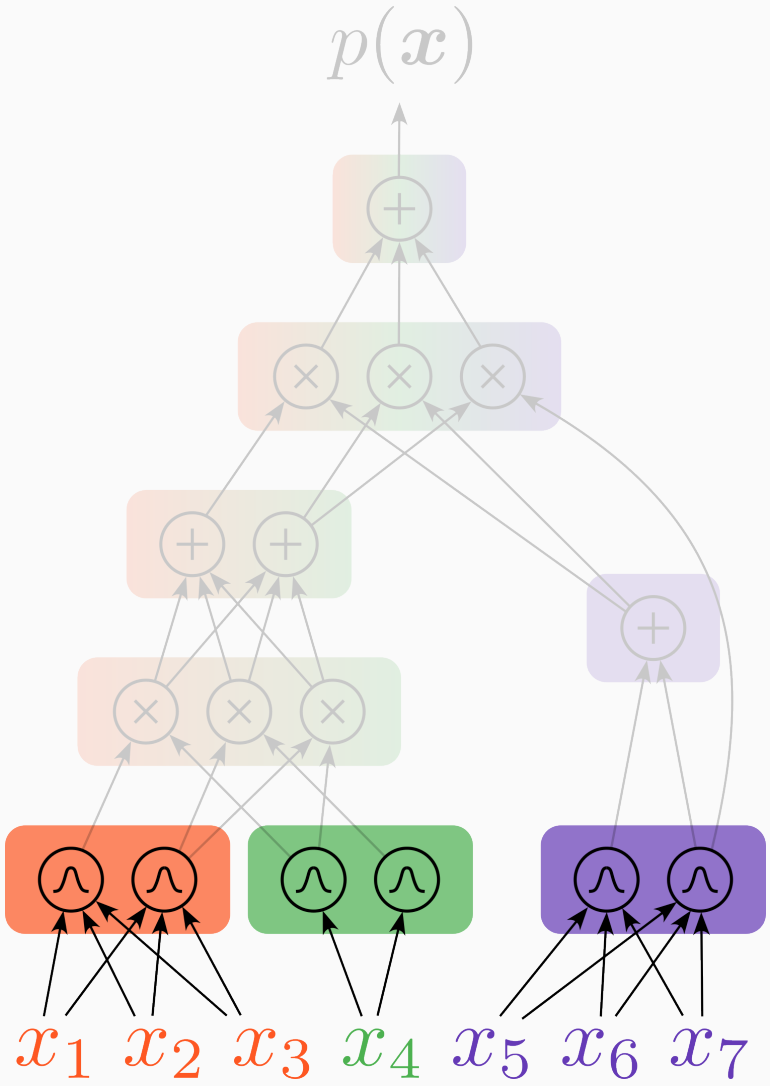
The scope of a node N is defined as the set of RVs it depends on. The **output node** of a PC has always full scope $\boldsymbol{X}$.

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



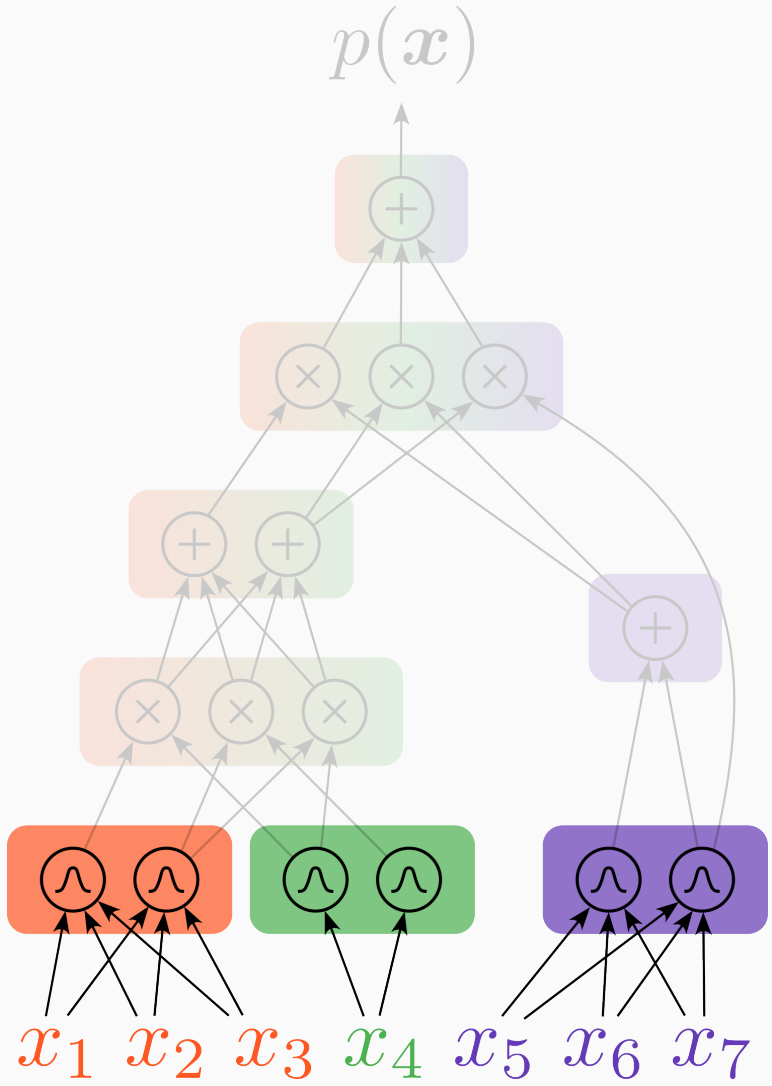
A **distribution (node)**  is just **some** density over its scope. In the **forward pass** it evaluates this density, which serves as input to the next nodes.

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

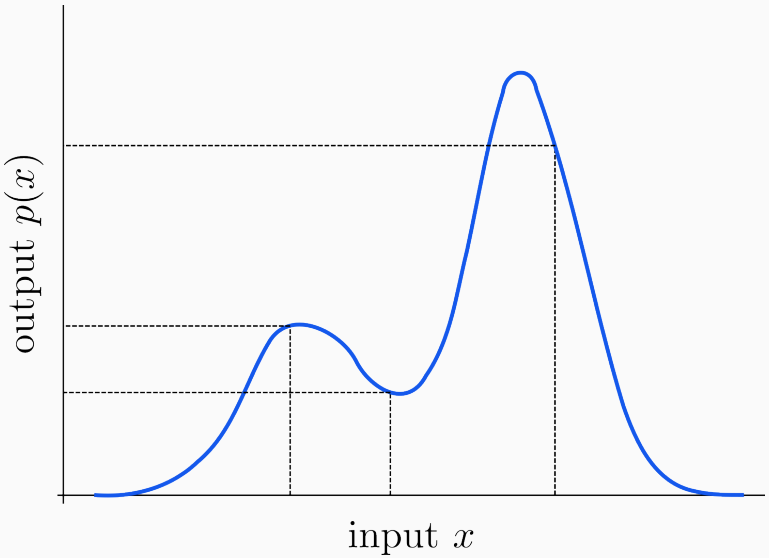
- 2 (joint) distributions over $\{X_1, X_2, X_3\}$ (orange, left)
- 2 distributions over $\{X_4\}$ (green, center)
- 2 (joint) distributions over $\{X_5, X_6, X_7\}$ (purple, right)



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

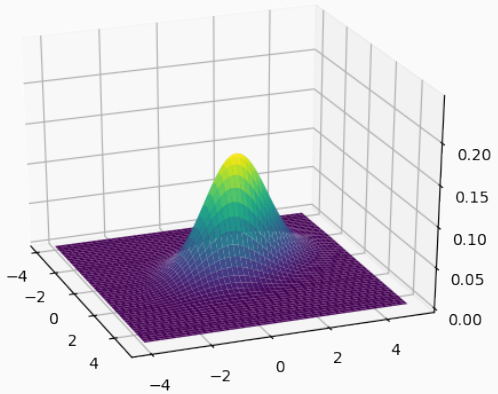
- probability densities and mass functions (can model both continuous **and** discrete RVs)
- whole other generative models (e.g. VAEs, Gaussian processes)
- distributions without density (probability measures)

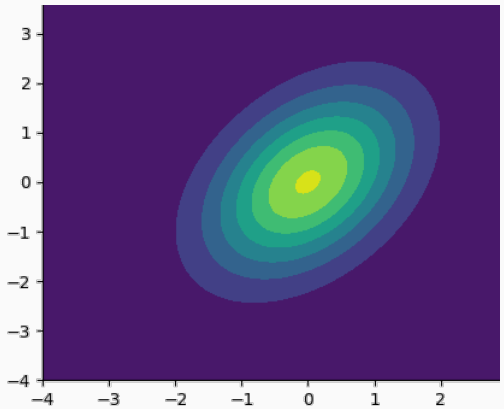
However, Gaussians are indeed often used as distribution nodes.

