Rethinking Probabilistic Circuit Parameter Learning

Anji Liu[1] **Zilei Shao**[2] **Guy Van den Broeck**[2]
[1]School of Computing, National University of Singapore
[2]University of California, Los Angeles

Abstract

Probabilistic Circuits (PCs) offer a computationally scalable framework for generative modeling, supporting exact and efficient inference of a wide range of probabilistic queries. While recent advances have significantly improved the expressiveness and scalability of PCs, effectively training their parameters remains a challenge. In particular, a widely used optimization method, full-batch Expectation-Maximization (EM), requires processing the entire dataset before performing a single update, making it ineffective for large datasets. Although empirical extensions to the mini-batch setting, as well as gradient-based mini-batch algorithms, converge faster than full-batch EM, they generally underperform in terms of final likelihood. We investigate this gap by establishing a novel theoretical connection between these practical algorithms and the general EM objective. Our analysis reveals a fundamental issue that existing mini-batch EM and gradient-based methods fail to properly regularize distribution changes, causing each update to effectively "overfit" the current mini-batch. Motivated by this insight, we introduce an**em**one, a new mini-batch EM algorithm for PCs. An**em**one applies an implicit adaptive learning rate to each parameter, scaled by how much it contributes to the likelihood of the current batch. Across extensive experiments on language, image, and DNA datasets, an**em**one consistently outperforms existing optimizers in both convergence speed and final performance.

1 Introduction

Probabilistic Circuits (PCs) are a class of generative models that represent probability distributions by recursively composing simpler distributions through sum (mixture) and product (factorization) operations (Choi et al., 2020). The key idea behind PCs is to examine how tractable probabilistic models, such as Hidden Markov Models (Rabiner and Juang, 1986), perform inference (e.g., computing marginal probabilities). PCs distill the structure of these models' computation graphs into a compact and general framework, which leads to a unified, computation-oriented perspective on tractable probabilistic modeling.

While significant progress has been made in improving the expressiveness of PCs through architectural innovations (Loconte et al., 2025; Liu and Van den Broeck, 2021) and system-level advancements (Liu et al., 2024; Peharz et al., 2020), there is still no clear consensus on how to effectively learn their parameters. Full-batch Expectation-Maximization (EM) and its empirical variants remain widely used approaches (Zhang et al., 2025; Liu et al., 2023a). However, full-batch EM requires aggregating information across the entire dataset before each parameter update, making it hard to scale to large datasets or streaming settings. Although mini-batch extensions and gradient-based optimization methods can converge faster, they typically achieve lower final log-likelihood than full-batch EM.

Based on Kunstner et al. (2021), which studies the full-batch EM algorithm for exponential-family latent variable models, we discover that the full-batch EM update of PCs corresponds to optimizing a 1st order Taylor approximation, regularized by a Kullback–Leibler (KL) divergence that penalizes deviation from the current distribution. This yields a novel view of the full-batch EM update for PCs, which has appeared in various forms across different contexts (Peharz, 2015; Choi et al., 2021; Poon and Domingos, 2011).

This perspective naturally suggests a theoretically grounded mini-batch extension: by increasing the weight on the KL term, we can compensate for the reduced information available in a mini-batch compared to the full dataset. The resultant algorithm, an**em**one ("an EM one"), adaptively applies large learning rates only to the PC parameters that strongly influence the

likelihood of the current batch. The other parameters are kept (almost) unchanged to prevent excessive distribution drift, thereby ensuring that updates remain consistent with the data distribution.

The theoretical insights also explain the inferior performance of existing EM- and gradient-based mini-batch algorithms, as they regularize the distribution shift before and after an update using the KL divergence of the local distributions defined at sum nodes[1] and the L2 distance in the parameter space, respectively. As a result, these methods fail to effectively minimize the distribution shift while learning to improve the likelihood given the current batch, causing each update to "overfit" to the current samples and impede overall convergence.

An**em**one admits a closed-form expression, making it efficient and easy to implement. We conduct extensive empirical evaluations on three types of datasets (language, image, and DNA) and four widely used classes of PC architectures. The results demonstrate that an**em**one consistently and significantly outperforms existing optimizers in both convergence speed and final likelihood.

2 Background

2.1 Distributions as Circuits

Probabilistic Circuits (PCs) represent probability distributions with deep and structured computation graphs that consist of sum and product operations (Choi et al., 2020). They serve as a general framework encompassing tractable probabilistic models, which are designed to support efficient and exact probabilistic inference over complex queries, such as Sum Product Networks (Poon and Domingos, 2011), cutset networks (Rahman et al., 2014), Hidden Markov Models (Rabiner and Juang, 1986), and Probabilistic Generating Circuits (Zhang et al., 2021). The syntax and semantics of PCs are as follows:

Definition 1 (Probabilistic Circuit)**.** A PC p over variables $\mathbf{X}$ is a directed acyclic computation graph with one single root node n_{r}. Every input node (those without incoming edges) in p defines an univariate distribution over variable $X \in \mathbf{X}$. Every inner node (those with incoming edges) is either a *product* or a *sum* node, where each product node encodes a factorized distribution over its child distributions and each sum node represents a weighted mixture of its child distributions. Formally, the distribution p_n encoded by a node n can be represented recursively as

$$p_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & n \text{ is an input node,} \\ \prod_{c \in \mathsf{ch}(n)} p_c(\boldsymbol{x}) & n \text{ is a product node,} \\ \sum_{c \in \mathsf{ch}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) & n \text{ is a sum node,} \end{cases} \quad (1)$$

where f_n is an univariate primitive distribution defined over $X \in \mathbf{X}$ (e.g., Gaussian, Categorical), $\mathsf{ch}(n)$ denotes the set of child nodes of n, and $\theta_{n,c} \geq 0$ is the parameter corresponds to the edge (n, c) in the PC. Define the *log-parameter* of (n, c) as $\phi_{n,c} := \log \theta_{n,c}$, which will be used interchangeably with $\theta_{n,c}$. We further denote $\boldsymbol{\phi} := \{\phi_{n,c}\}_{(n,c)}$ as the set of all sum node parameters in the PC. Without loss of generality, we assume that every path from the root node to an input node alternates between sum and product nodes.[2]

To ensure the exact and efficient computation of various probabilistic queries, including marginalization and moment calculations, we must impose structural constraints on the circuit. Specifically, smoothness and decomposability (Peharz et al., 2015) are a set of sufficient conditions that ensure tractable computation of marginal and conditional probabilities. This tractability arises because smooth and decomposable circuits represent multilinear functions, which are known to support efficient marginalization (Broadrick et al., 2024). We provide details in Appendix A.

PCs can be viewed as latent variable models with discrete latent spaces (Peharz et al., 2016). Each sum node can be interpreted as introducing a discrete latent variable Z that selects among its child distributions. Specifically, assigning $Z = i$ corresponds to choosing the i-th child of the sum node. By aggregating all such latent variables, the PC can be seen as defining a hierarchical latent variable model.

2.2 Expectation-Maximization

Expectation-Maximization (EM) is a well-known algorithm to maximize the log-likelihood given data $\boldsymbol{x}$ of a distribution defined by a latent variable model. Specifically, the distribution $p_{\boldsymbol{\phi}}(\mathbf{X})$ with parameters $\boldsymbol{\phi}$ is defined as $\sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\mathbf{X}, \boldsymbol{z})$ over latents $\mathbf{Z}$. Our goal is to maximize

$$\mathtt{LL}(\boldsymbol{\phi}) := \log p_{\boldsymbol{\phi}}(\boldsymbol{x}) = \log \Big(\sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) \Big). \quad (2)$$

EM can effectively maximize the above objective when $p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z})$ permits much simpler (or even closed-form) maximum likelihood estimation. It optimizes $\mathtt{LL}(\boldsymbol{\phi})$ by maximizing the following surrogate objective:

$$Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}') := \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \cdot \log p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z}). \quad (3)$$

[1] See Section 2 for the definition of sum nodes.

[2] This can be efficiently enforced since we can directly "collapse" consecutive sum nodes or product nodes.

EM updates the current parameters $\boldsymbol{\phi}$ by solving for $\boldsymbol{\phi}'$ that maximizes $Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$, which is guaranteed to be a lower bound of $\texttt{LL}(\boldsymbol{\phi}')$ since

$$Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = \texttt{LL}(\boldsymbol{\phi}') + \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \cdot \log p_{\boldsymbol{\phi}'}(\boldsymbol{z}|\boldsymbol{x}) \leq \texttt{LL}(\boldsymbol{\phi}').$$

3 EM for Probabilistic Circuits

While variants of the EM algorithm have been proposed for training PCs in various contexts (Poon and Domingos, 2011; Peharz, 2015), their connection to the general EM objective $Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ (cf. Eq. (3)) remains implicit. The lack of a unified formulation makes it difficult to fully understand the existing optimization procedures or to extend them to new settings, such as training with mini-batches of data, which is critical for scaling the optimizer to large datasets.

Specifically, there are multiple ways to define mini-batch EM algorithms that all reduce to the same full-batch EM algorithm in the limit. However, it is often unclear what objective these variants are optimizing in the mini-batch case, which complicates the design of new learning algorithms.

In this section, we bridge this gap by deriving EM for PCs explicitly from the general objective. In Section 3.1, we begin with a derivation for the full-batch case, showing how existing formulations can be recovered and interpreted from this viewpoint. We then extend the derivation to the mini-batch setting in Section 3.2, leading to a principled and theoretically-grounded mini-batch EM algorithm for PCs.

