Probabilistic Circuits

Anabel Yong

10th December 2025

Probabilistic Circuits (PCs) are a family of tractable generative models that unify ideas from graphical models, deep learning, and arithmetic circuits. They provide exact, efficient probabilistic inference while remaining expressive enough to model complex, high-dimensional data. PCs achieve this by imposing structural constraints on computation graphs made of sum, product, and distribution (leaf) nodes.

1 Tractable vs. Intractable Models

The main difference between different probabilistic models like (Fully factorized, NaiveBayes, AndOrGraphs, PDGs, Thin Junction Trees, ACs, DPPs, Mixtures) and (NADES, MADES, MAFs, VAEs, FVSBNs, TACs, IAFs, NAFs, RAEs, BNs, NICE< FGs, GANs, RealNVP, MNs) are whether they are tractable for answering certain queries. Tractability is a spectrum and it depends on the queries you want to ask. Ensuring a model is tractable will be asking for expressive models without compromise; this means if you want your model to be tractable, you are making a compromise on its expressivity. The goal of the tutorial is to provide a unifying framework for tractable probabilistic models.

2 Why Tractable Inference? (expressiveness vs tractability)

We want to fit a generative model, a model which encodes a joint probability distribution, We want to fit a generative model to answer probabilistic queries.

Q_1 = What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd?

$$X = \{\text{day,time}, \text{Jam}_{Street1}, \text{Jam}_{Street2}, \ldots, \text{Jam}_{StreetN}\}$$

$$q_1(m) = p_m(\text{Day} = \text{Mon}, , \text{Jam}_{Westwood} = 1)$$

We want to marginalize out the rest of the variables. We can formalize probabilistic queries as computing some quantity over the probability distribution which is encoded in our generative model.

Tractable Probabilistic Inference: A class of queries $\mathcal{Q}$ is tractable on a

family of probabilistic models $\mathcal{M}$ if and only if for any query $q \in \mathcal{Q}$ and model $m \in \mathcal{M}$ exactly computing $q(m)$ runs in time $\mathcal{O}(\text{poly}(|m|))$. We have a family of probabilistic models. Each model $m \in \mathcal{M}$ is like a particular Bayesian network, factor graph, etc. We have a class of queries $\mathcal{Q}$. A query $q \in \mathcal{Q}$ might be things like computing a marginal probability, MAP, partition function, etc. The class of queries is tractable for this model family if for any model m in the family and any query q, the answer $q(m)$ can be computed exactly in time polynomial in $|m|$. Here $|m|$ is the size of the model (number of parameters, factors, etc.). We also need to note that often the polynomial time is actually linear in the size of the model. If the model size $|m|$ itself only grows polynomially with the number of variables X, then query time is also polynomial in the number of variables.

Q_1 = What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood

$$X = \{\text{day,time}, \text{Jam}_{Street1}, \text{Jam}_{Street2}, \ldots, \text{Jam}_{StreetN}\}$$
$$q_3(m) = p_m(X = \text{Mon}, 12.00, \text{Jam}_{Westwood} = 1)$$

q_3 is a complete-evidence query which means "What is the probability of this exact full assignment of all variables?" In maximum likelihood estimation, training data contains many such full assignments x. To fit parameters θ, you maximize the probability of all observed data. $\theta_m^{MLE} = \arg\max_\theta = \Pi_{x \in \mathcal{D}} p + m(x; \theta)$. So computing $p_m(x)$ for complete evidence like in the query is exactly the operation needed for MLE. Which kind of probabilistic model are we allowed to use to answer complete evidence queries? We use Generative Adversarial Networks.

2.1 Generative Adversarial Networks

We cannot use GANs to answer complete evidence queries because:

1. no explicit likelihood; adversarial training instead of MLE, no tractable EVI.
2. good sample quality but lots of samples needed for MC
3. unstable training (mode collapse)

A complete-evidence query asks for $p(X = x)$ which is the probability(density) of a specific full datapoint. This requires a model that explicitly defines a likelihood function $p_\theta(x)$. GANs specify only a generator function $x = G_\theta(z)$ and a discriminator $D_\phi(x)$, but these are just functions, not probability distributions. A GAN never defines a formula for $p_\theta(x)$. GAN training is adversarial, not likelihood-based. GANs optimize:

$$\min_\theta \max_\phi \mathbb{E}_{x \sim p_{\text{data}}} \log D_\phi(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_\phi(G_\theta(z))$$

No term involves $p_\theta(x)$. The generator is only trained to produce samples that fool the discriminator, not to represent a probability model. GANs have no tractable explicit probability distribution to query. GANs can produce data points x via simulation but they cannot tell how likely that data point is, how its likelihood compares to others, or the probability of a specific configuration.