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:

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

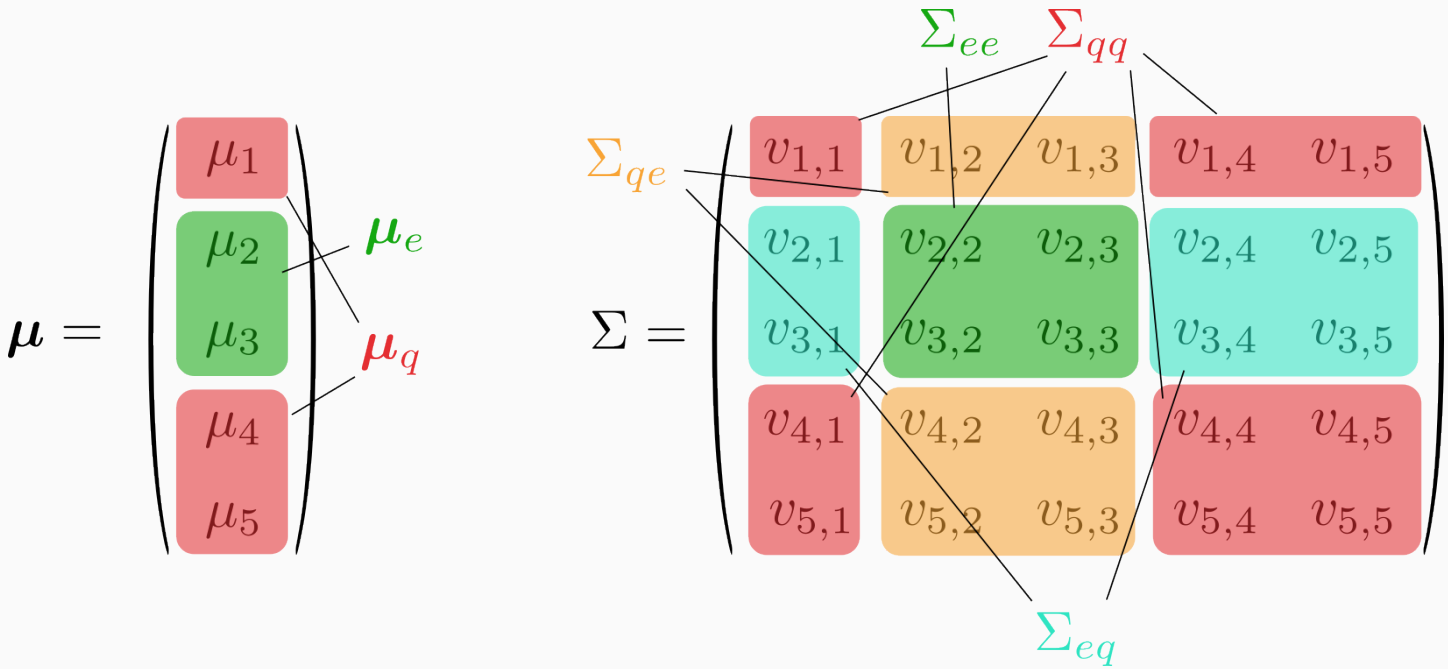
$$\boldsymbol{\mu}_q = \begin{pmatrix} \mu_1 \\ \mu_4 \\ \mu_5 \end{pmatrix} \qquad \Sigma_{qq} = \begin{pmatrix} v_{1,1} & v_{1,4} & v_{1,5} \\ v_{4,1} & v_{4,4} & v_{4,5} \\ v_{5,1} & v_{5,4} & v_{5,5} \end{pmatrix}$$

Gaussian Conditional Distributions

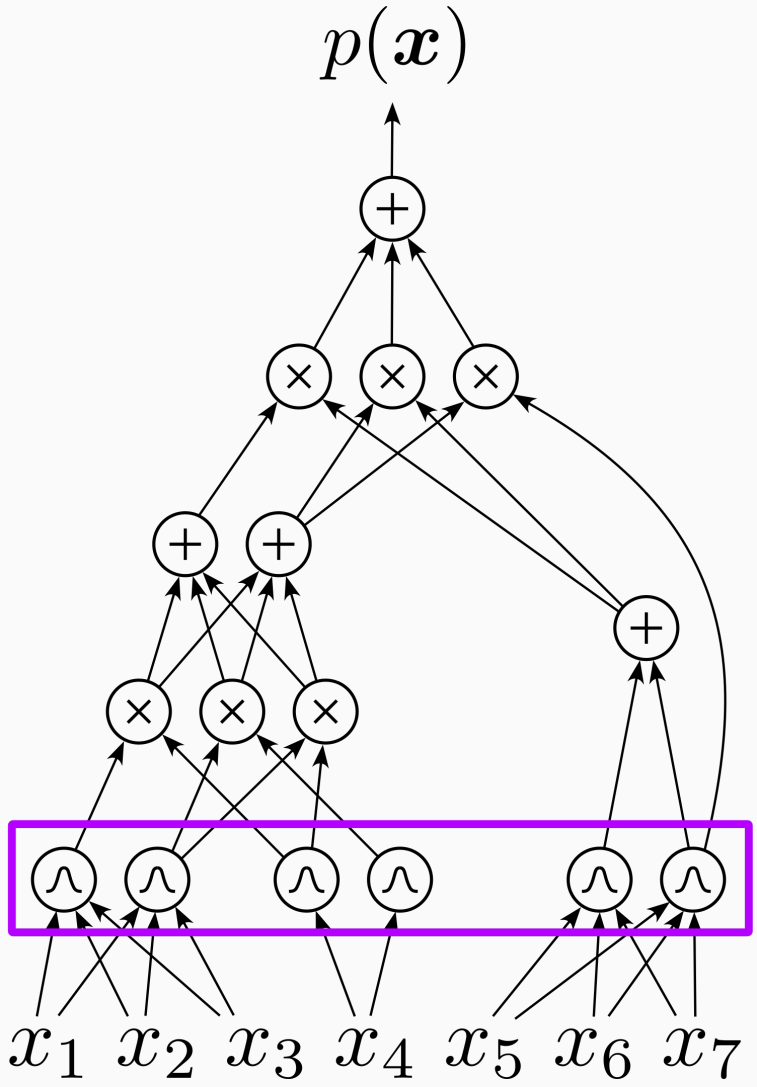
In Gaussians with parameters $\boldsymbol{\mu}$ and Σ, any conditional distribution $p(\boldsymbol{X}_q \mid \boldsymbol{X}_e = \boldsymbol{x}_e)$ is again Gaussian with **closed form parameters**:

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

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



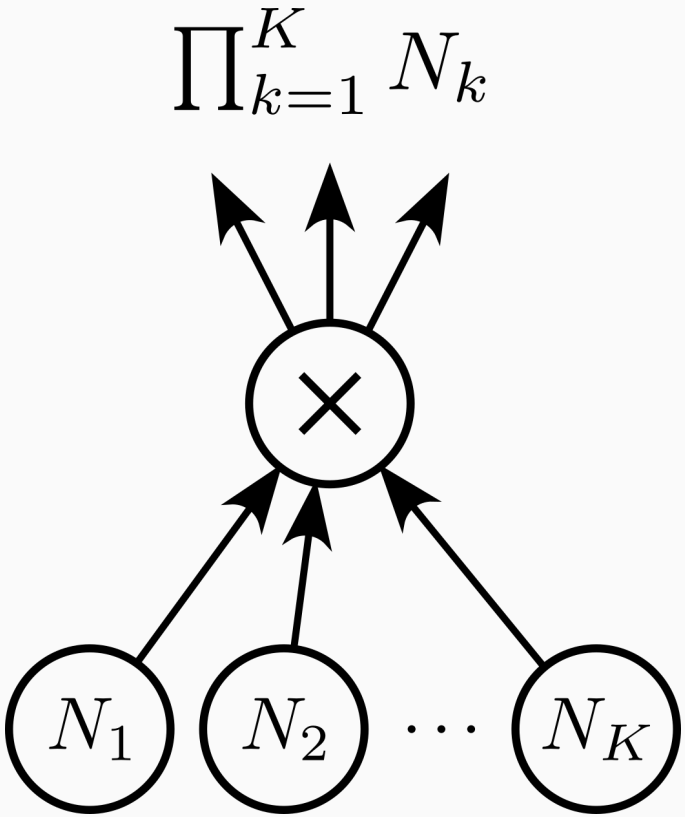
- **distribution nodes** are in the first layer of PCs (hence, sometimes called **input distributions** or **leaves**)
- they represent a density over their **scope**
- each distribution node has its own parameters
- should allow **tractable inference** (**marginalization**, **conditioning**)
- Gaussians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

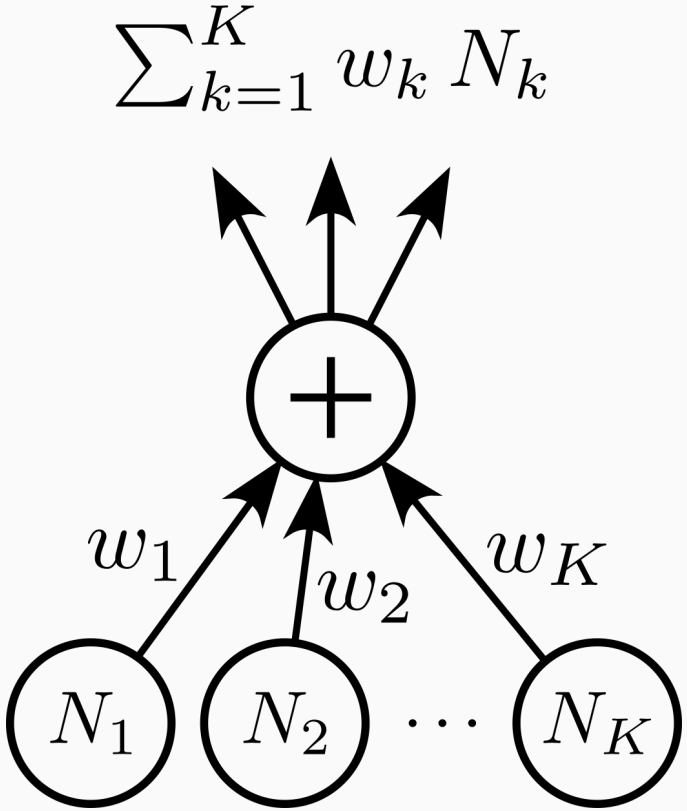


- a **sum node** S computes a weighted sum of its inputs (**linear unit**)
- if S is a sum node with inputs $N_1, N_2, \ldots, N_K$, it computes