3.1 Revisiting Full-Batch EM

Recall from Definition 1 that we define the log-parameter that corresponds to the edge (n, c) as $\phi_{n,c} := \log \theta_{n,c}$, and the set of all parameters of a PC as $\boldsymbol{\phi} := \{\phi_{n,c}\}_{n,c}$.[3] Since $\boldsymbol{\phi}$ does not necessarily define a normalized PC, we distinguish between the unnormalized and normalized forms of the model: let $\tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})$ denote the unnormalized output of the circuit computed via the feedforward pass defined by Equation (1), and define the normalized distribution as

$$p_{\boldsymbol{\phi}}(\boldsymbol{x}) := \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x}) / Z(\boldsymbol{\phi}),$$

where $Z(\boldsymbol{\phi})$ is the normalizing constant. We extend the single-sample EM objective in Equation (3) to the following, which is defined on a dataset $\mathcal{D}$:

$$Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}') := \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \cdot \log p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z}).$$

[3]We assume for simplicity that distributions of input nodes have no parameters (e.g., indicator distributions). Our analysis can be easily extended to exponential family input distributions.

Our analysis is rooted in the following result.

Proposition 1. *Given a PC $p_{\boldsymbol{\phi}}$ with log-parameters $\boldsymbol{\phi}$ (cf. Def. 1) and a dataset $\mathcal{D}$, $Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ equals the following up to a constant term irrelevant to $\boldsymbol{\phi}'$:*

$$\frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \left\langle \frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' \right\rangle - \texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}'), \tag{4}$$

where $\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}') := \mathrm{D}_{\mathrm{KL}}\left(p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z}) \,\|\, p_{\boldsymbol{\phi}'}(\mathbf{X}, \mathbf{Z})\right)$ is the KL divergence between $p_{\boldsymbol{\phi}}$ and $p_{\boldsymbol{\phi}'}$.

The proof follows Kunstner et al. (2021) and is provided in Appendix B.1. Proposition 1 reveals that the EM update can be interpreted as maximizing a *regularized first-order approximation* of the log-likelihood. To see this, we rewrite Equation (4) by adding terms irrelevant to $\boldsymbol{\phi}'$:

$$\frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \underbrace{\log p_{\boldsymbol{\phi}}(\boldsymbol{x}) + \left\langle \frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle}_{\texttt{LinLL}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')} - \texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}').$$

In the above equation, the term $\texttt{LinLL}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ corresponds to the linearization of $\log p_{\boldsymbol{\phi}'}(\boldsymbol{x})$ around the current parameters $\boldsymbol{\phi}$, capturing the local sensitivity of the log-likelihood to parameter changes. The KL term, $\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$, acts as a regularizer that penalizes large deviations in the joint distribution over $\mathbf{X}$ and $\mathbf{Z}$.

According to Proposition 1, solving for the updated parameters $\boldsymbol{\phi}'$ requires computing two key quantities: the gradient of the log-likelihood $\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x}) / \partial \boldsymbol{\phi}$, and the KL divergence $\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$. To express these terms in closed form, we introduce the concept of top-down probabilities, which is defined by Dang et al. (2022).

Definition 2 (TD-prob)**.** Given a PC p parameterized by $\boldsymbol{\phi}$, we define the top-down probability $\texttt{TD}(n)$ of a node n recursively from the root node to input nodes:

$$\texttt{TD}(n) := \begin{cases} 1 & n \text{ is the root node,} \\ \sum_{m \in \texttt{pa}(n)} \texttt{TD}(m) & n \text{ is a sum node,} \\ \sum_{m \in \texttt{pa}(n)} \theta_{m,n} \cdot \texttt{TD}(m) & n \text{ is a product node,} \end{cases}$$

where $\theta_{m,n} := \exp(\phi_{m,n})$ and $\texttt{pa}(n)$ is the set of parent nodes of n. Define the TD-prob of $\phi_{m,n}$ as $\texttt{TD}(\phi_{m,n}) := \theta_{m,n} \cdot \texttt{TD}(m)$, and denote by $\texttt{TD}(\boldsymbol{\phi})$ the vector containing the TD-probs of all edge parameters in the circuit.

Intuitively, the TD-prob of a parameter quantifies how much influence it has on the overall output of the PC, and in particular, on the normalizing constant $Z(\boldsymbol{\phi})$.[4] We continue to express the two key terms in Proposition 1 in closed form.

[4]This can be observed from the fact that $Z(\boldsymbol{\phi})$ can be computed via the same feedforward pass (Eq. (1)) except that we set the output of input nodes to 1.

Lemma 1. *Assume the distributions defined by all nodes in a PC are normalized. For every $\boldsymbol{x}$, we have:*

$$\textit{(i)}\ \partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})/\partial\boldsymbol{\phi} = \partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})/\partial\boldsymbol{\phi} - \texttt{TD}(\boldsymbol{\phi}),$$
$$\textit{(ii)}\ \texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = -\langle \texttt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}'\rangle + C,$$

where C is a constant term independent of $\boldsymbol{\phi}'$.

The assumption that the PC is normalized is mild and practical. In Section 4 and Appendix C, we introduce a simple and efficient algorithm that adjusts the PC parameters to ensure normalization without affecting the structure of the circuit. We can now substitute the closed-form expressions for the gradient and the KL divergence into the general EM objective $Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$, which converts the problem into[5]

$$\left\langle \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} - \cancel{\texttt{TD}(\boldsymbol{\phi})}, \boldsymbol{\phi}' \right\rangle + \cancel{\langle \texttt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}'\rangle}.$$

If we additionally require each node in the PC to define a normalized distribution, we impose the constraint $\sum_{c\in\mathsf{ch}(n)} \exp(\phi'_{n,c}) = 1$ for all sum nodes n. Incorporating these constraints into the EM objective results in a constrained maximization problem that has the following solution for every edge (n, c) (see Appx. B.2 for the derivation):

$$\phi'_{n,c} = \log \theta'_{n,c}, \quad \theta'_{n,c} = \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)/Z, \tag{5}$$

where we define $\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c) := \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\phi_{n,c}}$,[6] and $Z = \sum_{c'\in\mathsf{ch}(n)} \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c')$ ensures that n is normalized.

While this full-batch EM algorithm in Equation (5) has been derived in prior work (Choi et al., 2021; Peharz, 2015), we recover it here through Proposition 1. This paves the way for a principled mini-batch EM algorithm by generalizing the objective $Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$, as shown in the next section.

3.2 Extension to the Mini-Batch Case

When the dataset is large, full-batch EM becomes inefficient and impractical as it requires scanning the entire dataset before making any parameter updates. In such cases, we instead wish to update the parameters after processing only a small subset of data points, which is commonly referred to as a mini-batch. Given a mini-batch of samples $\mathcal{D}$, Peharz et al. (2020) propose to update the parameters following:

$$\theta'_{n,c} = (1-\alpha)\cdot\theta_{n,c} + \alpha\cdot \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)/Z,\ (\alpha\in(0,1)) \tag{6}$$

where we borrow notation from Equation (5). However, it remains unclear whether this update rule is grounded in a principled EM objective. In the following, we show that from the full-batch EM derivation, we can derive a mini-batch update rule that closely resembles the above, but with a crucial difference.

Proposition 1 expresses the EM objective as the sum of two terms: a linear approximation of the log-likelihood and a regularization term that penalizes deviation from the current model via KL divergence. When using only a mini-batch of samples, the log-likelihood may overlook parts of the data distribution not covered by the sampled subset.

To account for this, we can put a weighting factor $\gamma > 1$ on the KL divergence (i.e., $\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ becomes $\gamma\cdot\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$).[7] Plugging in Lemma 1 and dropping terms independent to $\boldsymbol{\phi}'$, the adjusted objective (i.e., $Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ with the additional weighting γ) becomes

$$\left\langle \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}, \boldsymbol{\phi}'\right\rangle + (\gamma-1)\cdot\langle \texttt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}'\rangle,$$

where $\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}$ collects all entries $\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$ into a single vector, with each $\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$ representing the aggregated gradient w.r.t. $\phi_{n,c}$. With the constraints that ensure each PC node defines a normalized distribution (i.e., for each sum node n, $\sum_{c\in\mathsf{ch}(n)}\theta'_{n,c}=1$), the solution is

$$\theta'_{n,c} = \left(\texttt{TD}_{\boldsymbol{\phi}}(n)\cdot\theta_{n,c} + \eta\cdot F^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)\right)/Z, \tag{7}$$

where $\eta := 1/(\gamma-1)$ is the learning rate, $\texttt{TD}_{\boldsymbol{\phi}}(n)$ is the TD-prob of node n (Def. 2), and Z is a normalizing constant. The derivation is deferred to Appendix B.2. In practice, compared to the full-batch EM update (Eq. (5)), the only additional computation required is $\texttt{TD}_{\boldsymbol{\phi}}(n)$, which can be efficiently implemented using any autodiff library to compute the gradient of $Z(\boldsymbol{\phi})$ w.r.t. the log-parameters $\phi_{n,c}$ (see proof in Appx. B.4).