2.2 Variational Autoencoders

While VAEs do define an explicit likelihood model:

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz$$

In principle, the VAE does assign a probability to any full datapoint x but the integral is the problem. Computing p_θ requires solving an intractable integral. The expression is over a continuous latent space z, which is high-dimensional and forming an uncountable mixture of infinitely many components. Therefore, VAEs must optimize the ELBO instead. You maximize a lower bound instead of the true log-likelihood:

$$\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p(z))$$

2.3 Normalizing Flows

Normalizing flows are the only one that can compute exact likelihoods, and therefore they can answer complete evidence queries(EVI). A normalizing flow defines a bijective invertible transformation between latent variable $z \sim p_Z(z)$(simple distribution like a Gaussian), and observed variable $x = f(x)$. Because f is invertible, we can solve for $z = f^{-1}(x)$. Then, the change-of-variables formula gives an exact density:

$$p_X(x) = p_Z(f^{-1}(x)) \det \frac{\delta f^{-1}}{\delta x}$$

This is fully tractable because f^{-1} is computable. The Jacobian determinant is designed to be cheap to compute and $p_Z(z)$ is known exactly. Evaluating the probability of a particular data point x is easy. They can be used to answer complete-evidence queries because they provide an explicit, tractable formula for the exact likelihood $p(x)$.

3 Marginal Queries

Q_1 = What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd?

A marginal query asks $p_m(E = e)$ where E is a subset of the variables and nothing is said about the others. Here, we are not fixing all the variables. You leave the rest (call them H) unknown or hidden. To compute the marginal, you must sum or integrate over all possible values of the remaining variables.

$$p_m(e) = \int p_m(e, H)dH$$

where $E \subset X$ are the variables you observe. and $H = X/E$ are the hidden ones you must sum/integrate out. So a marginal probability involves an expensive summation or integral over many configurations.

4 Probabilistic Graphical Models

PGMs separate the modeling assumptions from the inference algorithms. This means, the graph formalism structure tells you how variables depend on each other(this is the model - the assumptions about the world). Inference algorithms compute probabilities using that structure (this is reasoning - what questions we ask the model). For example, the graph in the slide shows variables $\{X_1, X_2, X_3, X_4, X_5\}$ with edges connecting them. Nodes are random variables and edges are dependencies (who influences whom or who shares information). This indicates **declarative semantics**, nothing about learning or inference is baked into the graph. It only expresses how the world works at a structural level. Once the graph is defined, you can ask questions like:

1. $p(X_1 = x_1 | X_3 = x_3)$
2. $p(X_2, X_4)$
3. $\max_{x_5} p(x_5|\text{evidence})$

To answer the above, PGMs give several general purpose inference methods. Conditioning (which is setting some variables to fixed values, reduce the model and compute the rest). Variable elimination (systematically sum/integrate out irrelevant variables while using the graph's structure to remain efficient). Message passing (belief propagation) - nodes send probability messages to neighbours, used in trees, HMMs, etc. These algorithms exploit conditional independencies encoded in the graph to avoid brute-force computation. This is different from neural models where inference is tied to the actual architecture of the model.

4.1 Complexity of Marginal Queries on PGMs

Computing marginal and complete queries is #P-hard. This means to compute a marginal probability $p(X = x)$ or $P(E = e)$, you must sum over exponentially many hidden-variable assignments. This summation is as hard as counting the number of satisfying assignmnets to a CNF formula - a classic #P-hard problem. Exact marginal inference is computationally intractable for general PGMs. Additionally, approximating MAR and CON within a factor of $2^{n^{1-\epsilon}}$ is NP-hard. This means exact computation is intractable and approximate computation with any reasonable error guarantee is also intractable. A marginal query $p(e) = \sum_H p(e, H)$ requires summing over all possible combinations of the hidden variables H. If there are n hidden variables, there are 2^n assignments.

