Probabilistic Circuits

Anabel Yong

8th March 2026

1 Probabilistic Circuit for Parameter Learning

Complex data often often looks multi-modal. The multiple overlapping bell-shaped curves represent a **mixture of Gaussians** - a distribution that cannot be captured well by a single Gaussian because data has several clusters or regimes. The key modelling move behind a mixture is to introduce a **latent (hidden) choice variable** Z: for each data point x, the model first secretly chooses which component generated it (e.g. "mode 1", "mode 2" or "mode 3"), then draws x from that component's Gaussian. This statement is:

$$p(x) = \sum_z p(z)p(x|z)$$

which is a weighted sum of component distributions. Now this mixture idea can be formulated into a probabilistic circuit(PC). A PC can be drawn as a directed acyclic graph with two key operations: **sum nodes and product nodes**. A **sum node** represents the same concept as a mixture: it expresses a distribution as weighted combination of alternatives, which can be interpreted as making a latent choice about which child branch is active. A **product node**, in contrast, represents a factorization: it multiplies distributions over disjoint set of variables, corresponding to an independence decomposition. Leaves are simple distributions (such as Gaussians for continuous variables). By composing these two operations, a PC represents complicated distributions as a structured expression: "choose a regime (sum), then within that regime factorize into simpler parts (product), then evaluate simple leaf distributions.
This leads to a useful mental model: **a PC is a mixture of factorizations**. Each complete set of latent choices through the sum nodes corresponds to selecting on "explanation: of the data; those choices are fixed, the remaining computation is mostly products of simple terms. That is why circuits are often described as sums of products: the overall probability for x is an addition over alternatives, where each alternative is a product of simpler probabilities.

1.1 How do we fit the weights of sum nodes (mixture weights) from data?

The latent variable view makes learning natural: because the choices Z are hidden, we use an expectation-maximization(EM)-style idea. In the **E-step**, we compute how much each branch/component is responsible for explaining each data point (posterior "responsibilities" $p(z|x)$). In the **M-step**, we update the mixture weights to match those expected responsibilities. Now in Liu, Shao and Broeck [1], they emphasize an important modern interpretation: EM updates can be seen not just as "maximize likelihood", but as a controlled step that improves fit while discouraging the model from changing too radically in one jump. That "**do not jump too far**" idea is often expressed as a **KL divergence regularization** between the old and new distributions over the observed data and the latent choices (X, Z). We want to improve the likelihood but we also want to keep the distribution stable unless the data strongly supports a change.

1.2 EM fits PCs because sum nodes are latent variables.

Each sum node can be interpreted as introducing a discrete altent variable Z that selects among its children; aggregating these latents makes the PC a hierarchical latent-variable model. That explains EM:

- **E-step:** compute $p(Z|x)$ (posterior over latent choices) using exact inference in the circuit.
- **M-step:** update sum-node weights using those posteriors.

The paper presents EM in the standard latent-variable form(maximize log-likelihood via the surrogate Q).

2 Why do PCs change inference?

Traditional probabilistic models (graphical models, HMMs) often define a joint distribution and then inference (marginals/conditionals) requires specialized algorithms (variable elimination, message passing) and can become expensive.
PCs flip this perspective, they represent a distribution as a computation graph that directly mirrors the oeprations needed for inference. The paper describes PCs as recursively composing distributions with **sum (mixtures)** and **product(factorization)** operations, inspired by how tractable models like HMMs perform marginal inference and distilling their computation graphs into a unified framework.

2.1 How a PC represents a distribution?

A node's distribution is defined recursively, where a leaf/input node: a primitive univariate distribution $f_n(x)$(e.g. Gaussian), a product node where we multiply child distributions, and a sum node which is the weighted sum of child distributions. Liu, Shao and Broeck [1] notes that to ensure **exact and efficient** computation of probabilistic queries (marginals/conditionals/moments), circuits typically satisfy structural constraints like smoothness and decomposability, which make marginalization tractable. This is why PCs are advertised as supporting exact and efficient inference over many queries. In practice, exact inference often means set evidence at leaves (or set unknown-variable leaves to integrate/sum to 1), do a bottom-up pass (sums/products) or optionally do a top-down/backprop-like pass for posteriors over latent choices. **No sampling required**.