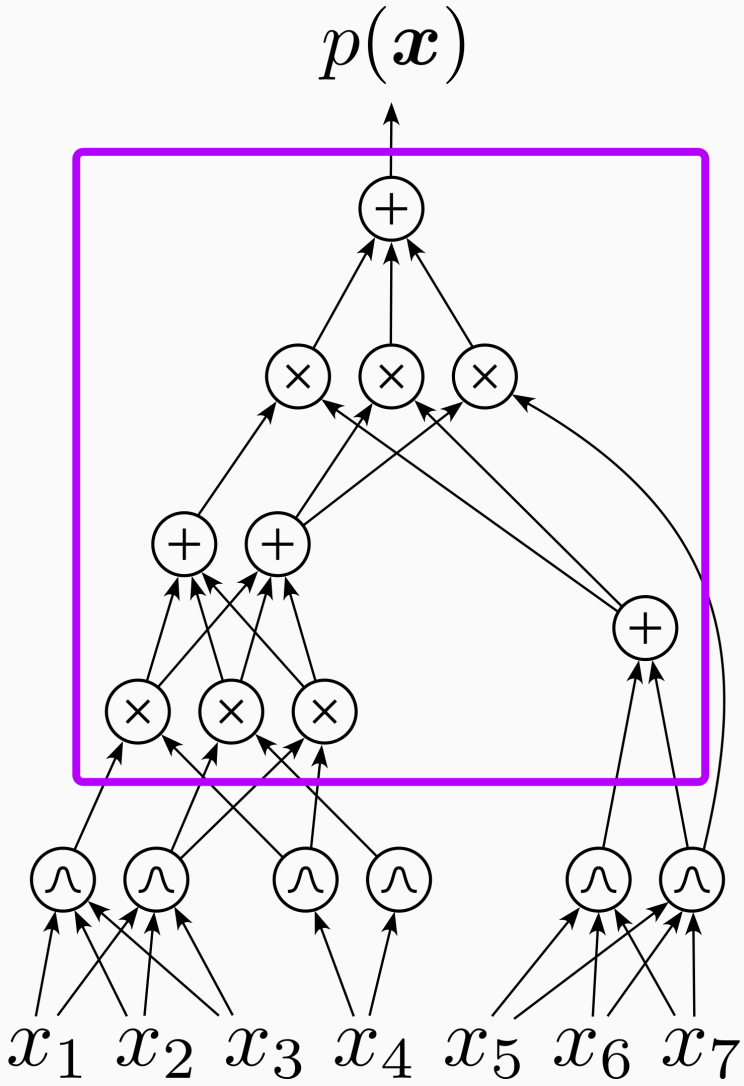
$$S := \sum_{k=1}^{K} w_k \, N_k$$

- 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

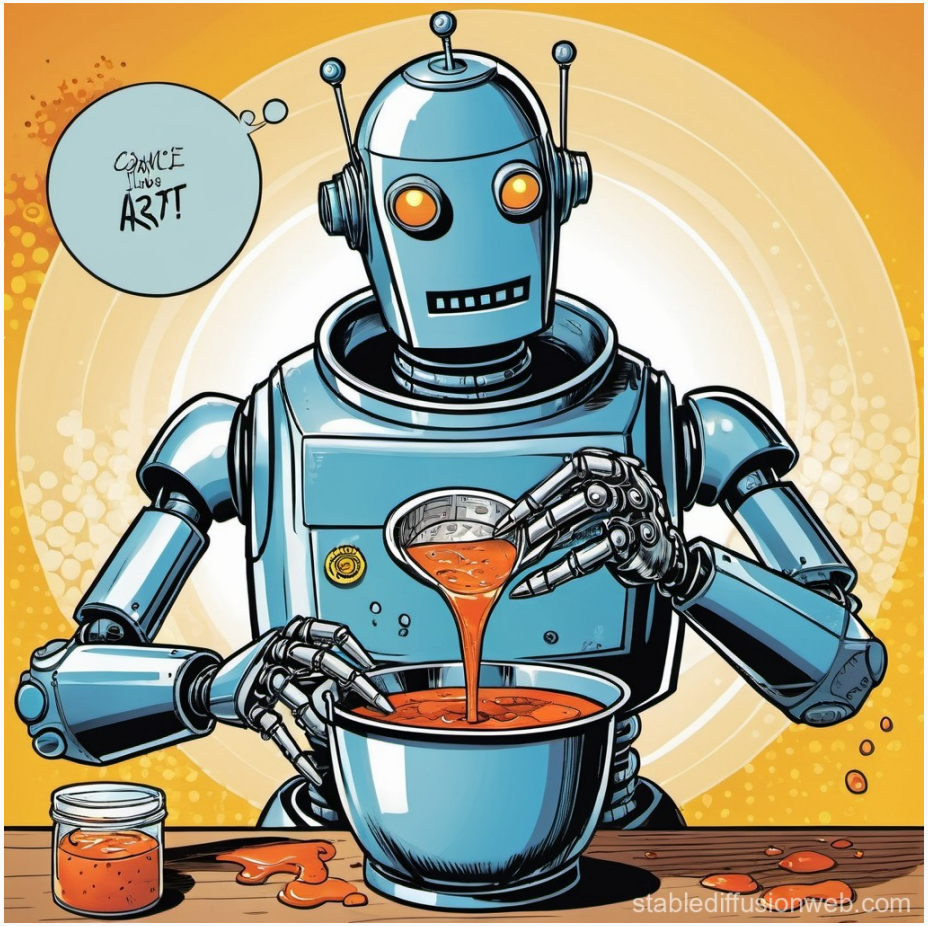


Tractability?

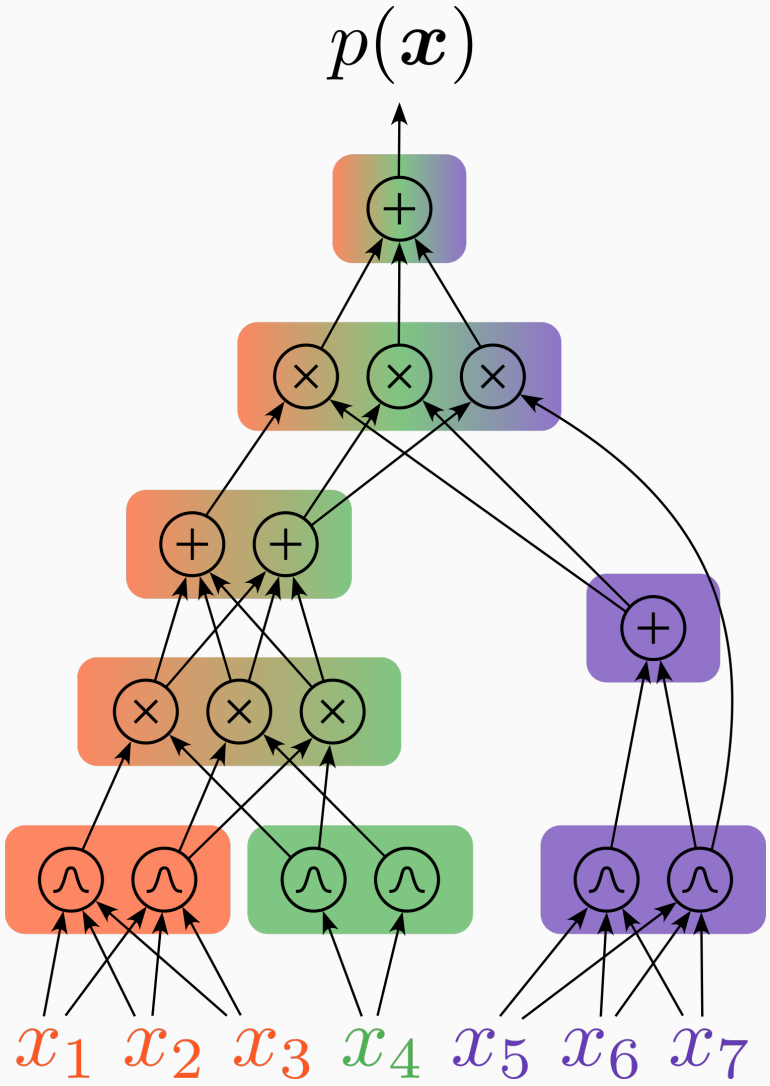
PCs, as discussed so far, do define some (potentially unnormalized) distribution $p(\boldsymbol{X})$.

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.