4.2 Why is it intractable? Treewidth.

Treewidth is how close the graphical model is to being a tree. If the graph were a tree, then inference is easy in linear time. If the graph has lots of loop and dense connections, inference is hard. Formally, treewidth is the minimum size of the largest "cluster"(minus 1) required to turn the graph into a tree structure (a tree decomposition). For a fixed treewidth w, inference scales linearly with

the number of variables $\mathcal{O}(|X| \times 2^w)$. So if you can design a model so that w is small, MAR and CON become tractable. What about bounding the treewith by design? This hints at model classes that intentionally restrict the structure so that inference is always efficient(e.g. arithmetic circuits).

4.3 Tree-structured Bayesian Network

A tree-structured BN is a graphical model where each variable X_i has at most one parent. The entire structure forms a tree - no loops, no nodes with multiple parents. If our model is a tree, the joint distribution factorizes as:

$$p(X) = \Pi_{i=1}^{n} p(x_i | P_{a_{x_i}})$$

This factorization is exact and extremely efficient. We have three kinds of probabilistic queries, EVI(complete evidence queries), MAR(marginal queries), and CON(conditional queries). For general graphical models, these are #P-hard or NP-hard. But for trees, all these queries can be computed in linear time $O(|X|)$. This time required grows only proportionally to the number of variables. As trees have treedwidth of 1, complexity of inference is $O(|X| \times 2^w)$. Therefore, $O(|X| \times 2) = O(|X|)$. But, we lose expressiveness - ability to represent rich and complex classes of distributions. Is there a way to solve this? Yes, using **mixtures as a convex combination of k simpler probabilistic models**.

4.4 Mixture Models

A mixture model takes several simple models and blends them teogether.

$$p(X) = w_1 p_1(X) + w_2 p_2(X) + \cdots + w_k p_k(X)$$

where each $p_i(X)$ is a simple model like a tree distribution. $w_i \geq 0$ and the summation of w_i sums to 1. This is a convex combination of simpler distributions. EVI, MAR, CON queries scale linearly in k. This is because evaluating mixtures is just evaluating each component and summing EVI ($p(x) = \sum_i w_i p_i(x)$), MAR ($p(e) = \sum_i w_i p_i(e)$) and CON also decomposes across components. So the cost increases by a factor of k(number of components). Inference stays tractable because queries reduce to weighted sum over components.
A mixture model means marginalizing a categorical latent variable.

$$p(X) = p(Z = 1)p_1(X|Z = 1) + p(Z = 2)p_2(X|Z = 2)$$

This indicates a mixture is just a probabilistic model with a discrete latent vairable Z. Z chooses which component model generates the data. If $Z = 1$, use the distribution $p_1(X)$, if $Z = 2$, use the distribution $p_2(X)$. The final separation is obtained by marginalizing out Z:

$$p(X) = \sum_{z=1}^{k} p(Z = z)p(X|Z = z)$$

5 Maximum A Posteriori(MAP)

Find parameters θ that are the most probable after observing data and considering prior beliefs.

$$\theta_{MAP} = \arg\max_{\theta} p(\theta|\mathcal{D})$$

MAP asks:

$$\arg\max_{j_1,j_2,\ldots} p_m(j_1, j_2, \ldots | \text{Day} = \text{Mon}, \text{Time} = 9)$$

Here the evidence is that the day is Monday, and time is 9 pm. Unknown query variables $\mathcal{Q}$ are the jam indicators for each road. So in MAP, this means among all possible combinations of jams, which one is most probable given the evidence? Given a trained model, picks the mosts likely configuration of variables under the evidence. No parameter learning happens in MAP inference.

6 Probabilistic Circuits

A probabilistic circuit is a computational graph(like a neural network), where each node computes part of a probability distribution. Leaves represent simple distributions (univariate probabilities), and internal nodes combine them using sum and product operations. The entire circuit computes a (possibly unnormalized) probability $p(X)$. Operational semantics = the exact rules for computing probabilities by running a circuit. This means at sum nodes, we compute a mixture $p = w_1p_1 + w_2p_2 + \ldots w_kp_k$. At product nodes, you compute a factorized joint $p = p(A) \cdot p(B)$. Everything in the circuit can be computed exactly by evaluating the graph bottom-up.