To build intuition for the update rule, we consider the case where $\mathcal{D}$ contains only a single sample $\boldsymbol{x}$. In this setting, the update direction $\texttt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c)$ can be decomposed using the chain rule of derivatives:

$$\texttt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c) = \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\phi_{n,c}} = \underbrace{\frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}}_{\texttt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n)} \cdot \underbrace{\frac{\partial \log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\phi_{n,c}}}_{\hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n,c)},$$

where we define $\log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})$ as the (unnormalized) log-likelihood of node n. A key observation is that the second term $\hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c)$ is normalized w.r.t. all children of sum node n: $\sum_{c\in\mathsf{ch}(n)}\hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c) = 1$ (see Appx. B.3 for the proof). Intuitively, we now break down $\texttt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c)$ into the *importance of node n* to the overall output (i.e., $\texttt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n)$) and the *relative contribution of child c* to

[5] We drop all terms that are independent of $\boldsymbol{\phi}'$.

[6] $\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$ is to the PC flows defined by Choi et al. (2021).

[7] Note that this is equivalent to $Q^{\mathcal{D}}_{\boldsymbol{\phi}}(\boldsymbol{\phi}') - (\gamma-1)\cdot\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ according to Proposition 1.

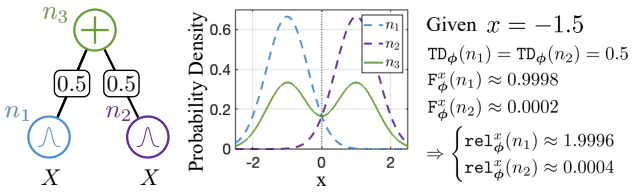

Figure 1: The proposed algorithm implicitly applies an adaptive learning rate to each node. For the PC shown on the left, given a sample $x = -1.5$, the algorithm uses a large learning rate to update n_1 while keeping n_2 almost unchanged.

n (i.e., $\hat{\mathtt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c)$). This simplifies Equation (7) as

$$\theta'_{n,c} = \big(\theta_{n,c} + \eta \cdot \mathtt{rel}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n) \cdot \hat{\mathtt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n, c)\big)/Z,$$

where $\mathtt{rel}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n) := \mathtt{F}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n)/\mathtt{TD}_{\boldsymbol{\phi}}(n)$ can be viewed as the relative importance of n to the PC output given $\boldsymbol{x}$. The term $\eta \cdot \mathtt{rel}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n)$ then acts as an adaptive learning rate for updating the child parameters of n, scaling the update magnitude according to how influential n is for $\boldsymbol{x}$. In comparison, the mini-batch algorithm in Equation (6) uses a fixed learning rate for all parameters.

This difference is reflected in the example shown in Figure 1. Given the PC on the left, which represents a mixture of two Gaussians (middle). Suppose we draw one sample $x = -1.5$. This sample does not reflect the full distribution and only activates the left mode. Our algorithm accounts for this by assigning a small effective learning rate to node n_2 ($\mathtt{rel}^{x}_{\boldsymbol{\phi}}(n_2) \approx 0.0004$) as it is "not responsible" for explaining this particular input and focuses the update on n_1 ($\mathtt{rel}^{x}_{\boldsymbol{\phi}}(n_1) \approx 1.9996$). In contrast, Equation (6) applies a uniform learning rate across all parameters, leading it to also update n_2 unnecessarily to fit the current sample.

4 Connections with Gradient-Based Optimizers

Gradient-based optimization methods, such as stochastic gradient descent (SGD), can be interpreted through a lens similar to the EM formulation. Recall from Proposition 1 that the EM objective comprises a linear approximation of the log-likelihood, along with a KL divergence regularizer that penalizes deviations from the current model. In contrast, standard gradient-based updates can be viewed as maximizing the same linear approximation of the log-likelihood, but with an *L2 regularization* penalty on parameter updates instead of a KL divergence:

$$\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \log p_{\boldsymbol{\phi}}(\boldsymbol{x}) + \left\langle \frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle - \gamma \|\boldsymbol{\phi}' - \boldsymbol{\phi}\|_2^2.$$

Solving for $\boldsymbol{\phi}'$ gives $\boldsymbol{\phi}' = \boldsymbol{\phi} + \eta \cdot \partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})/\partial \boldsymbol{\phi}$, a standard gradient ascent step with $\eta = 1/(2\gamma)$.

The Regularization Terms. While this analogy highlights a shared structure between EM and gradient-based optimizers, it also reveals a fundamental discrepancy. The KL divergence in the EM formulation is a natural measure of distance between distributions, while the L2 penalty in gradient-based updates only constrains the movement in parameter space. This distinction is important because proximity in parameter space does not necessarily translate to proximity in distribution space. For instance, adding all parameters $\boldsymbol{\phi}$ (note that they represent log-probabilities) by a constant leaves the distribution unchanged, yet the L2 penalty would still register this as a large deviation. Therefore, compared to gradient-based methods, the EM formulation better respects the geometry of distributions.

Normalization Constraints. Another notable difference between gradient-based methods and the EM algorithms (both the full-batch and the mini-batch ones) is the treatment of normalization constraints. In EM, parameters are updated in a way that preserves local normalization, i.e., the edge parameters of each sum node n are guaranteed to sum to one after every update. On the other hand, standard gradient-based optimization does not enforce this constraint, and thus intermediate parameter values may fall outside the normalized parameter space.

One might naturally wonder whether this discrepancy leads to better or worse training dynamics, since optimization would be carried out in an enlarged parameter space. Perhaps surprisingly, we show that it has *no* effect on the optimization process, as the parameters $\boldsymbol{\phi}$ can always be mapped to a locally normalized counterpart $\boldsymbol{\phi}'$ that preserves both the represented distribution and the gradients with respect to the parameters.

Specifically, given a PC with parameters $\boldsymbol{\phi}$ (corresponds to $\boldsymbol{\theta} := \exp(\boldsymbol{\phi})$), there exists a normalization algorithm that outputs $\boldsymbol{\phi}'$ and ensures (i) all parameters are locally normalized (i.e., $\forall n, \sum_{c\in\mathsf{ch}(n)} \theta_{n,c} = 1$), (ii) the represented distributions are unchanged (i.e., $\forall \boldsymbol{x}, \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x}) \propto \tilde{p}_{\boldsymbol{\phi}'}(\boldsymbol{x})$), and (iii) the gradients with respect to the parameters are preserved (i.e., $\forall \boldsymbol{x}, \partial \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})/\partial \boldsymbol{\phi} = \partial \tilde{p}_{\boldsymbol{\phi}'}(\boldsymbol{x})/\partial \boldsymbol{\phi}'$.

Intuitively, the above three conditions guarantee that applying this normalization algorithm neither alters the model's probabilistic semantics nor interferes with the optimization dynamics. In other words, the model continues to represent the same distribution, and the gradients used for subsequent updates remain the same. Therefore, it can be applied after each gradient update without changing the learning dynamics.

In the following, we present the normalization algo-

rithm and show how it can be implemented efficiently.[8] A detailed analysis and formal proofs of the three properties are provided in Appendix C.

The Algorithm. The normalization procedure consists of two simple passes over the PC. First, we perform a feedforward evaluation of the PC to compute the partition function $Z_n(\boldsymbol{\phi})$ at every node n:

$$Z_{\boldsymbol{\phi}}(n) = \begin{cases} 1 & n \text{ is an input node,} \\ \prod_{c \in \mathsf{ch}(n)} Z_{\boldsymbol{\phi}}(c) & n \text{ is a product node,} \\ \sum_{c \in \mathsf{ch}(n)} \exp(\phi_{n,c}) \cdot Z_{\boldsymbol{\phi}}(c) & n \text{ is a sum node.} \end{cases}$$

Next, for every edge (n, c) where n is a sum node and $c \in \mathsf{ch}(n)$, we update the parameter as

$$\phi'_{n,c} = \log\left(\frac{\theta_{n,c} \cdot Z_{\boldsymbol{\phi}}(c)}{Z_{\boldsymbol{\phi}}(n)}\right). \tag{8}$$

5 Experiments

In this section, we empirically evaluate an**em**one against existing EM- and gradient-based optimizers across a range of PC models and datasets. Section 5.1 introduces the models, datasets, and baseline optimizers. In Section 5.2, we ask whether an**em**one can achieve higher log-likelihoods at convergence compared to existing approaches, and whether an**em**one converges faster. Finally, Section 5.3 investigates the key design factors in setting hyperparameters through a series of ablation studies.

5.1 Experimental Setup and Baselines

We conduct our empirical evaluation across three distinct domains (i.e., DNA sequence, image, and text) using various PC architectures.

DNA Sequence Modeling. We evaluate density estimation performance on a high-dimensional genomics dataset from the UK Biobank (Sudlow et al., 2015). For this task, we train Hidden Chow-Liu Trees (HCLTs) (Liu and Van den Broeck, 2021) and their variant that combines the PD structure (Poon and Domingos, 2011) termed Partitioned Data HCLTs (PDHCLTs), which is defined in Appendix E. A detailed description of the dataset and all model configurations is provided in Appendix D.1.

Image Modeling. We adopt the ImageNet32 dataset (Deng et al., 2009) with two color transformations, i.e., a lossy YCC transformation and its lossless variant YCC-R (Malvar and Sullivan, 2003). Each 32×32 image is partitioned into four 16×16 image patches, which results in a total of $16 \times 16 \times 3 = 768$ variables. We employ HCLTs and PDHCLTs for both datasets. Details of the datasets and the models are included in Appendix D.1.

Language Modeling. We use the WikiText-103 (Merity et al., 2017) dataset, which is widely used for language modeling. The dataset is preprocessed by the GPT-2 tokenizer (Radford et al., 2019a). We evaluate the Hidden Markov Model (HMM) PC architecture and its recently proposed variant Monarch HMM (Zhang et al., 2025) on it.