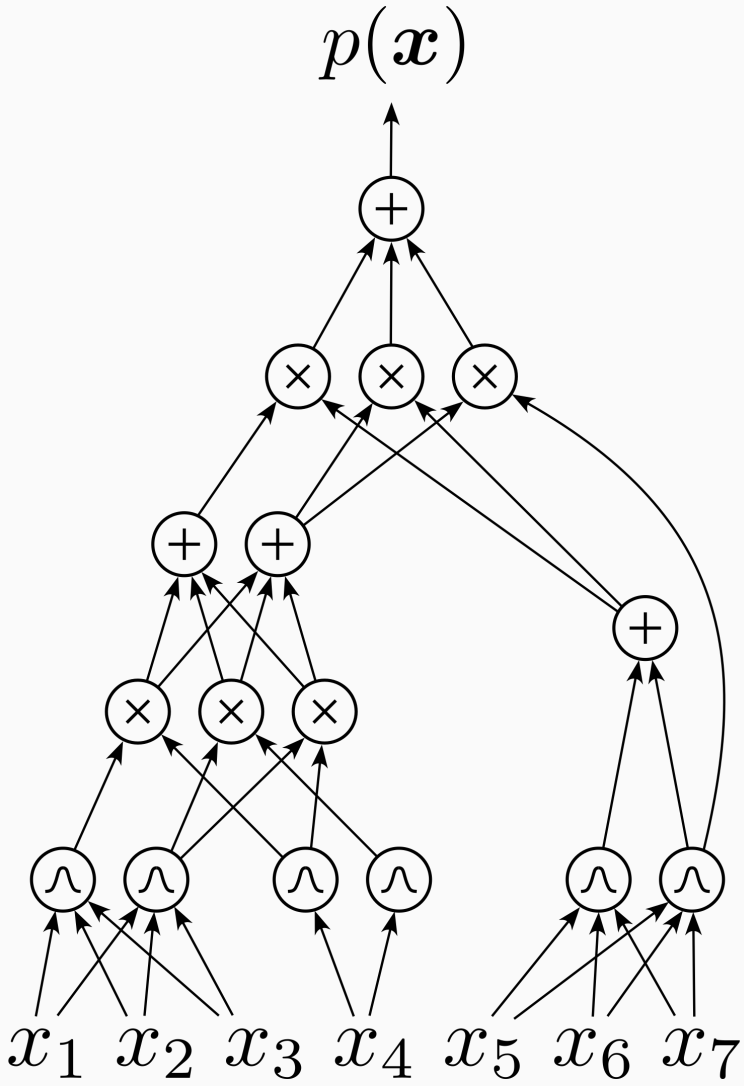
Test Time

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

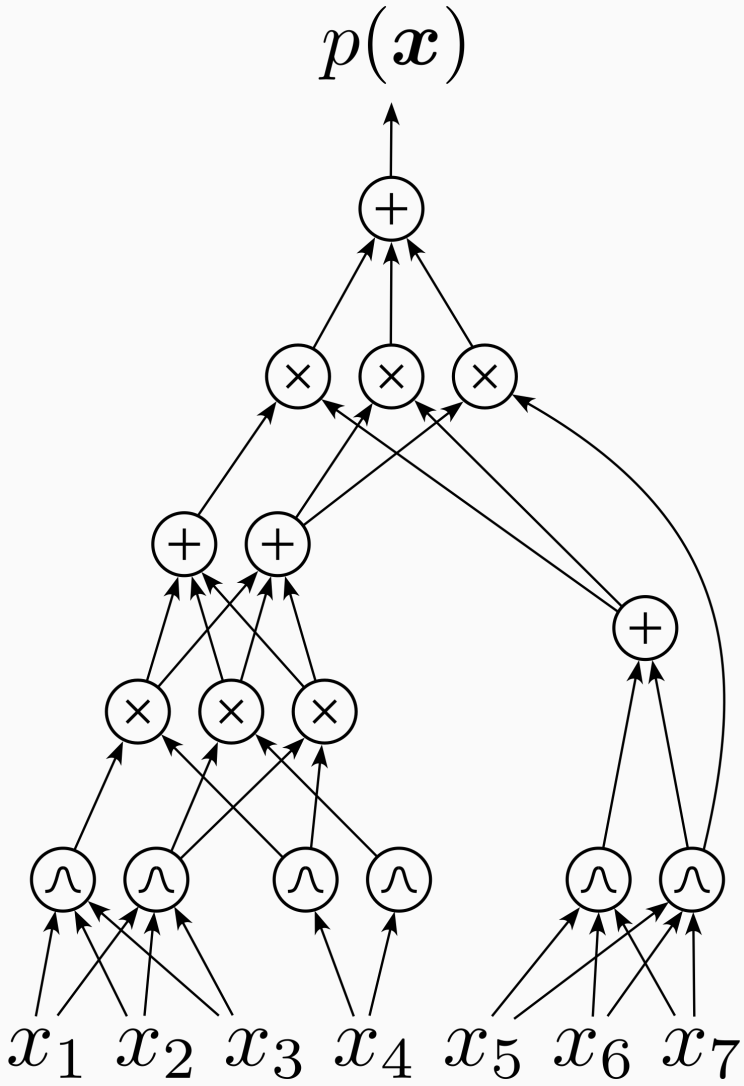
Is this PC smooth?

(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



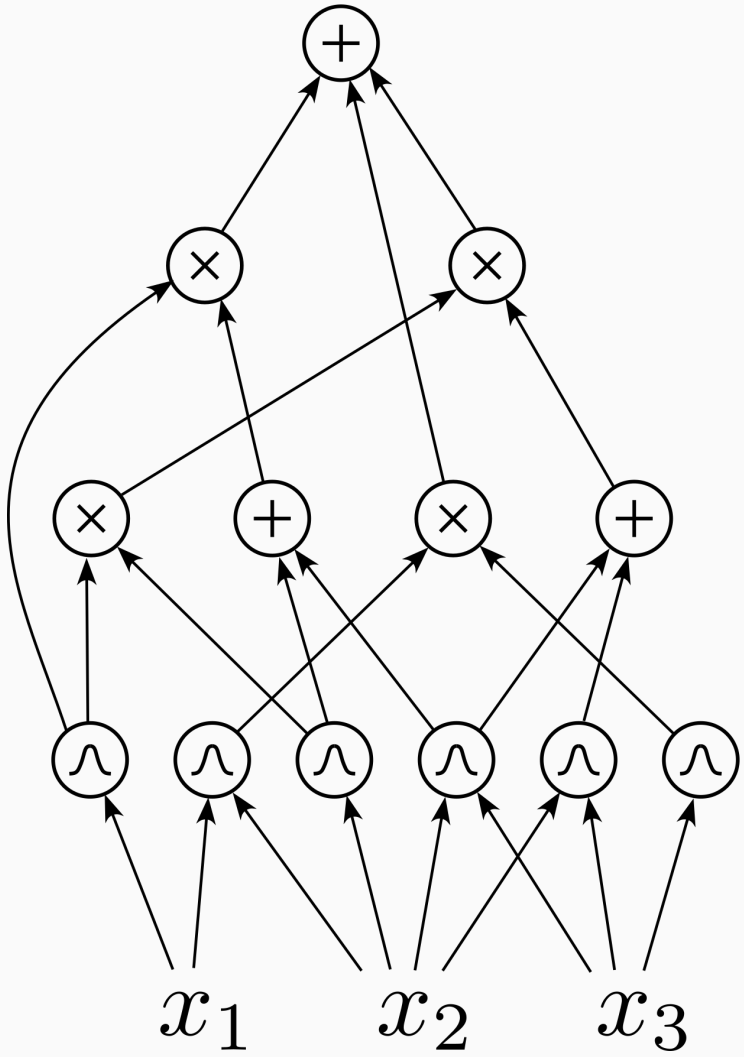
Test Time

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

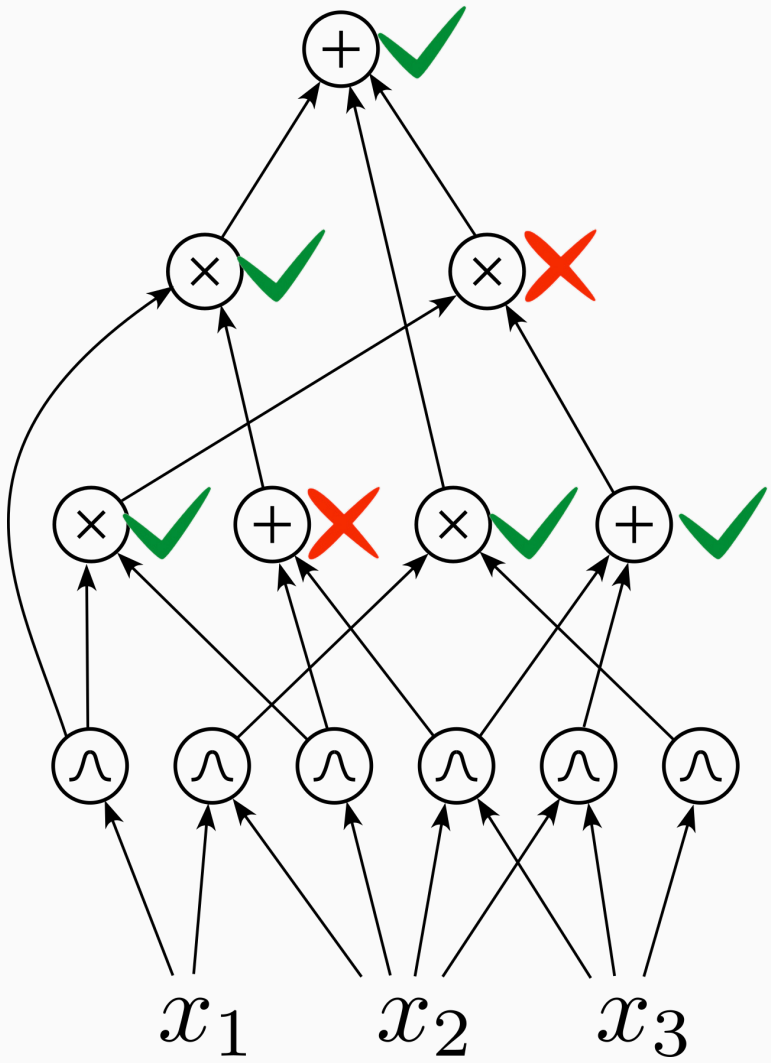
Is this PC smooth?

(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

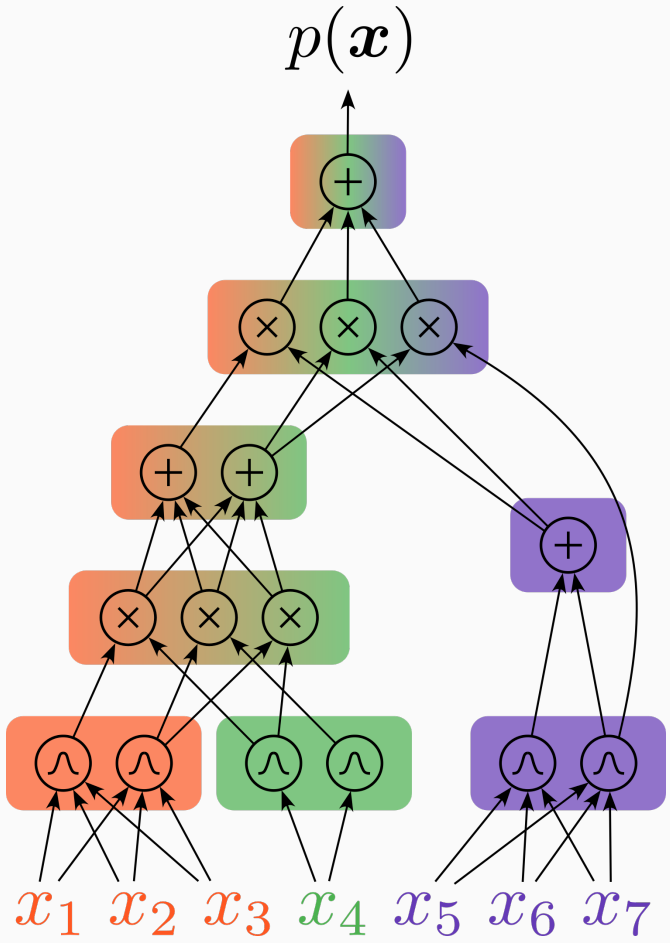
of the distributions represented by the N_k's.