3 Minimal PC example with Gaussians

3.1 The circuit

We build a simple PC: a **a single sum node** whose children are two Gaussian leaves. This is a Gaussian mixture model, but viewed as a PC:

1. leaf 1: $p_1(x) = \mathcal{N}(x|\mu_1 = -1, \sigma^2 = 1)$
2. leaf 2: $p_2(x) = \mathcal{N}(x|\mu_2 = 3, \sigma^2 = 1)$
3. sum-node weights: $\theta_1 = 0.5$, $\theta_2 = 0.5$

From Equation 1 from Liu, Shao and Broeck [1],

$$\prod_{c \in ch(n)} p_c(x)$$

The root computes:

$$p(x) = \theta_1 p_1(x) + \theta_2 p_2(x)$$

This is inference as evaluation: plug in x, evaluate two leaves, weighted sum them at the root.

3.2 The latent choice Z

Here $Z \in \{1, 2\}$, is "which child of the sum node generated this datapoint?". Exact posterior ("responsibility") is:

$$p(Z = 1|x) = \frac{\theta_1 p_1(x)}{\theta_1 p_1(x) + \theta_2 p_2(x)}$$

This is exact (no approximation) based on quantities the circuit has already computed.

3.3 Expectation Maximization for PC using 5 datapoints

Let us have 5 observed scalars:

$$D = \{-1.5, -0.8, 0.2, 2.6, 3.1\}$$

E-step: Compute $p_1(x_i), p_2(x_i)$, then responsibilities $r_i := p(Z = 1|x_i)$. Using Gaussian density values with $\sigma = 1$ and $\theta_1 = \theta_2 = 0.5$.

x	$p_1(x) = \mathcal{N}(x \mid -1, 1)$	$p_2(x) = \mathcal{N}(x \mid 3, 1)$	$r = p(Z = 1 \mid x)$
-1.5	0.3521	0.0000160	0.99995
-0.8	0.3910	0.000292	0.99925
0.2	0.1942	0.007915	0.96083
2.6	0.000612	0.36827	0.001659
3.1	0.0000893	0.39695	0.000225

The first 3 points are overwhelmingly explained by the left Gaussian and the last two by the right Gaussian. This is the E-step.

3.4 How the paper expresses the M-step: "flows" and the closed-form EM update

The paper distinguishes unnormalized circuit output $\tilde{p}_\phi(x)$ from normalized $p_\phi(x) = \tilde{p}_\theta(x)/Z(\phi)$. For this toy mixture, with normalized Gaussians and $\theta_1 + \theta_2 = 1$, the model is already normalized. In the paper, they focus on learning **sum-edge parameters** $\theta_{n,c}$ (or log-parameters $\phi_{n,c} = \log \theta_{n,c}$). "Flows" are the sufficient-statistics analogue. Full-batch EM has the closed-form update (Equation 5 from Liu, Shao and Broeck [1]):

$$\theta'_{n,c} = \frac{F^D_\phi(n,c)}{\sum_{c' \in ch(n)} F^D_\phi(n,c')} \text{ where } F^D_\phi(n,c') = \frac{1}{|D|} \sum_{x \in D} \frac{\delta \log \tilde{p}_\phi(x)}{\delta \phi_{n,c}}$$

For a single sum node in our example, those derivatives reduce to an intuitive thing: posterior responsibility mass assigned to each child. The appendix shows (for a sum node), the per-sample normalized term becomes:

$$F^x_\phi(n,c') = \frac{\theta_{n,c} \tilde{p}_c(x)}{\tilde{p}_n(x)}$$

and $\sum_c F^x_\phi(n,c) = 1$. That ratio is the responsibility we know from Gaussian mixtures. Therefore, in our M-step for the toy example, as there is only one sum node, the EM update becomes: Because there's only one sum node, the EM update becomes:

$$\theta_1^{\text{new}} = \frac{1}{5} \sum_{i=1}^{5} r_i, \qquad \theta_2^{\text{new}} = 1 - \theta_1^{\text{new}}.$$

From the table:

- $\sum r_i \approx 2.96193$
- so $\theta_1^{\text{new}} \approx 2.96193/5 \approx 0.59239$
- $\theta_2^{\text{new}} \approx 0.40761$

That's **full-batch EM** for a PC in its simplest form:

1. exact inference gives responsibilities (E-step),
2. normalize aggregated responsibilities into new sum-edge weights (M-step), matching Equation 5.

References

1. Liu A, Shao Z and Broeck GV den. Rethinking Probabilistic Circuit Parameter Learning. 2025. arXiv: `2505.19982 [cs.LG]`. Available from: `https://arxiv.org/abs/2505.19982`