6.1 Distributions as computational graphs

The simplest base case is a leaf node which represents a univariate Gaussian distribution. A simple probability distribution, can be treated as a black-box computational node in a probabilistic circuit. You feed in a value x, the node outputs the probability $p_X(x)$. For example, we know a Gaussian node with mean μ, and variance σ^2 computes:

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

This is a tractable computation. Gaussians are perfect circuit building blocks. For a Gaussian, and computing Complete Evidence Inference (EVI), this is just evaluating the Gaussian PDF which is $p_X(x) = \mathcal{N}(x|\mu, \sigma^2)$. This is fast in $O(1)$ time. For marginal inference (MAR), the marginal of X is 1 (normalized) where $\int p(x)dx = 1$. If its unnormalized, the MAR operation outputs the partition function Z - the integral of an unnormalized distribution. For MAP, the mode is simply its mean $\arg\max_x p_X(x) = \mu$. Gaussians are tractable for all common inference tasks. Therefore, when we build PCs from simple nodes like Gaussians, all the higher-level inference operations remain tractable.

6.2 Factorizations

There are a few components which make models tractable and expressive. We do not want to work with simple distributions as they are not expressive. Factorization is a method, this means if you have a joint distribution over 3 random variables $p(X_1, X_2, X_3) = p(X_1) \times p(X_2) \times p(X_3)$. I can assume that it is factorized, meaning that it is the probability of the individual random variables multiplied. Factorizations are product nodes, in our case here, it is a product node over some univariate Gaussian distribution. We know that mixture models enhance richness and expressivity of the model. Modelling a mixture of Gaussians $p(X) = w_1 p_1(X) + w_2 p_2(X)$. We can write mixtures as sum nodes which has weights, as a weighted sum node over Gaussian input distributions.

6.3 Which structural constraints ensure tractability?

Decomposability is a property of factorized distributions. We make product nodes behave like valid probability distributions. Ensures that inference is efficient (no variable interactions, no exponential summations, avoids double-counting variables).
Smoothness requires that all children of a sum node must have the same scope (they must depend on the same set of variables). A sum node in a probabilistic circuit represents a mixture of distributions. To satisfy smoothness, if one child models (age, income) or someone, the other must also model age and income of someone. We use both decomposability and smoothness to compute a marginal (MAR). We want to compute:

$$\int p(x)dx$$

$$p(e) = \int p(e, h)dh$$

This requires integrating out hidden variables. Integration is normally #P-hard in graphical models. If thee model is smooth:

$$p(x) = \sum_i w_i p_i(x)$$

Then the scope of all p_i is the same, so the integral behaves nicely:

$$\int p(x)dx = \int \sum_i w_i p_i(x)dx$$

We can swap the integral and the sum:

$$\sum_i w_i \int p_i(x)dx$$

Decomposability means:

$$p_i(x) = \Pi_j p_{ij}(x_{ij})$$

and all variable subsets are disjoint. So:

$$\int p_i(x)dx = \int \Pi_j p_{ij}(x_{ij})dx$$

This becomes:

$$\Pi_j \int p_{ij}(x_{ij})dx_{ij}$$

Product of all the integration of all the probabilities for the product nodes. Recursively simplify these leaf nodes, get my answer and multiply. Compute with dynamic programming. Now to compute marginal probabilities (MAR): We want to compute $p(q|e)$. This is always computed using the definition of conditional probability:

$$p(q|e) = \frac{p(q,e)}{p(e)}$$

We compute two marginals, where $p(q,e)$ is the joint probability of query and evidence and $p(e)$ is the marginal over evidence only.

6.4 PCs on MAP inference

Finding either the probability of most likely state of all the random variables given some observations.

$$\max_q p(q|e)$$

Using Bayes rule, this is proportional to $\max_q p(q,e)$. We want to propagate a max instead of a sum or integral. The key problem is max does not distribute over sum. At a sum node, the model:

$$p(q,e) = \sum_i w_i p_i(q,e)$$

MAP would require $\max_q \sum_i w_i p_i(q,e)$. But max does not commute with sum:

$$\max_q \sum_i w_i p_i(q,e) \neq \sum_i w_i \max_q p_i(q,e)$$

MAP for latent variable models is intractable, but there is another property we can assume which is determinism.