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:

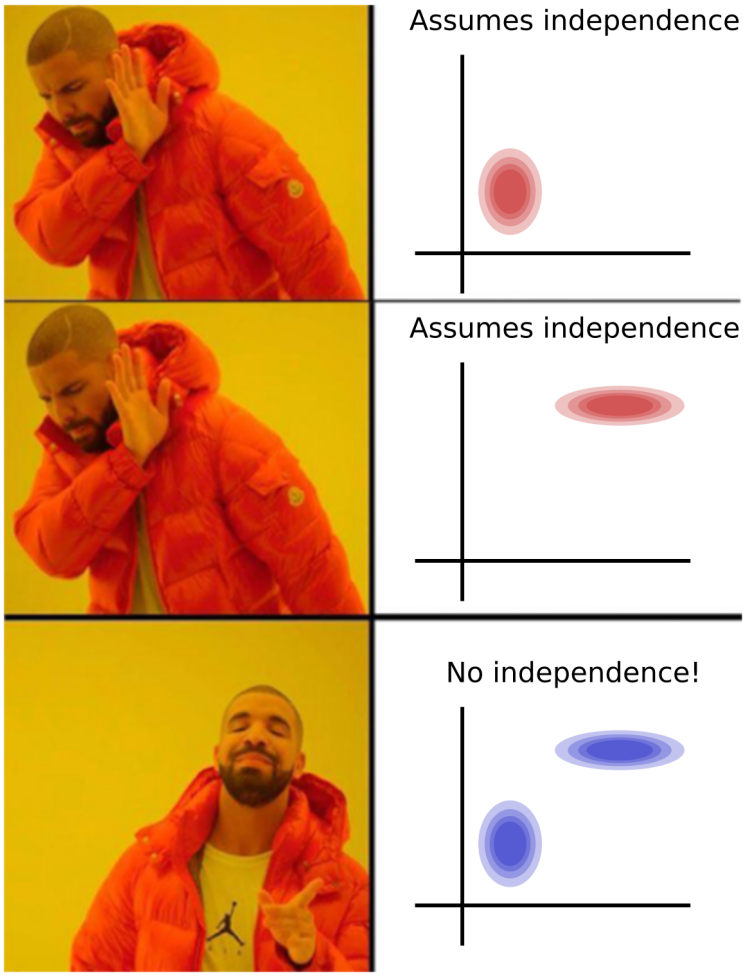
- Distribution nodes (induction basis): by definition
- Products (induction step): the decomposable product of normalized distributions is again a normalized distribution.
- Sums (induction step): a mixture of normalized distributions is again a normalized distribution.



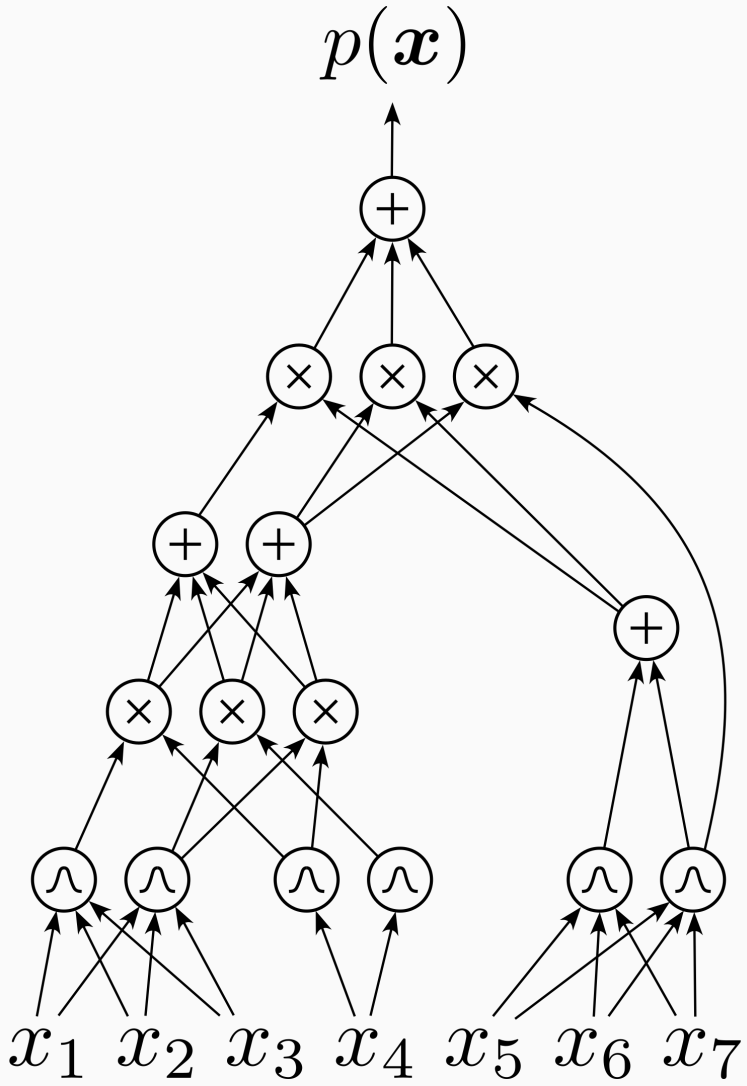
Common Pitfall 3

"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**.

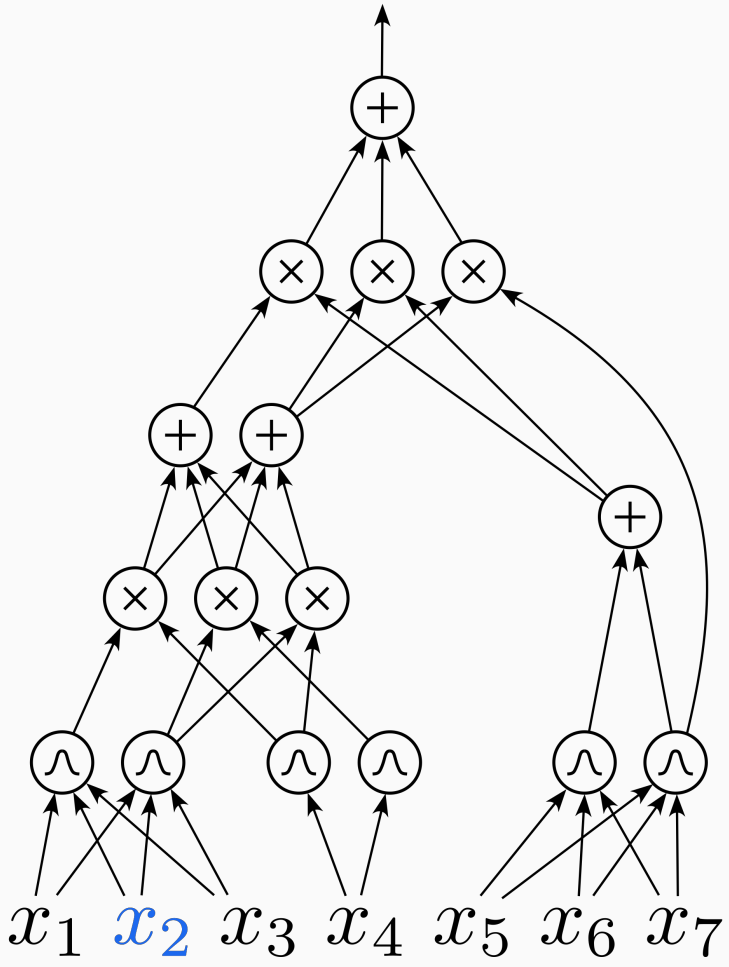
Key requirement: all input distributions allow marginalization and conditioning (e.g. Gaussians). A smooth and decomposable PC essentially inherits this property from its input distributions.