EM-Based Baselines. We adopt two EM baselines, which are the standard full-batch EM and the mini-batch EM proposed by Peharz et al. (2020) (cf. Eq. (6)). For the full-batch EM, there are no hyperparameters to tune. In contrast, for the mini-batch EM, we tune the batch size in the range $\{512, 16384\}$ and the step size α in Eq. (6) over $\{0.05, 0.1, 0.4\}$.

Gradient-Based Baselines. We adopt the Adam optimizer (Adam et al., 2014), which is used by many recent works. We tune the batch size and the learning rate in the ranges $\{512, 1024\}$ and $\{1 \times 10^{-2}, 3 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-3}\}$, respectively.

Our Method. For ease of definition, we express the step size of our method as $\alpha = \eta/(\eta - 1)$, where η is given in Equation (7). The hyperparameter search is detailed in Appendix D.2. We propose to apply a momentum update to the flows $\mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$. Specifically, we initialize the momentum flows $\mathtt{Fm}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c) = \mathbf{0}$, then before every EM step, we update $\mathtt{Fm}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$ following:

$$\mathtt{Fm}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c) \leftarrow \beta \cdot \mathtt{Fm}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c) + (1 - \beta) \cdot \mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c).$$

Finally, we replace the $\mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)$ in Equation (6) with $\mathtt{Fm}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)/(1 - \beta^{T+1})$, where T is the number of updates performed. We use the proposed momentum update with $\beta = 0.9$, and we compare the performance with and without the momentum update for both an**em**one and mini-batch EM in Section 5.3.

5.2 Overall Performance and Convergence

We begin by examining the training dynamics of different optimizers. Figure 2 shows the training curves (i.e., validation LL vs. number of epochs) across all four datasets, each paired with an appropriate PC architecture (see the caption for the details).

We start by focusing on the three baseline approaches, i.e., full-batch EM, Adam, and mini-batch EM. A consistent trend is that full-batch EM consistently reaches better (or comparable) performance at convergence. This is a strong indicator that the existing mini-batch

[8] The existence of this normalization algorithm has been previously shown by Martens and Medabalimi (2014), although they did not provide an explicit algorithm.

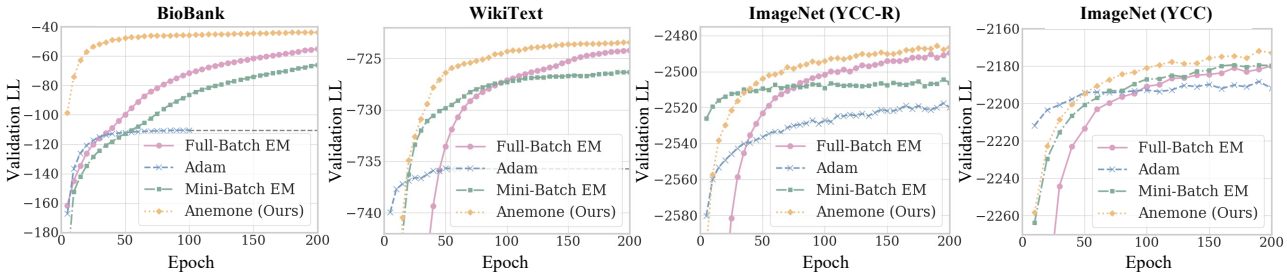

Figure 2: **Log-Likelihood over epochs on four diverse datasets.** For the ImageNet (YCC-R) and ImageNet (YCC) datasets, an HCLT with hidden size 512 is used; for the WikiText dataset, an HMM with hidden size 256 is used; for the BioBank dataset, a PDHCLT with hidden size 1024 is used. An**em**one achieves significantly faster convergence as well as final LL across all four cases.

Table 1: **Negative LLs on the UK BioBank Chromosome 6 dataset.** An**em**one consistently and significantly outperforms all baseline across every tested model architecture. The results also show a clear performance hierarchy among the baseline methods. The best result in each column is marked in bold.

Optimizer	BioBank Chr6			
	HCLT 512	HCLT 1024	PDHCLT 512	PDHCLT 1024
Full-batch EM	55.3	53.8	46.5	45.1
Adam	102.7	100.4	112.4	110.4
Mini-batch EM	55.7	55.5	49.5	47.2
An**em**one (Ours)	**54.5**	**52.1**	**45.3**	**42.2**

Table 2: **Convergence speed (epochs) for PDHCLT 1024 on BioBank Chr6 Dataset.** The table reports epochs to reach specific LL thresholds, with Δ representing the difference from the best LL of -45.1 before an**em**one (Table 1). Lower is better. Bold marks the best result per column; ∞ indicates failure to reach the threshold in time.

Method	LL $\geq$ -48 ($\Delta \approx 2.9$)	LL $\geq$ -46 ($\Delta \approx 0.9$)	LL $\geq$ -45.1 ($\Delta = 0$)
Full EM	450	645	1000
Adam	∞	∞	∞
Mini EM	715	∞	∞
An**em**one	**50**	**80**	**130**

optimizers (Adam and mini-batch EM) fail to compensate for the reduced information available in a mini-batch compared to the full dataset.

In contrast, despite being a mini-batch algorithm, an**em**one achieves significantly better performance at convergence, even compared to full-batch EM. We evaluate the final log-likelihoods of each optimizer across a wider range of PC architectures. The final log-likelihoods, evaluated either at convergence or after a fixed maximum number of epochs, are shown in Tables 1, 3 and 4 for the DNA, image, and text modeling tasks, respectively. On the BioBank Chr6 dataset, an**em**one achieves consistent and significant performance gains over all baselines. For the ImageNet32 and WikiText datasets, it again obtains better results in the majority of cases.

Additionally, as indicated by Figure 2, an**em**one converges faster than all baselines, including the mini-batch ones that are designed for faster convergence. We further quantify the number of epochs used to reach a certain LL. Specifically, as shown in Table 2 an**em**one requires $\sim$8x fewer epochs to reach the same validation LL. Further experiments are deferred to Appendix D.4.

5.3 Ablation Study

To disentangle the performance gains of an**em**one from the general effects of momentum (cf. Section 5.1), we conduct ablation study shown in Figure 3. While the results confirm that momentum improves the final log-likelihood for an**em**one (left panel), they also show that it provides little benefit when applied to vanilla mini-batch EM (right panel), suggesting that the improvements are not from momentum alone, but from the synergy between momentum and an**em**one, that is not observed in standard mini-batch approaches.

6 Related Works and Conclusion

Modeling Advancements in PCs. The development of PCs has been marked by significant progress in enhancing their expressiveness and utility as generative models. Since the initial establishment of PCs, research has focused on developing more expressive

Table 3: **Negative LLs on the ImageNet32 dataset's validation subset.** An**em**one consistently outperforms other baselines when training HCLT models. We observe a general performance ranking where Adam optimizer is the weakest, followed by Mini-batch EM and Full-batch EM. OOM denotes out-of-memory errors due to computing resource limitations. The best result in each column is marked in bold.

Optimizer	ImageNet32 YCC-R			ImageNet32 YCC	
	PDHCLT 256	HCLT 512	HCLT 1024	HCLT 512	HCLT 1024
Full-batch EM	**2529**	2480	**2469**	2164	2163
Adam	2553	2518	OOM	2187	OOM
Mini-batch EM	**2529**	2506	2470	2179	2232
An**em**one (Ours)	2530	**2477**	**2469**	**2158**	**2159**

Table 4: **Negative LLs on the WikiText-103 dataset.** An**em**one achieves the best LLs on three of the four tested model configurations and is highly competitive on the fourth. The best results are marked in bold.

Optimizer	WikiText			
	HMM 256	HMM 512	HMM 1024	Monarch HMM 1024
Full-batch EM	722.6	702.2	682.8	738.1
Adam	735.7	OOM	OOM	OOM
Mini-batch EM	725.2	703.2	**682.1**	734.6
An**em**one (Ours)	**722.2**	**701.7**	682.3	**734.0**

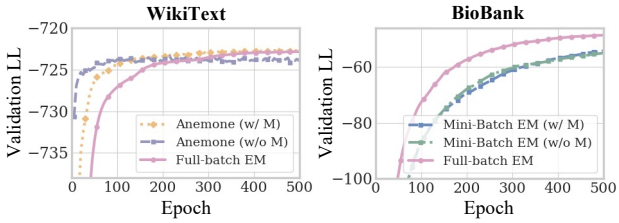

Figure 3: **Ablation study on the effect of momentum when combined with anemone and vanilla mini-batch EM, respectively. Left:** For an**em**one optimizer (HMM with hidden size of 256 on WikiText), incorporating momentum improves the final log-likelihood despite slightly slower initial convergence, while still being significantly faster than full-batch EM. **Right:** In contrast, for mini-batch EM (PDHCLT with hidden size of 512 on BioBank Chr6), momentum provides little benefit.

and scalable PC structures. Specifically, a line of research has sought to design PC structures that are expressive yet parameter-efficient (Rahman et al., 2014; Adel et al., 2015; Liu and Van den Broeck, 2021), while another set of approaches pursues iterative structure learning strategies that progressively expand the model's capacity (Liang et al., 2017; Dang et al., 2022; Di Mauro et al., 2021; Liu et al., 2023b). Both directions have contributed to significant performance gains on widely used text and image datasets.