6.5 Determinism aka selectivity

Determinism makes MAP inference tractable. A sum node is deterministic if for any complete assignment of all variables, at most one child of the sum node has a non-zero output. Equivalently, children of a deterministic sum node have disjoint support. This turns mixture behaviour into decision behaviour, each input activates exactly one branch. This is why determinsitic PCs allow bottom up MAP inference.

$$\max_x p(x) = \max_x \max_i [w_i p_i(x)]$$

As only one child is active at a time, max and sum no longer compete, the MAP flow can propagate locally. PCs allow bottom-up MAP inference.

$$\max_q p(q, e) = \sum_i w_i p_i(q, e)$$

$$\max_q p(q, e) = \max_q \max_i w_i p_i(q, e)$$

$$\max_q p(q, e) = \max_i \max_q w_i p_i(q, e)$$

Sum Product Networks are not deterministic, not exact MAP inference.

7 Learning Probabilistic Circuits

7.1 Exponential-family distributions matter in PCs

Since a PC is a computational graph with sum nodes, product nodes and leaf distributions. The leaves of the circuit are simple probability distributions. These leaves must be easy to evaluate, marginalize, differentiate and are parameter-efficient. Each leaf node represents a probability distribution over one variable. To learn a PC from data, we must fit the parameters of these leaves. EFs have analytical maximum-likelihood solutions. This indicates closed-form marginalization for $\int p(x)dx$. EFs distributions have closed or easily computable integrals. **How to learn the leaf distributions in a probabilistic circuit:** using maximum likelihood estimation (MLE) for exponential-family distributions. PCs rely heavily on exponential-family leaves because they allow easy, closed-form learning. Every exponential-family distribution can be written as:

$$p_L(x) = h(x) \exp(T(x)^\top \theta - A(\theta))$$

where $T(x)$ are sufficient statistics$(x, x^2, ..)$, θ are natural parameters (log-odds), $A(\theta)$ is the log-partition function(normalizer) and $h(x)$ is the base measure. For exponential families, maximizing the likelihood over data D yields a simple rule:

$$\phi_{ML} = \mathbb{E}_D[T(x)] = \frac{1}{|D|} \sum_{x \in D} T(x)$$

Compute the average sufficient statistics over data. So we know that, a sum node in a PC computes:

$$p(X) =_i w_i p_i(X)$$

We introduce a latent categorical variable $Z \in \{1, 2, \ldots, k\}$, the circuit represents:

$$p(X) = \sum_z p(Z = z) p(X | Z = z)$$

Thus, each child $p_i(X)$ is a component, and each weight $w_i = p(Z = i)$ is the mixing probability.

7.2 Expectation Maximization: maximum likelihood under missing data

E-step: Estimate the expected latent assignments (soft cluster). M-step: Maximize parameters (weights + leaf parameters) given these assignments. Given data X and hidden cluster assignments Z, EM maximizes the likelihood of:

$$p(X, Z; \theta)$$

We do not observe Z, so we replace it with its expectation under the current parameters:

$$\theta_{\text{new}} = \arg\max_{\theta} \mathbb{E}_{p(Z|X;\theta_{old})}[\log p(X, Z; \theta)]$$

E-step: Compute responsibilities (posterior over the latent variable). For each data point x and each mixture component i:

$$\gamma_i(x) = p(Z = i|X = x) = \frac{w_i p_i(x)}{\sum_j w_j p_j(x)}$$

These $\gamma_i(x)$ are soft assignments - how much each subcircuit is responsible for generating the data point. M-step: update the mixture weights:

$$w_i^{new} = \frac{1}{N} \sum_{x \in D} \gamma_i(x)$$

Which counts how often each component is used, and normalizes to get valid probabilities. Weighted MLE for exponential family leaves, we compute weighted sufficient statistics. For example, a Gaussian leaf:

$$\mu^{new} = \frac{\sum_x \gamma_i(x) x}{\sum_x \gamma_i(x)}$$

$$\sigma^{2new} = \frac{\sum_x \gamma_i(x)(x - \mu)^2}{\sum_x \gamma_i(x)}$$