Marginal of PC

Example

- say we want to marginalize X_2

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

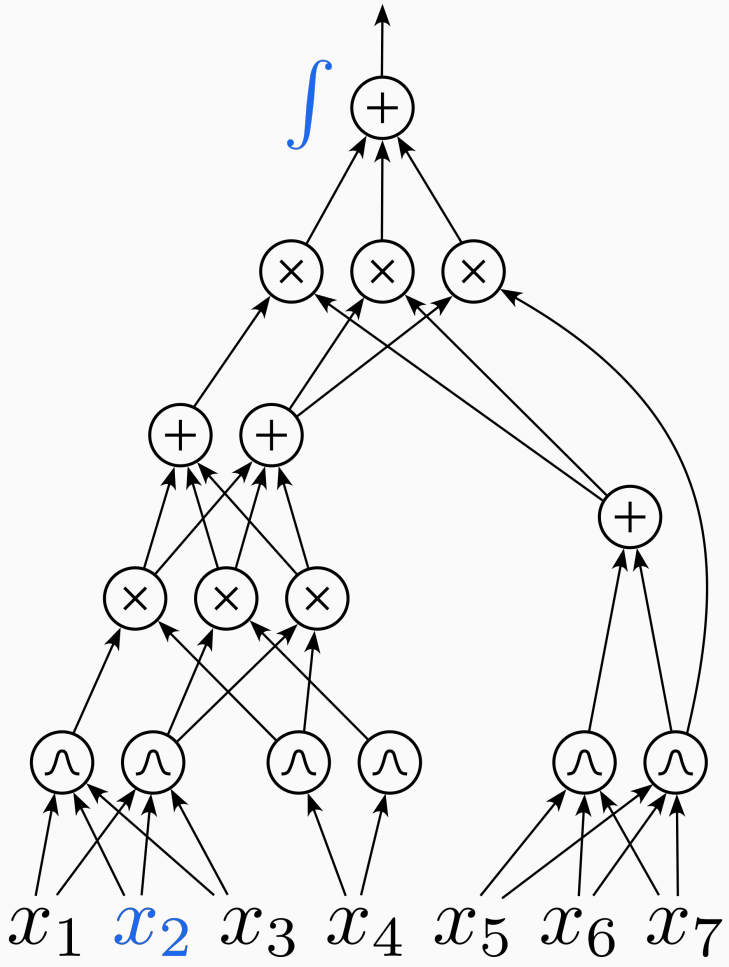


Marginal of PC

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

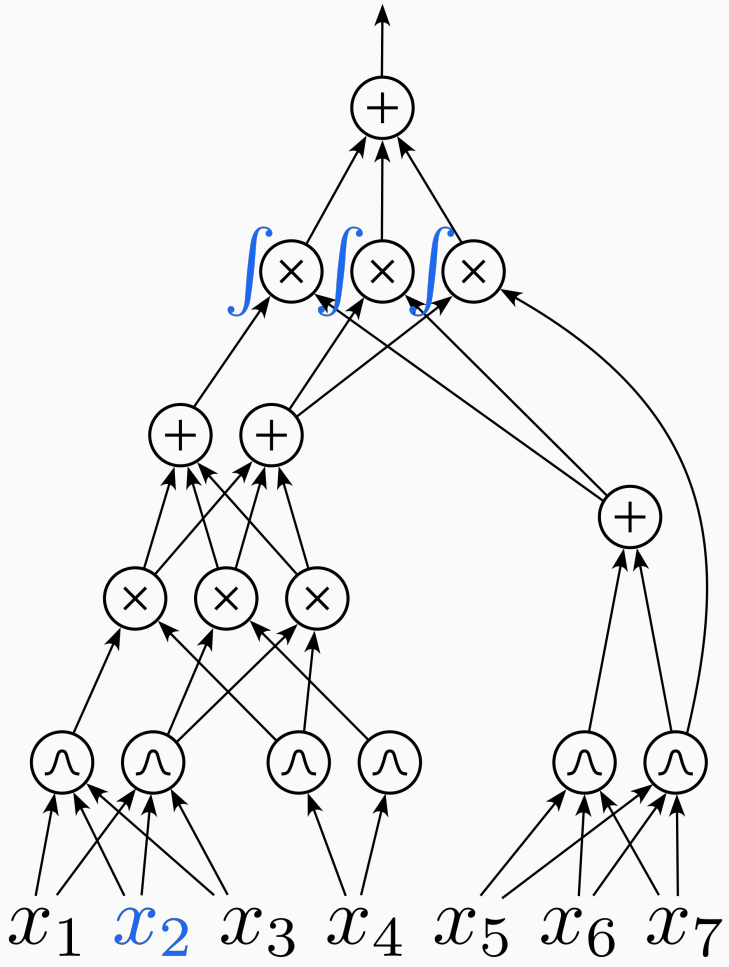
Example

- 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

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



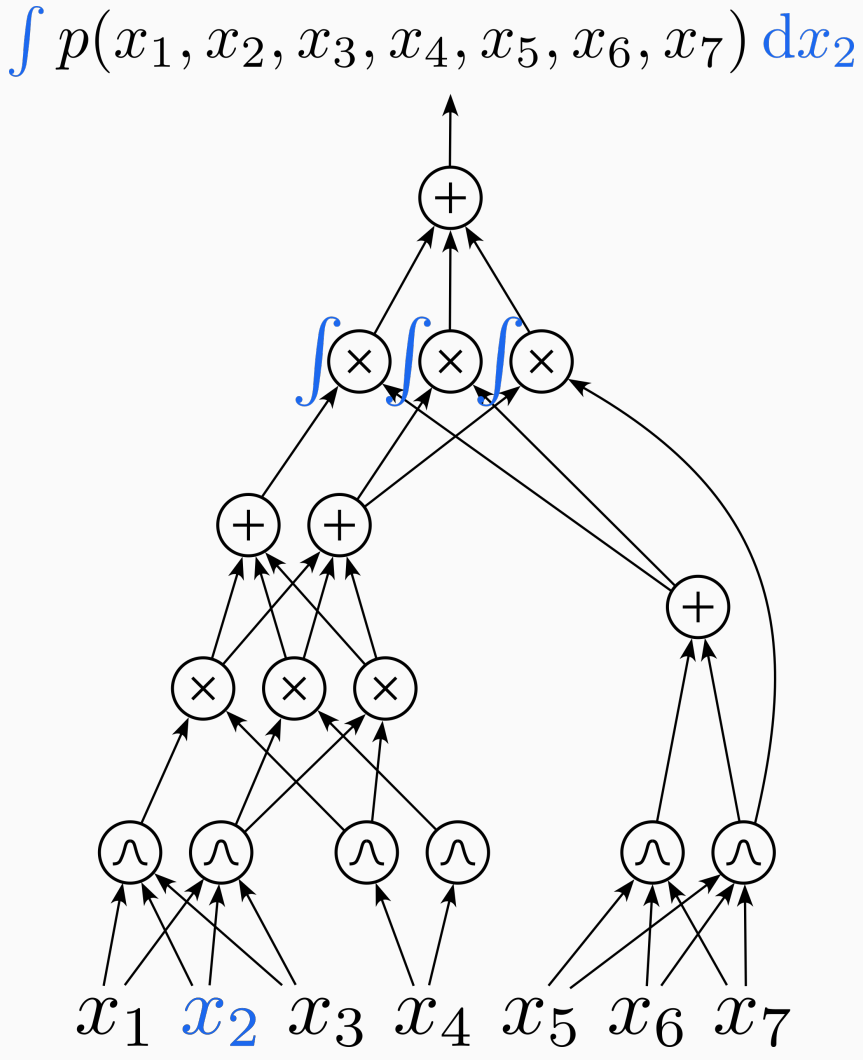
Marginal of PC

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)



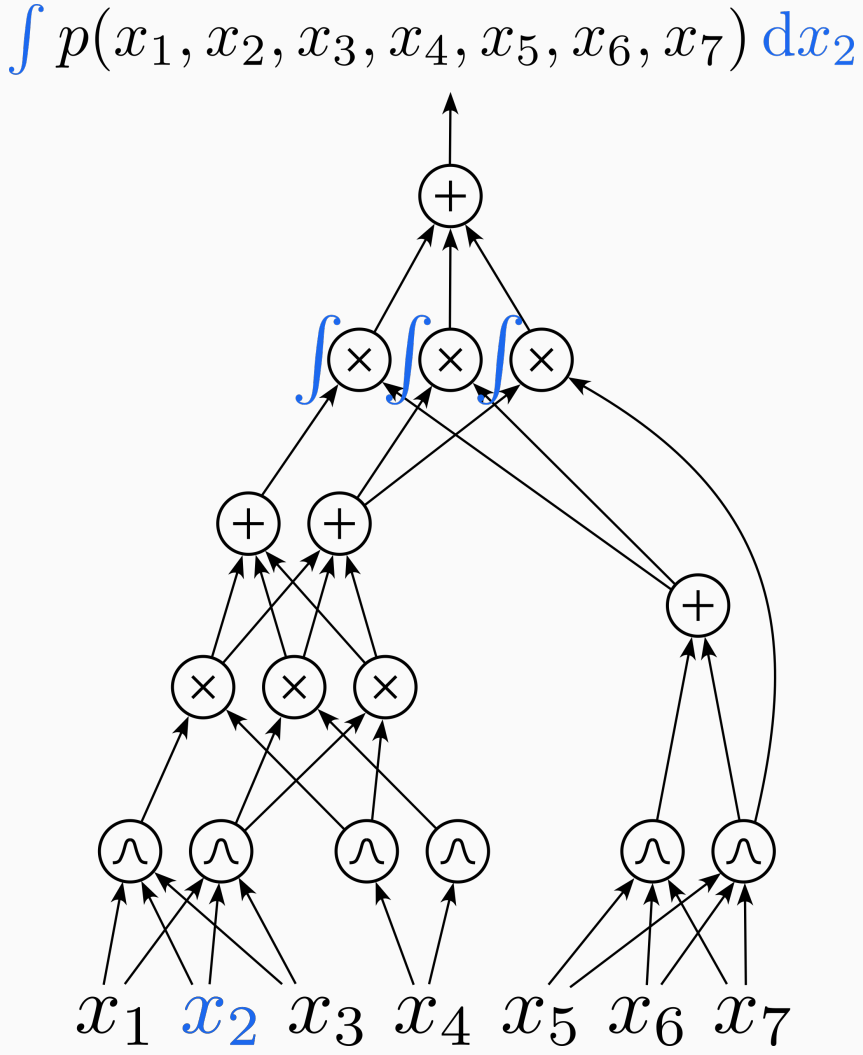
Marginal of PC

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:

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

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



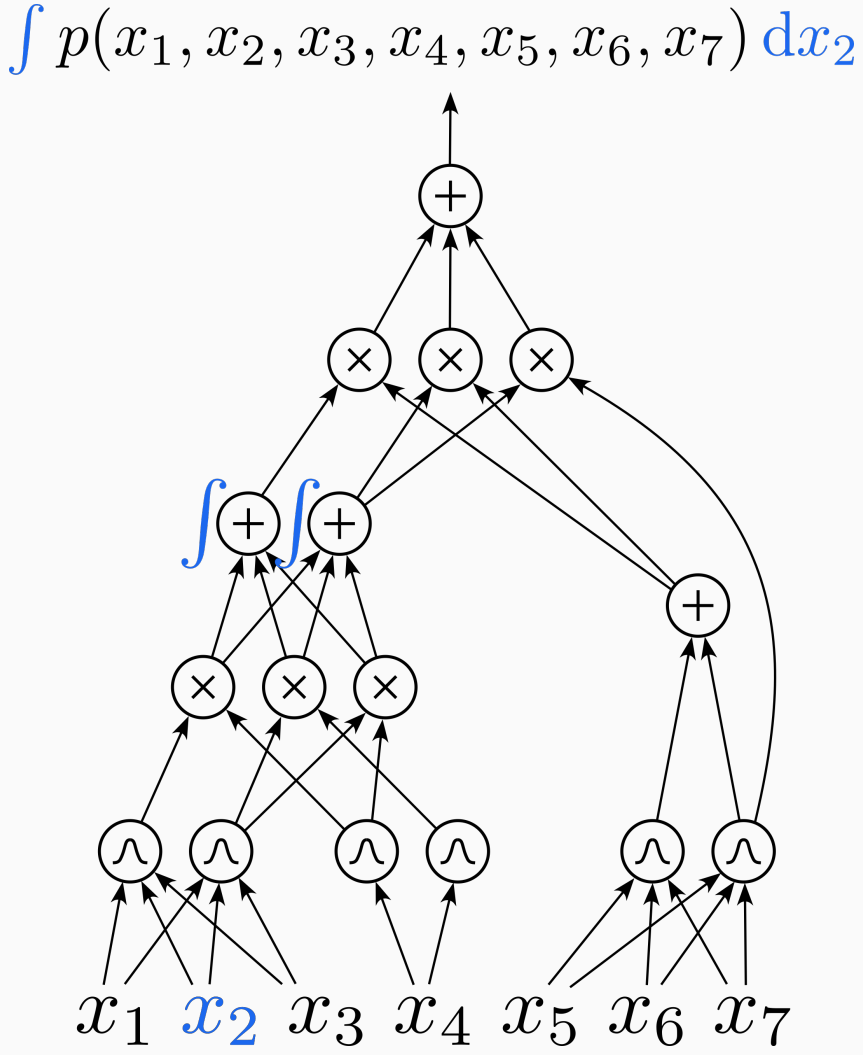
Marginal of PC

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:

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

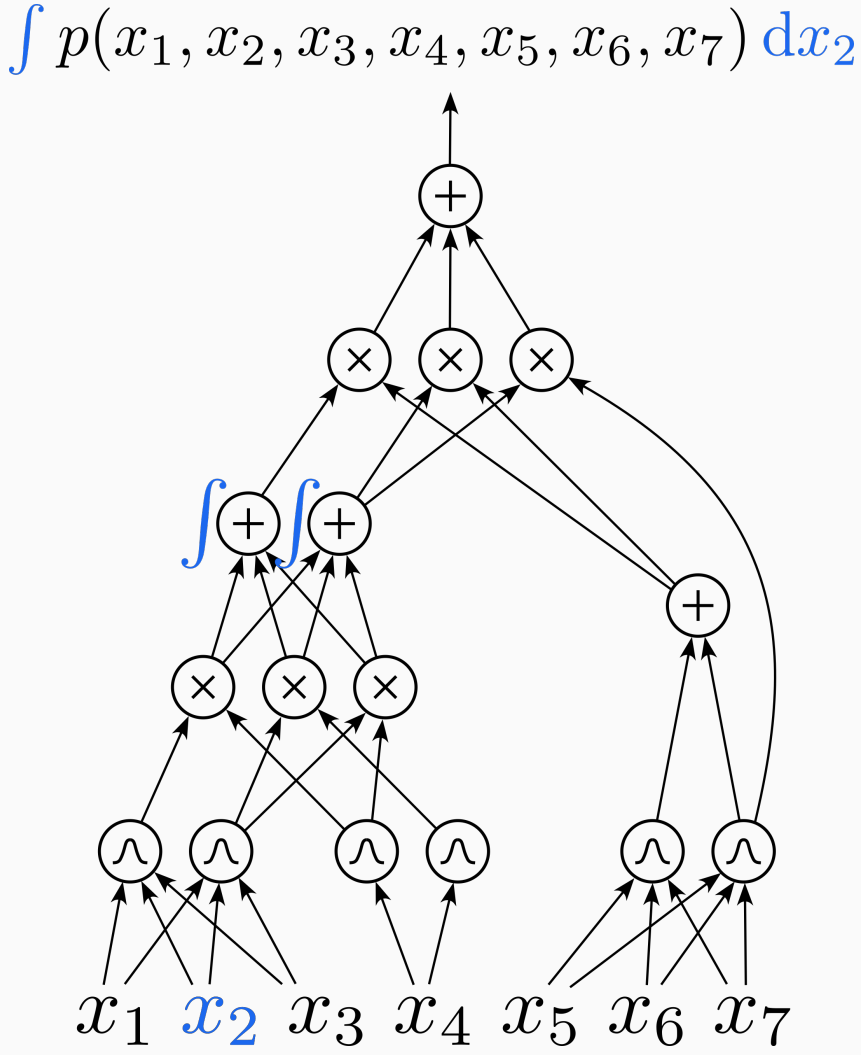
- hence, we can push the integral over each product, to the respective node containing X_2