Parameter Learning of PCs. Beyond structure, a central challenge in learning PCs lies in the optimization of their parameters. This problem has been studied from both the algorithmic and the systems perspective. On the algorithmic side, two families of approaches dominate: EM-style updates and gradient-based methods. The EM algorithm was first applied to PCs by Poon and Domingos (2011), and later extended to mini-batch training in Peharz et al. (2020). Gradient-based optimizers such as Adam (Kingma, 2014) have also become a common choice in practice due to their simplicity and scalability.

On the systems side, considerable effort has gone into developing efficient implementations that can handle the large computational and memory demands of PCs. Optimized einsum backends (Peharz et al., 2020), specialized libraries such as SPFlow (Molina et al., 2019), and more recently PyJuice (Liu et al., 2024) provide high-performance primitives that enable scaling to PCs with billions of parameters.

Conclusion. This work addresses the underperformance of mini-batch optimizers for PCs. We identify that existing methods ineffectively regularize distribution changes, causing them to "overfit" to the current mini-batch. We propose anemone, a novel mini-batch EM algorithm that applies an implicit adaptive learning rate to each parameter, scaled by its contribution to the batch likelihood. Across extensive experiments, anemone consistently outperforms existing optimizers in both convergence speed and final performance.

Acknowledgements

This work was funded in part by the National University of Singapore under its Start-up Grant (Award No: SUG-251RES2505), the DARPA ANSR, CODORD,

and SAFRON programs under awards FA8750-23-2-0004, HR00112590089, and HR00112530141, NSF grant IIS1943641, and gifts from Adobe Research, Cisco Research, Qualcomm, and Amazon. Approved for public release; distribution is unlimited.

References

Adam, K. D. B. J. et al. (2014). A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1412(6).

Adel, T., Balduzzi, D., and Ghodsi, A. (2015). Learning the structure of sum-product networks via an svd-based algorithm. In *UAI*, pages 32–41.

Broadrick, O., Zhang, H., and Van den Broeck, G. (2024). Polynomial semantics of tractable probabilistic circuits. In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, pages 418–429.

Choi, Y., Dang, M., and Van den Broeck, G. (2021). Group fairness by probabilistic modeling with latent fair decisions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12051–12059.

Choi, Y., Vergari, A., and Van den Broeck, G. (2020). Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, page 6.

Dang, M., Liu, A., and Van den Broeck, G. (2022). Sparse probabilistic circuits via pruning and growing. *Advances in Neural Information Processing Systems*, 35:28374–28385.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Di Mauro, N., Gala, G., Iannotta, M., and Basile, T. M. (2021). Random probabilistic circuits. In *Uncertainty in Artificial Intelligence*, pages 1682–1691. PMLR.

Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kunstner, F., Kumar, R., and Schmidt, M. (2021). Homeomorphic-invariance of em: Non-asymptotic convergence in kl divergence for exponential families via mirror descent. In *International Conference on Artificial Intelligence and Statistics*, pages 3295–3303. PMLR.

Liang, Y., Bekker, J., and Van den Broeck, G. (2017). Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*.

Liu, A., Ahmed, K., and Van Den Broeck, G. (2024). Scaling tractable probabilistic circuits: A systems perspective. In *International Conference on Machine Learning*, pages 30630–30646. PMLR.

Liu, A., Niepert, M., and Broeck, G. V. d. (2023a). Image inpainting via tractable steering of diffusion models. *arXiv preprint arXiv:2401.03349*.

Liu, A. and Van den Broeck, G. (2021). Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34:3558–3570.

Liu, X., Liu, A., Van den Broeck, G., and Liang, Y. (2023b). Understanding the distillation process from deep generative models to tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 21825–21838. PMLR.

Loconte, L., Mari, A., Gala, G., Peharz, R., de Campos, C., Quaeghebeur, E., Vessio, G., and Vergari, A. (2024). What is the relationship between tensor factorizations and circuits (and how can we exploit it)? *arXiv preprint arXiv:2409.07953*.

Loconte, L., Mengel, S., and Vergari, A. (2025). Sum of squares circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 19077–19085.

Malvar, H. and Sullivan, G. (2003). Ycocg-r: A color space with rgb reversibility and low dynamic range. *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q*, 6.

Martens, J. and Medabalimi, V. (2014). On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Di Mauro, N., Poupart, P., and Kersting, K. (2019). Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. *arXiv preprint arXiv:1901.03704*.

Peharz, R. (2015). *Foundations of sum-product networks for probabilistic modeling*. PhD thesis, PhD thesis, Medical University of Graz.

Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. (2016). On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K.,

and Ghahramani, Z. (2020). Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR.

Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. (2015). On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR.

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE.

Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019a). Language models are unsupervised multitask learners.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019b). Language models are unsupervised multitask learners.

Rahman, T., Kothalkar, P., and Gogate, V. (2014). Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*, pages 630–645. Springer.

Sudlow, C., Gallacher, J., Allen, N., Beral, V., Burton, P., Danesh, J., Downey, P., Elliott, P., Green, J., Landray, M., Liu, B., Matthews, P., Ong, G., Pell, J., Silman, A., Young, A., Sprosen, T., Peakman, T., and Collins, R. (2015). Uk biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS medicine*, 12:e1001779.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. (2021). A compositional atlas of tractable circuit operations for probabilistic inference. *Advances in Neural Information Processing Systems*, 34:13189–13201.

Zhang, H., Juba, B., and Van den Broeck, G. (2021). Probabilistic generating circuits. In *International conference on machine learning*, pages 12447–12457. PMLR.

Zhang, H., Wang, B., Dang, M., Peng, N., Ermon, S., and Van den Broeck, G. (2025). Scaling up probabilistic circuits via monarch matrices. In *AAAI'25 workshop on CoLoRAI - Connecting Low-Rank Representations in AI*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes/No/Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [No]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Yes]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Rethinking Probabilistic Circuit Parameter Learning: Supplementary Materials

A Structural Properties of PCs

We provide formal definitions of smoothness and decomposability. Please refer to Choi et al. (2020) for a comprehensive overview.

Definition 3 (Smoothness and Decomposability)**.** Define the scope $\texttt{Var}(n)$ of a PC node n as the set of all variables defined by its descendant input nodes. A PC p is smooth if for every sum node n, its children share the same scope: $\forall c_1, c_2 \in \mathsf{ch}(n)$, $\texttt{Var}(c_1) = \texttt{Var}(c_2)$. p is decomposable if for every product node n, its children have disjoint scopes: $\forall c_1, c_2 \in \mathsf{ch}(n)$ $(c_1 \neq c_2)$, $\texttt{Var}(c_1) \cap \texttt{Var}(c_2) = \varnothing$.

B Proofs

This section provides proof of the theoretical results stated in the main paper.

B.1 Interpreting the EM Algorithm of PCs

This section provides the proof of Proposition 1, which interprets the full-batch EM algorithm of PCs in a new context.

Proof of Proposition 1. We begin by formalizing the latent-variable-model view of PCs. Given a PC $p_{\boldsymbol{\phi}}(\mathbf{X})$ parameterized by $\boldsymbol{\phi}$, we define a set of latent variables $\mathbf{Z}$ such that $p_{\boldsymbol{\phi}}(\mathbf{X}) = \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z} = \boldsymbol{z})$. Specifically, we associate a latent variable Z_n with each sum node n in the PC. We use $Z_n = i$ $(i \in \{1, \ldots, |\mathsf{ch}(n)|\})$ to denote that we "select" the i-th child node by zeroing out all the probabilities coming from all other child nodes:

$$p_n(\boldsymbol{x}, \mathbf{Z}_n = i, \boldsymbol{z}_{\backslash n}) = \sum_{c \in \mathsf{ch}(n)} \exp(\phi_{n,c}) \cdot p_c(\boldsymbol{x}, \mathbf{Z}_n = i, \boldsymbol{z}_{\backslash n}) \cdot \mathbb{1}[c = c_i],$$

where we define c_i as the i-th child node of n, and $\mathbf{Z}_{\backslash n} := \mathbf{Z} \backslash Z_n$.

We further show that $p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z})$ is an exponential family distribution. To see this, it suffices to construct a set of $|\boldsymbol{\phi}|$ sufficient statistics $S(\boldsymbol{x}, \boldsymbol{z})$ such that for every $\boldsymbol{x}$ and $\boldsymbol{z}$, the likelihood can be expressed as:

$$p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(\langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi} \rangle - A(\boldsymbol{\phi})\right),$$

where $A(\boldsymbol{\phi}) = \log \sum_{\boldsymbol{x}, \boldsymbol{z}} \langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi} \rangle$ is the log partition function that normalizes the distribution. Note that $A(\boldsymbol{\phi})$ is convex by definition.

To construct $S(\boldsymbol{x}, \boldsymbol{z})$, we first define the *support* $\texttt{supp}(n)$ of every node recursively as follows:

$$\texttt{supp}(n) := \begin{cases} \{(\boldsymbol{x}, \boldsymbol{z}) : p_n(\boldsymbol{x}) > 0\} & n \text{ is an input node,} \\ \bigcap_{c \in \mathsf{ch}(n)} \texttt{supp}(c) & n \text{ is a product node,} \\ \bigcup_{c \in \mathsf{ch}(n)} \left(\{(\boldsymbol{x}, \boldsymbol{z}) : z_n = c\} \cap \texttt{supp}(c)\right) & n \text{ is a sum node,} \end{cases}$$

where $z_n = c$ means $z_n = i$ if c is the i-th child of n.

The sufficient statistics $S(\boldsymbol{x}, \boldsymbol{z})$ can be defined using the support. Specifically, the sufficient statistics corresponding to the parameter $\phi_{n,c}$, denoted $S_{\phi_{n,c}}(\boldsymbol{x}, \boldsymbol{z})$, can be represented as:

$$S_{\phi_{n,c}}(\boldsymbol{x}, \boldsymbol{z}) = \mathbb{1}\Big[(\boldsymbol{x}, \boldsymbol{z}) \in \texttt{supp}(c) \text{ and } z_n = c\Big],$$

where $\mathbb{1}[\cdot]$ is the indicator function.

Before proceeding, we define the Bregman divergence induced by a convex function h as:

$$D_h(\boldsymbol{y}, \boldsymbol{x}) := h(\boldsymbol{y}) - h(\boldsymbol{x}) - \left\langle \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}}, \boldsymbol{y} - \boldsymbol{x} \right\rangle.$$

The following part partially follows Kunstner et al. (2021). We plug in the exponential family distribution form of the PC into the definition of $Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$:

$$\begin{aligned}
Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}') &= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z}), \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{\boldsymbol{z}} p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \left[\left\langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi}' \right\rangle - A(\boldsymbol{\phi}') \right], && \triangleright \text{Definition of } p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z}) \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \left\langle \mathbb{E}_{p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right], \boldsymbol{\phi}' \right\rangle - A(\boldsymbol{\phi}'). && \triangleright \text{Linearity of expectation}
\end{aligned}$$

We then subtract both sides by $Q_{\boldsymbol{\phi}}(\boldsymbol{\phi})$, which is irrelevant to $\boldsymbol{\phi}'$:

$$\begin{aligned}
Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}') - Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}) &= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \left\langle \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right], \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle - A(\boldsymbol{\phi}') + A(\boldsymbol{\phi}), \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \left\langle \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right] - \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle - \underbrace{\left(A(\boldsymbol{\phi}') - A(\boldsymbol{\phi}) - \left\langle \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle \right)}_{D_A(\boldsymbol{\phi}', \boldsymbol{\phi})}, \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \left\langle \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right] - \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle - D_A(\boldsymbol{\phi}', \boldsymbol{\phi}). \qquad (9)
\end{aligned}$$

We continue by simplifying the first term in the above expression. To do this, consider the gradient of $\texttt{LL}(\boldsymbol{\phi})$ w.r.t. $\boldsymbol{\phi}$:

$$\begin{aligned}
\frac{\partial \texttt{LL}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} &= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \frac{\partial \log \left(\sum_{\boldsymbol{z}} \exp \left(\langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi} \rangle \right) \right)}{\partial \boldsymbol{\phi}} - \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{\boldsymbol{z}} \frac{\exp \left(\langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi} \rangle \right) \cdot S(\boldsymbol{x}, \boldsymbol{z})}{\sum_{\boldsymbol{z}'} \exp \left(\langle S(\boldsymbol{x}, \boldsymbol{z}'), \boldsymbol{\phi} \rangle \right)} - \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \\
&= \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}, \boldsymbol{z} \sim p_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})} [S(\boldsymbol{x}, \boldsymbol{z})] - \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}.
\end{aligned}$$

Plug in Equation (9), we have

$$Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}') - Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}) = \left\langle \frac{\partial \texttt{LL}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' - \boldsymbol{\phi} \right\rangle - D_A(\boldsymbol{\phi}', \boldsymbol{\phi}). \qquad (10)$$

We proceed to demonstrate that $D_A(\boldsymbol{\phi}', \boldsymbol{\phi}) = D_{\mathrm{KL}}(p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z}) \,||\, p_{\boldsymbol{\phi}'}(\mathbf{X}, \mathbf{Z}))$, where $D_{\mathrm{KL}}(p \,||\, q)$ is the KL divergence between distributions p and q:

$$\begin{aligned}
D_{\mathrm{KL}}(p_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{Z}) \,||\, p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z})) &= \mathbb{E}_{\boldsymbol{x}, \boldsymbol{z} \sim p_{\boldsymbol{\phi}}} \left[\log \frac{p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\phi}'}(\boldsymbol{x}, \boldsymbol{z})} \right], \\
&= \mathbb{E}_{\boldsymbol{x}, \boldsymbol{z} \sim p_{\boldsymbol{\phi}}} \left[\left\langle S(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{\phi} - \boldsymbol{\phi}' \right\rangle \right] + A(\boldsymbol{\phi}') - A(\boldsymbol{\phi}), \\
&= \left\langle \mathbb{E}_{\boldsymbol{x}, \boldsymbol{z} \sim p_{\boldsymbol{\phi}}} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right], \boldsymbol{\phi} - \boldsymbol{\phi}' \right\rangle + A(\boldsymbol{\phi}') - A(\boldsymbol{\phi}), \\
&= \left\langle \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi} - \boldsymbol{\phi}' \right\rangle + A(\boldsymbol{\phi}') - A(\boldsymbol{\phi}), && \triangleright \text{Since } \frac{\partial A(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} = \mathbb{E}_{\boldsymbol{x}, \boldsymbol{z} \sim p_{\boldsymbol{\phi}}} \left[S(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= D_A(\boldsymbol{\phi}', \boldsymbol{\phi}).
\end{aligned}$$

Plug the result back to Equation (10), we conclude that $Q_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ and the following are equivalent up to a constant independent of $\boldsymbol{\phi}'$:

$$\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}}\left\langle\frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}},\boldsymbol{\phi}'\right\rangle-\mathtt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}'). \tag{11}$$

□

We proceed to prove Lemma 1, which offers a practical way to compute the two key quantities in Equation (11).

Proof of Lemma 1. Recall from our definition that $p_{\boldsymbol{\phi}}(\boldsymbol{x}) := \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})/Z(\boldsymbol{\phi})$. We start by proving a key result: for each parameter $\phi_{n,c}$, the partition function

$$Z(\boldsymbol{\phi}) = \mathtt{TD}(n)\cdot\exp(\phi_{n,c}) + C, \tag{12}$$

where C is independent of $\phi_{n,c}$. Note that by definition $Z(\boldsymbol{\phi})$ is computed by the same feedforward pass shown in Equation (1), with the only difference that the partition function is set to 1 for input nodes. Specifically, denote $Z_n(\boldsymbol{\phi})$ as the partition function of node n, we have

$$Z_n(\boldsymbol{\phi}) = \begin{cases} 1 & n \text{ is an input node,} \\ \prod_{c\in\mathsf{ch}(n)} Z_c(\boldsymbol{\phi}) & n \text{ is a product node,} \\ \sum_{c\in\mathsf{ch}(n)} \exp(\phi_{n,c})\cdot Z_c(\boldsymbol{\phi}) & n \text{ is a sum node.} \end{cases}$$

Define $\mathtt{TD}_m(n)$ as the TD-prob of node n for the PC rooted at m (assume n is a descendant node of m). We prove Equation (12) by induction over m in $Z_m(\phi)$.

In the base case where $m = n$, we have that

$$\begin{aligned} Z_m(\boldsymbol{\phi}) = Z_n(\boldsymbol{\phi}) &= \sum_{c'\in\mathsf{ch}(n)} \exp(\phi_{n,c'})\cdot Z_{c'}(\boldsymbol{\phi}), \\ &= \sum_{c'\in\mathsf{ch}(n)} \exp(\phi_{n,c'}), && \triangleright \text{Since we assume } \forall c, Z_c(\boldsymbol{\phi}) = 1 \\ &= \exp(\phi_{n,c}) + \sum_{c'\in\mathsf{ch}(n),c'\neq c} \exp(\phi_{n,c'}), \\ &= \mathtt{TD}_m(n)\cdot\exp(\phi_{n,c}) + \sum_{c'\in\mathsf{ch}(n),c'\neq c} \exp(\phi_{n,c'}), && \triangleright \text{Since } \forall c, \mathtt{TD}_c(c) = 1 \\ &= \mathtt{TD}_m(n)\cdot\exp(\phi_{n,c}) + C. \end{aligned}$$

Next, assume m is a sum node and Equation (12) holds for all its children. That is,

$$\forall b\in\mathsf{ch}(m),\quad Z_b(\boldsymbol{\phi}) = \mathtt{TD}_b(n)\cdot\exp(\phi_{n,c}) + C.$$

We proceed by plugging in the definition of $Z_m(\boldsymbol{\phi})$:

$$\begin{aligned} Z_m(\boldsymbol{\phi}) &= \sum_{b\in\mathsf{ch}(m)} \exp(\phi_{m,b})\cdot Z_b(\boldsymbol{\phi}), \\ &= \sum_{b\in\mathsf{ch}(m)} \exp(\phi_{m,b})\cdot\mathtt{TD}_b(n)\cdot\exp(\phi_{n,c}) + C. \end{aligned} \tag{13}$$

Denote $\mathcal{A}\subseteq\mathsf{ch}(m)$ as the set of child nodes that are ancestors of n, and $\mathcal{B} = \mathsf{ch}(m)\backslash\mathcal{A}$ is its complement. From the definition of TD-probs, we have

$$\begin{aligned} \mathtt{TD}_m(n) &= \sum_{b\in\mathcal{A}} \mathtt{TD}_m(b)\cdot\mathtt{TD}_b(n), \\ &= \sum_{b\in\mathcal{A}} \exp(\phi_{m,b})\cdot\mathtt{TD}_b(n). && \triangleright \text{Since } \mathtt{TD}_m(b) = \exp(\phi_{m,b}) \end{aligned}$$