Marginal of PC

Example

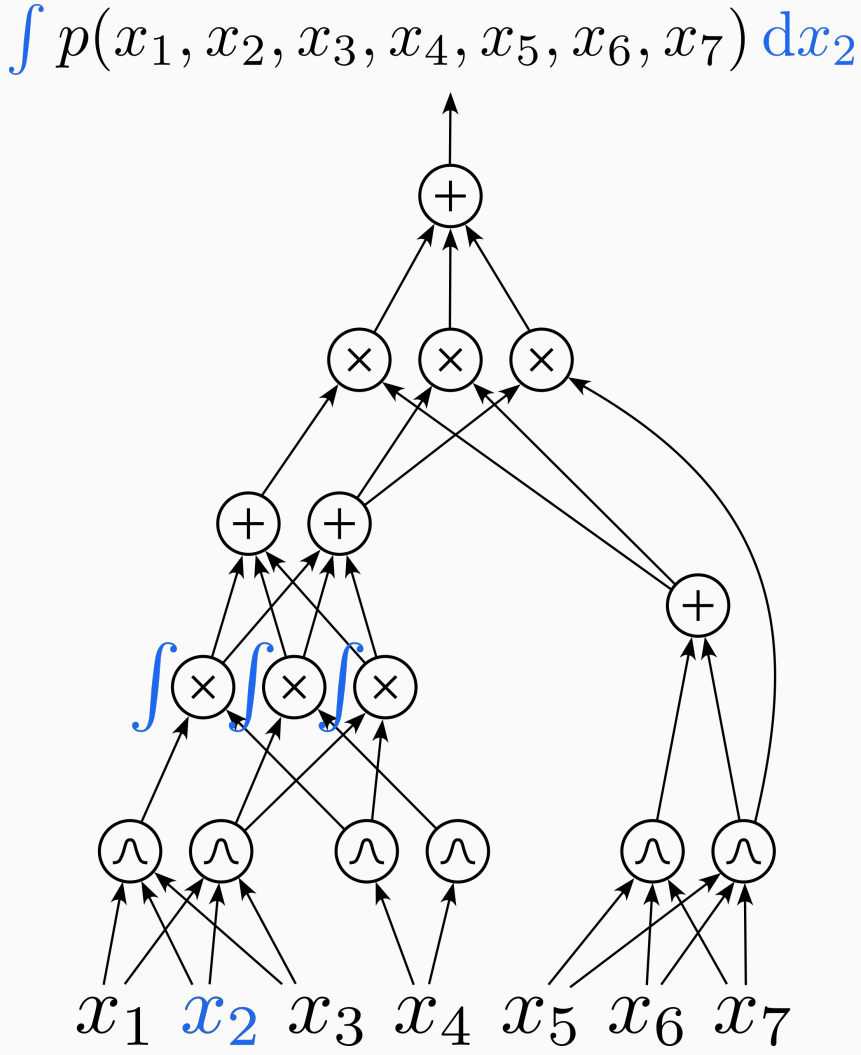
- these are sum nodes again



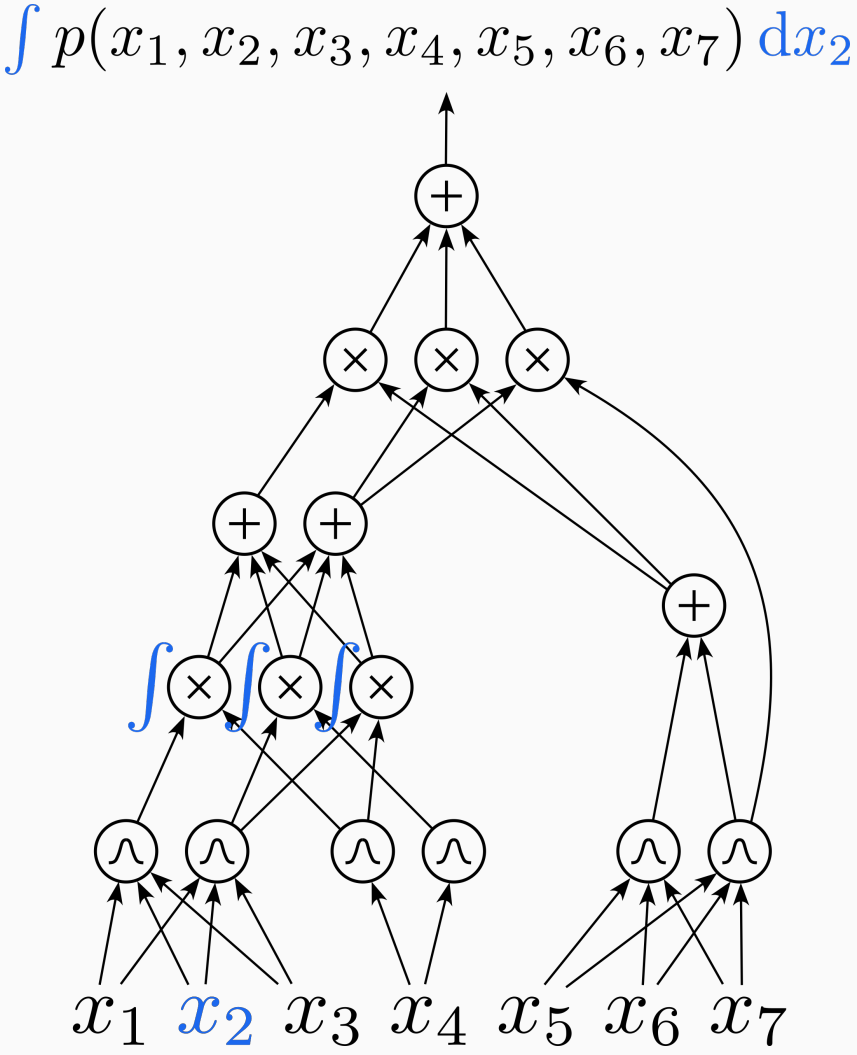
Marginal of PC

Example

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



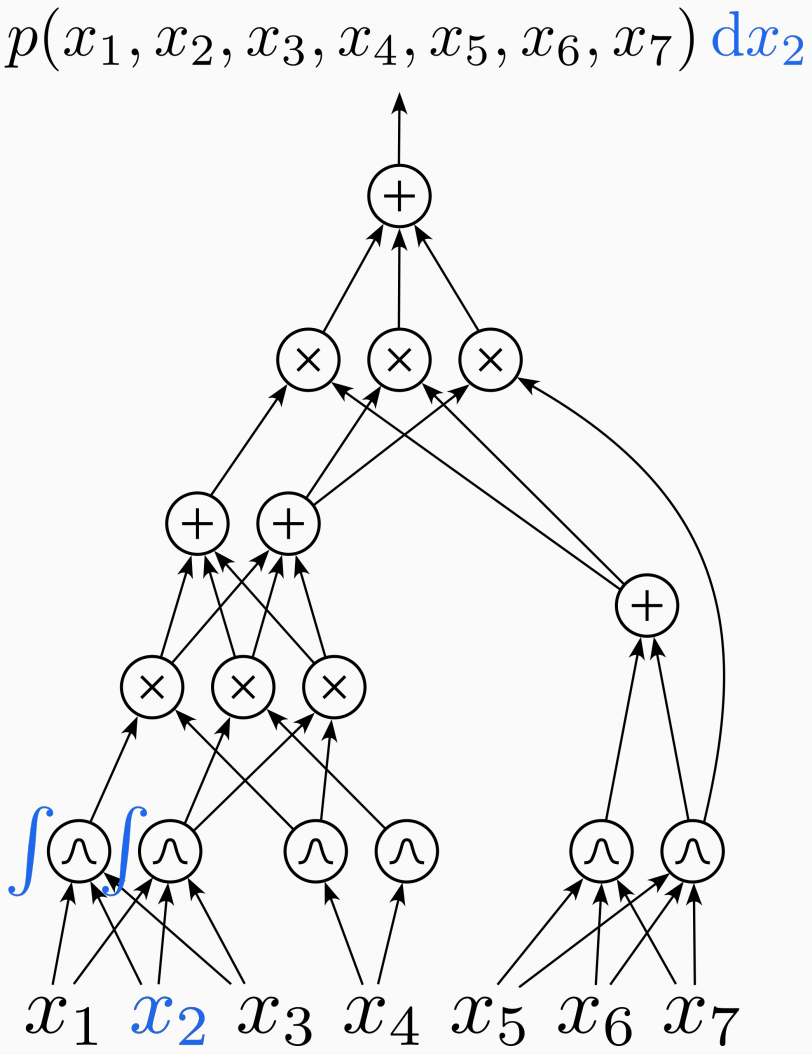
- decomposable products again



Marginal of PC

Example

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

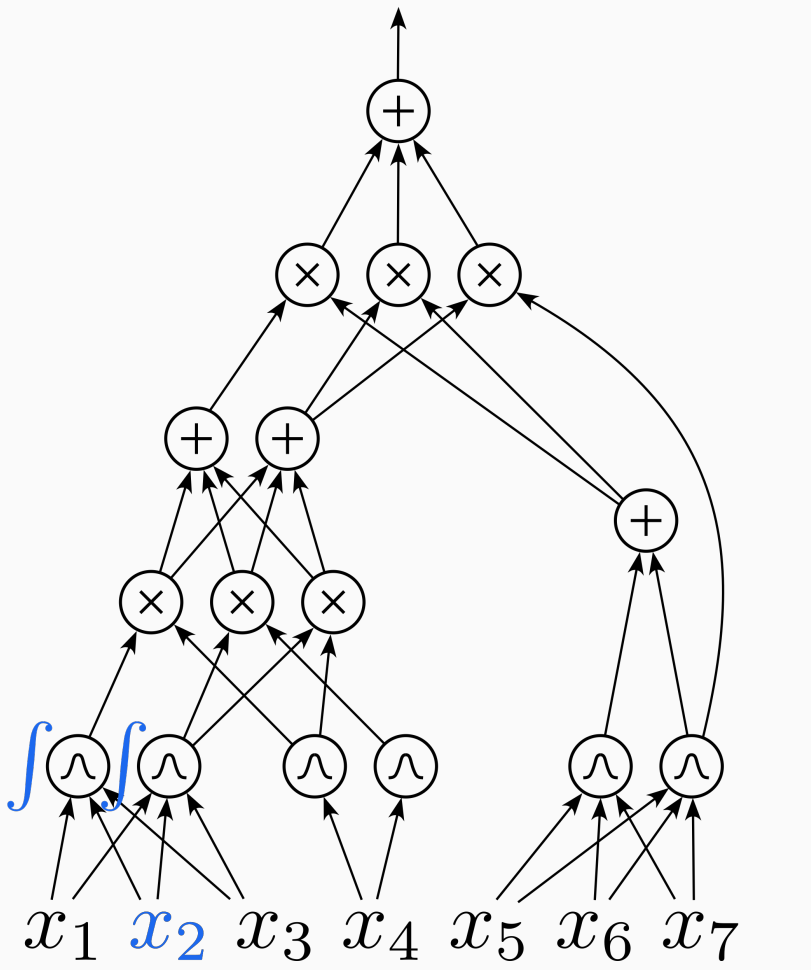


Marginal of PC

Example

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

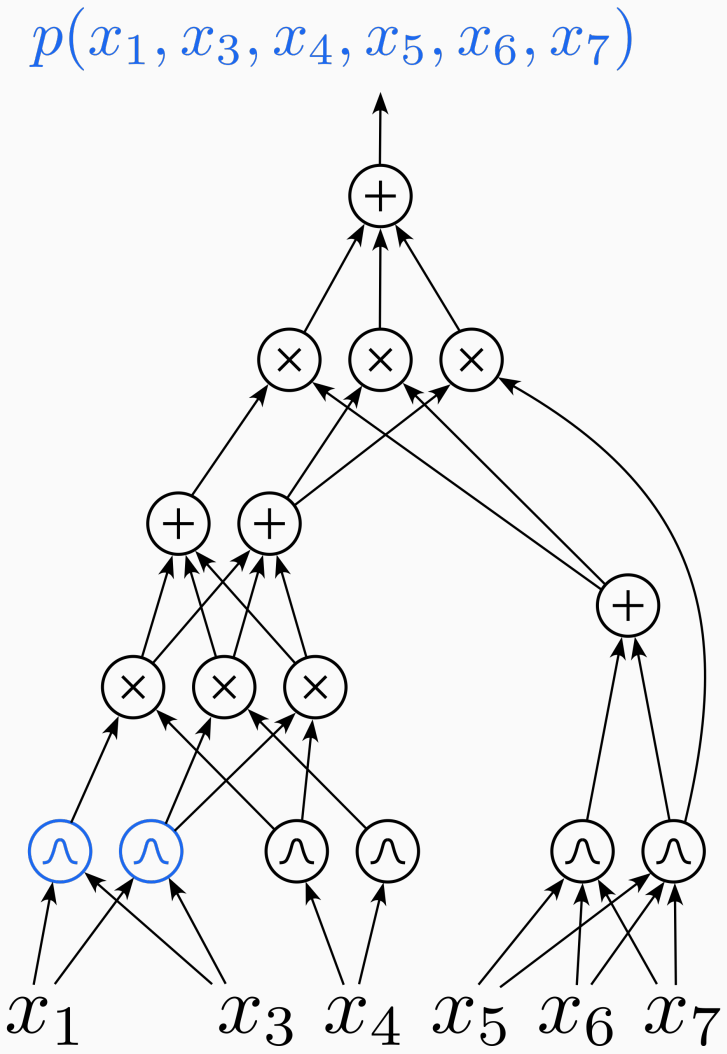
$$\int p(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\, \mathrm{d}x_2$$



Marginal of PC

Example

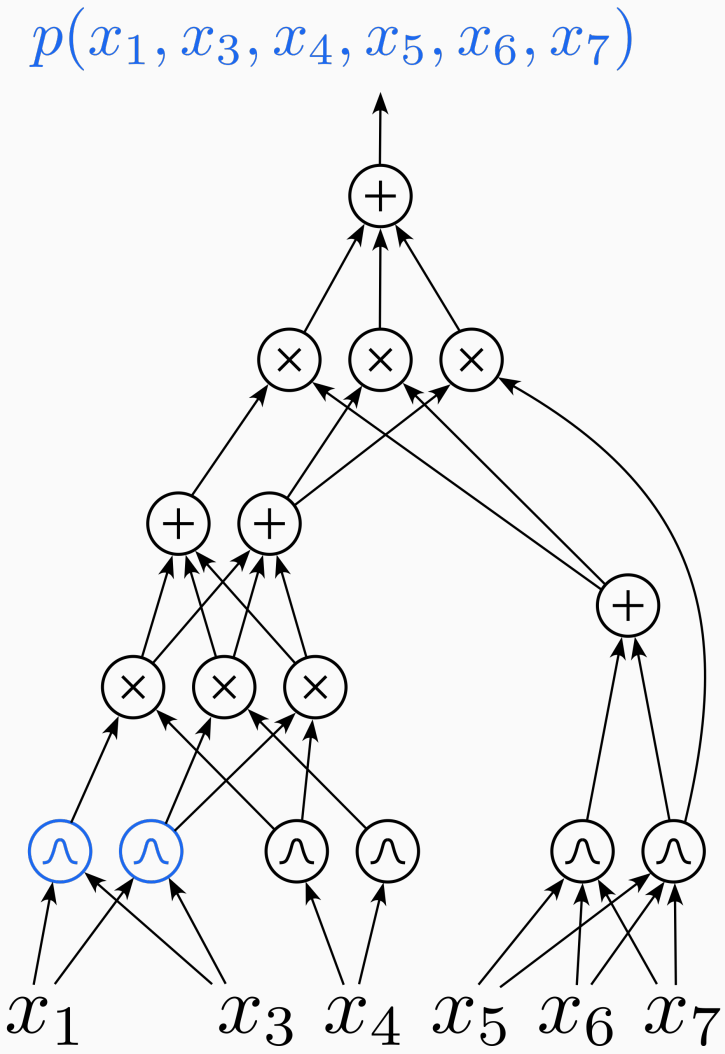
- now, the integrals are at input distributions
- but we assumed that they allow tractable marginals!
- thus, just replace the input distributions containing X_2 with the marginals where X_2 is marginalized out
- this yields a new PC, representing the desired marginal $p(X_1, X_3, X_4, X_5, X_6, X_7)$



Marginal of PC

Example

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



Marginal PC

Let a PC over $\boldsymbol{X}$ be given and let $\boldsymbol{X}_m \subseteq \boldsymbol{X}$ be variables to be marginalized. For each input distribution D, replace D with the marginal distribution, with $\boldsymbol{X}_m \cap sc(D)$ marginalized out (this is 1 if $sc(D) \subset \boldsymbol{X}_m$). The resulting PC is called **marginal PC** w.r.t. $\boldsymbol{X}_m$.

- the marginal PC represents the **exact** marginal distribution of the original PC
- actually, it represents the marginals for **each** node, not only the root
- for implementation, we don't need to spawn the marginal PC explicitly in memory, but can just "tell" the input distributions which RVs are marginalized

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

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

Conditional PC

Let a PC over $\boldsymbol{X}$ be given and let $\boldsymbol{X}_q \subseteq \boldsymbol{X}$ be query variables and $\boldsymbol{X}_e = \boldsymbol{X} \setminus \boldsymbol{X}_q$.

- for each sum with weights w_k and input nodes p_k, replace the weights with

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

  where $p_k(\boldsymbol{x}_e)$ has been computed by the marginal PC.
- replace each input distribution D with the corresponding conditional, with query variables $\boldsymbol{X}_q \cap sc(D)$ and evidence variables $\boldsymbol{X}_e \cap sc(D)$.

The conditional PC represents the **exact** conditional $p(\boldsymbol{X}_q \mid \boldsymbol{x}_e)$ of the original PC!

- **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