Plug in Equation (13), we conclude that

$$\begin{aligned} Z_m(\boldsymbol{\phi}) &= \sum_{b\in\mathcal{A}} \exp(\phi_{m,b})\cdot \texttt{TD}_b(n)\cdot\exp(\phi_{n,c}) + \sum_{b\in\mathcal{B}} \exp(\phi_{m,b})\cdot \texttt{TD}_b(n)\cdot\exp(\phi_{n,c}) + C, \\ &= \texttt{TD}_m(n)\cdot\exp(\phi_{n,c}) + \sum_{b\in\mathcal{B}} \exp(\phi_{m,b})\cdot \texttt{TD}_b(n)\cdot\exp(\phi_{n,c}) + C, \\ &= \texttt{TD}_m(n)\cdot\exp(\phi_{n,c}) + C'. \end{aligned}$$

Finally, if m is a product node such that Equation (12) holds for all its children, we have that

$$Z_m(\boldsymbol{\phi}) = \prod_{b\in\mathsf{ch}(m)} Z_b(\boldsymbol{\phi}).$$

Since m is decomposable (cf. Def. 3), there is at most one $b\in\mathsf{ch}(m)$ that is an ancestor of n (otherwise multiple child nodes contain the variable scope of n). Denote that child node as $\hat{b}$, we further simplify the above equation to

$$Z_m(\boldsymbol{\phi}) = Z_{\hat{b}}(\boldsymbol{\phi}) = \texttt{TD}_{\hat{b}}(n)\cdot\exp(\phi_{n,c}) + C \tag{14}$$

since all other terms are independent of $\phi_{n,c}$ and are assumed to be 1. According to the definition of TD-probs, we have

$$\forall b\in\mathsf{ch}(m),\quad \texttt{TD}_b(n) = \texttt{TD}_m(n). \tag{15}$$

Plug this into Equation (14) gives the desired result:

$$Z_m(\boldsymbol{\phi}) = \texttt{TD}_m(n)\cdot\exp(\phi_{n,c}) + C.$$

This completes the proof of Equation (9).

We continue on proving the first equality in Lemma 1:

$$\begin{aligned} \frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} &= \frac{\partial \log \hat{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} - \frac{\partial \log Z(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}, \\ &= \frac{\partial \log \hat{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} - \frac{1}{Z(\boldsymbol{\phi})}\cdot\frac{\partial Z(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}, \\ &= \frac{\partial \log \hat{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} - \frac{\partial Z(\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}. \end{aligned}$$

According to Equation (12), we can simplify the derivative of $Z(\boldsymbol{\phi})$ with respect to $\phi_{n,c}$ as $\texttt{TD}(n)\cdot\exp(\phi_{n,c}) = \texttt{TD}(\phi_{n,c})$, where the last equality follows from Definition 2. Therefore, we conclude that

$$\frac{\partial \log p_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} = \frac{\partial \log \hat{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial\boldsymbol{\phi}} - \texttt{TD}(\boldsymbol{\phi}).$$

We move on to the second equality in Lemma 1. According to Vergari et al. (2021), $\texttt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ can be computed recursively as follows (define $\texttt{KL}^n_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ as the KLD w.r.t. n):

$$\texttt{KL}^n_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = \begin{cases} 0 & n \text{ is an input node,} \\ \sum_{c\in\mathsf{ch}(n)} \texttt{KL}^c_{\boldsymbol{\phi}}(\boldsymbol{\phi}') & n \text{ is a product node,} \\ \sum_{c\in\mathsf{ch}(n)} \exp(\phi_{n,c})\big(\phi_{n,c} - \phi'_{n,c}\big) + \exp(\phi_{n,c})\cdot\texttt{KL}^c_{\boldsymbol{\phi}}(\boldsymbol{\phi}') & n \text{ is a sum node.} \end{cases} \tag{16}$$

We want to show that for each m that is an ancestor of n, the following holds:

$$\texttt{KL}^m_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = -\texttt{TD}_m(n)\cdot\exp(\phi_{n,c})\cdot\phi'_{n,c} + C, \tag{17}$$

where C is independent of $\phi'_{n,c}$. We can use the exact same induction procedure that is used to prove Equation (12). For all ancestor sum nodes m of n, the first term in Equation (16) (the last row among the three

cases) is always independent of $\phi'_{n,c}$, and hence the recursive definition resembles that of $Z_m(\boldsymbol{\phi})$. Specifically, for all ancestor nodes of n, Equation (17) simplifies to

$$\mathtt{KL}^n_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = \begin{cases} 0 & n \text{ is an input node,} \\ \sum_{c\in\mathsf{ch}(n)} \mathtt{KL}^c_{\boldsymbol{\phi}}(\boldsymbol{\phi}') & n \text{ is a product node,} \\ \sum_{c\in\mathsf{ch}(n)} \exp(\phi_{n,c}) \cdot \mathtt{KL}^c_{\boldsymbol{\phi}}(\boldsymbol{\phi}') & n \text{ is a sum node.} \end{cases}$$

The key difference with $Z_n(\boldsymbol{\phi})$ is the definition of product nodes. Therefore, following the same induction proof of Equation (12), we only need to re-derive the case where m is a product node such that Equation (17) holds for all its children.

Since the PC is decomposable, there is only one child node $b \in \mathsf{ch}(m)$ that is an ancestor of n. Therefore, $\forall c \in \mathsf{ch}(m), c \neq b$, $\mathtt{KL}^c_{\boldsymbol{\phi}}(\boldsymbol{\phi}')$ is independent of $\phi'(n,c)$. Hence, we have

$$\begin{aligned} \mathtt{KL}^m_{\boldsymbol{\phi}}(\boldsymbol{\phi}') &= -\mathtt{TD}_b(n) \cdot \exp(\phi_{n,c}) \cdot \phi'_{n,c} + C, \\ &= -\mathtt{TD}_m(n) \cdot \exp(\phi_{n,c}) \cdot \phi'_{n,c} + C. && \triangleright \text{According to Eq. (15)} \end{aligned}$$

Writing Equation (17) in a vectorized form for every $\phi'_{n,c}$ leads to our final result:

$$\mathtt{KL}_{\boldsymbol{\phi}}(\boldsymbol{\phi}') = -\langle \mathtt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}' \rangle + C.$$

□

B.2 Derivation of the Full-Batch and Mini-Batch EM

Full-Batch EM. Define $\mathcal{S}$ as the set of all sum nodes in the PC, the constrained optimization problem is

$$\begin{aligned} \underset{\boldsymbol{\phi}'}{\text{maximize}} \left\langle \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' \right\rangle, \\ \text{s.t. } \forall n \in \mathcal{S}, \sum_{c\in\mathsf{ch}(n)} \exp(\phi'_{n,c}) = 1. \end{aligned}$$

To incorporate the constraints, we use the method of Lagrange multipliers. The Lagrangian for this problem is

$$\mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}}) = \left\langle \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}}, \boldsymbol{\phi}' \right\rangle + \sum_{n\in\mathcal{S}} \lambda_n \cdot \left(1 - \sum_{c\in\mathsf{ch}(n)} \exp(\phi'_{n,c})\right),$$

where the Lagrange multipliers $\{\lambda_n\}_{n\in\mathcal{S}}$ enforce the constraints.

To minimize the Lagrangian w.r.t. $\boldsymbol{\phi}'$, we take the partial derivative of $\mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}})$ w.r.t. each $\phi'_{n,c}$ and set it to 0:

$$\frac{\partial \mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}})}{\partial \phi'_{n,c}} = \mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c) - \lambda_n \exp(\phi'_{n,c}) = 0,$$

where $\mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c)$ is defined in Section 3.1. Simplifying this equation gives:

$$\phi'_{n,c} = \log \mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c) - \log Z,$$

where $Z = \sum_{c'\in\mathsf{ch}(n)} \mathtt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c')$.

Mini-Batch EM. Similar to the full-batch case, according to Section 3.2, the constrained optimization problem is

$$\begin{aligned} \underset{\boldsymbol{\phi}'}{\text{maximize}} \left\langle \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}} + (\gamma - 1) \cdot \mathtt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}' \right\rangle, \\ \text{s.t. } \forall n \in \mathcal{S}, \sum_{c\in\mathsf{ch}(n)} \exp(\phi'_{n,c}) = 1. \end{aligned}$$

Following the full-batch case, the Lagrangian is given by

$$\mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}}) = \left\langle \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \boldsymbol{\phi}} + (\gamma-1)\cdot \texttt{TD}(\boldsymbol{\phi}), \boldsymbol{\phi}' \right\rangle + \sum_{n\in\mathcal{S}} \lambda_n \cdot \left(1 - \sum_{c\in\mathsf{ch}(n)} \exp(\phi'_{n,c})\right).$$

To minimize the Lagrangian with respect to $\boldsymbol{\phi}'$, we compute the partial derivative of $\mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}})$ w.r.t. each $\phi'_{n,c}$ and set it equal to zero:

$$\frac{\partial \mathcal{L}(\boldsymbol{\phi}', \{\lambda_n\}_{n\in\mathcal{S}})}{\partial \phi'_{n,c}} = \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c) + (\gamma-1)\cdot \texttt{TD}(\phi'_{n,c}) - \lambda_n \exp(\phi'_{n,c}) = 0.$$

Using the definition $\texttt{TD}(\phi'_{n,c}) = \texttt{TD}_{\boldsymbol{\phi}}(n)\cdot\exp(\phi_{n,c})$, the solution is given by

$$\phi'_{n,c} = \log\Big(\texttt{TD}_{\boldsymbol{\phi}}(n)\cdot\exp(\phi_{n,c}) + \alpha\cdot \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c)\Big) - \log Z,$$

where $\alpha := 1/(\gamma-1)$ and $Z = \sum_{c\in\mathsf{ch}(n)} \texttt{TD}_{\boldsymbol{\phi}}(n)\cdot\exp(\phi_{n,c}) + \alpha\cdot\texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n,c)$.

B.3 Decomposition of Parameter Flows

In this section, we show that $\sum_{c\in\mathsf{ch}(n)} \hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n,c) = 1$, where n is a sum node. We start from the definition of $\hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n,c)$:

$$\begin{aligned}
\hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n,c) &= \frac{\partial \log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \phi_{n,c}} = \frac{1}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}\cdot\frac{\partial \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \phi_{n,c}}, \\
&= \frac{\theta_{n,c}}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}\cdot\frac{\partial \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \theta_{n,c}}, && \triangleright \text{By definition } \theta_{n,c} = \exp(\phi_{n,c}) \\
&= \frac{\theta_{n,c}\cdot \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}.
\end{aligned}$$

Now we have

$$\sum_{c\in\mathsf{ch}(n)} \hat{\texttt{F}}^{\boldsymbol{x}}_{\boldsymbol{\phi}}(n,c) = \sum_{c\in\mathsf{ch}(n)} \frac{\theta_{n,c}\cdot\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})} = 1.$$

B.4 Computing TD-Prob Using Auto-Differentiation

In this section, we prove that $\texttt{TD}(n)$ (and thus also $\texttt{TD}(\phi_{n,c})$) can be computed by differentiating $Z_{n_r}(\boldsymbol{\phi})$, where n_r is the root node. Note that the definition of $Z_n(\boldsymbol{\phi})$ follows Appendix B.1.

We proceed with the proof by induction. First, as a base case, we have that

$$\frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_{n_r}(\boldsymbol{\phi})} = 1 = \texttt{TD}(n_r).$$

Next, assume that for a sum/input node n, for all its parents $m \in \mathsf{pa}(n)$ (which are product nodes according to Def. 1) we satisfy that

$$\texttt{TD}(m) = \frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_m(\boldsymbol{\phi})}.$$

We proceed to derive $\partial Z_{n_r}(\boldsymbol{\phi})/\partial Z_n(\boldsymbol{\phi})$:

$$\begin{aligned}
\frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})} &= \sum_{m\in\mathsf{pa}(n)} \frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_m(\boldsymbol{\phi})}\cdot\frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})}, \\
&= \sum_{m\in\mathsf{pa}(n)} \texttt{TD}(m)\cdot\frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})}, \\
&= \sum_{m\in\mathsf{pa}(n)} \texttt{TD}(m). && \triangleright \text{By definition } \frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})} = 1
\end{aligned}$$

The final case is for a product node n, assuming all its parents satisfy the requirement, i.e., $\forall m \in \mathsf{pa}(n), \mathtt{TD}(m) = \partial Z_{n_r}(\boldsymbol{\phi})/\partial Z_m(\boldsymbol{\phi})$. We can simplify the gradient with respect to $Z_n(\boldsymbol{\phi})$ by

$$\begin{aligned}
\frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})} &= \sum_{m \in \mathsf{pa}(n)} \frac{\partial Z_{n_r}(\boldsymbol{\phi})}{\partial Z_m(\boldsymbol{\phi})} \cdot \frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})}, \\
&= \sum_{m \in \mathsf{pa}(n)} \mathtt{TD}(m) \cdot \frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})}, \\
&= \sum_{m \in \mathsf{pa}(n)} \mathtt{TD}(m) \cdot \theta_{m,n}. && \triangleright \text{ By definition } \frac{\partial Z_m(\boldsymbol{\phi})}{\partial Z_n(\boldsymbol{\phi})} = \theta_{m,n}
\end{aligned}$$

C Global Parameter Renormalization of PCs

In this section, we propose a simple renormalization algorithm that takes an unnormalized PC $p_{\boldsymbol{\phi}}(\mathbf{X})$ (i.e., its partition function does not equal 1) with parameters $\boldsymbol{\phi}$ and returns a new set of parameters $\boldsymbol{\phi}'$ such that for each node n in the PC

$$\forall \boldsymbol{x},\ \tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x}) = \frac{1}{Z_n(\boldsymbol{\phi})} \cdot \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x}),$$

where $Z(\boldsymbol{\phi}) := \sum_{\boldsymbol{x}} \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})$ is the partition function of $\tilde{p}^n_{\boldsymbol{\phi}}$.

Analysis. We begin by proving the correctness of the algorithm. Specifically, we show by induction that $\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x}) = \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(n)$ for every n and $\boldsymbol{x}$. In the base case, all input nodes satisfy the equation since they are assumed to be normalized. Next, given a product node n, assume the distributions encoded by all its children c satisfy

$$\forall c \in \mathsf{ch}(n),\ \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x}) = \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(c). \tag{18}$$

Then by definition, $p^n_{\boldsymbol{\phi}'}(\boldsymbol{x})$ can be written as:

$$\begin{aligned}
\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x}) &= \prod_{c \in \mathsf{ch}(n)} \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x}) = \prod_{c \in \mathsf{ch}(n)} \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(c), \\
&= \frac{\prod_{c \in \mathsf{ch}(n)} \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\prod_{c \in \mathsf{ch}(n)} Z_{\boldsymbol{\phi}}(c)}, \\
&= \frac{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{Z_{\boldsymbol{\phi}}(n)}.
\end{aligned}$$

Finally, consider a sum node n whose children satisfy Equation (18). We simplify $\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})$ in the following:

$$\begin{aligned}
\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x}) &= \sum_{c \in \mathsf{ch}(n)} \theta'_{n,c} \cdot \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x}), \\
&= \sum_{c \in \mathsf{ch}(n)} \frac{\theta_{n,c} \cdot Z_{\boldsymbol{\phi}}(c)}{Z_{\boldsymbol{\phi}}(n)} \cdot \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x}), && \triangleright \text{ According to Eq. (8)} \\
&= \sum_{c \in \mathsf{ch}(n)} \frac{\theta_{n,c} \cdot \cancel{Z_{\boldsymbol{\phi}}(c)}}{Z_{\boldsymbol{\phi}}(n)} \cdot \frac{\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\cancel{Z_{\boldsymbol{\phi}}(c)}}, && \triangleright \text{ By induction} \\
&= \frac{\sum_{c \in \mathsf{ch}(n)} \theta_{n,c} \cdot \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{Z_{\boldsymbol{\phi}}(n)}, \\
&= \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(n).
\end{aligned}$$

We proceed to show an interesting property of the proposed global renormalization.

Lemma 2. *Given a PC $p_{\boldsymbol{\phi}}(\mathbf{X})$. Denote $\boldsymbol{\phi}'$ as the parameters returned by the global renormalization algorithm. Then, for every sum edge (n, c), we have*

$$\forall \boldsymbol{x},\ \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\partial \phi'_{n,c}} = \frac{\partial \log \tilde{p}_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \phi_{n,c}}.$$

Proof. We begin by showing that for each node n is one of its children, the following holds:

$$\forall \boldsymbol{x},\ \frac{\partial \log \tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\partial \log \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})} = \frac{\partial \log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \log \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}.$$

If n is a product node, both the left-hand side and the right-hand side equal 1. For example, consider the left-hand side. According to the definition, we have

$$\log \tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x}) = \sum_{c \in \mathsf{ch}(n)} \log \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x}).$$

Hence, its derivative w.r.t. $\log \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})$ is 1.

If n is a sum node, then for each $\boldsymbol{x}$, we have

$$\begin{aligned}
\frac{\partial \log \tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\partial \log \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})} &= \frac{\tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})} \cdot \frac{\partial \tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\partial \tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})}, \\
&= \frac{\tilde{p}^c_{\boldsymbol{\phi}'}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}'}(\boldsymbol{x})} \cdot \theta'_{n,c}, \\
&= \frac{\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(c)}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})/Z_{\boldsymbol{\phi}}(n)} \cdot \theta'_{n,c}, \\
&= \frac{\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})/\cancel{Z_{\boldsymbol{\phi}}(c)}}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})/\cancel{Z_{\boldsymbol{\phi}}(n)}} \cdot \frac{\theta_{n,c} \cdot \cancel{Z_{\boldsymbol{\phi}}(c)}}{\cancel{Z_{\boldsymbol{\phi}}(n)}}, \\
&= \frac{\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})} \cdot \theta_{n,c}, \\
&= \frac{\tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})} \cdot \frac{\partial \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}, \\
&= \frac{\partial \log \tilde{p}^c_{\boldsymbol{\phi}}(\boldsymbol{x})}{\partial \log \tilde{p}^n_{\boldsymbol{\phi}}(\boldsymbol{x})}.
\end{aligned}$$

□

D Additional Experimental Details

D.1 Details about the Datasets and the PC Models

ImageNet32 and The Corresponding PCs. For ImageNet32, we partition every 32×32 image (three color channels) into four 16×16 patches and treat these as individual data samples. There are in total $16 \times 16 \times 3 = 768$ categorical variables in the PC.

We preprocess the data in color space with a lossy transformation, YCoCg, and its scaled, reversible variant, YCoCg-R, proposed by Malvar and Sullivan (2003). Specifically, in YCoCg transformation, given a pixel with RGB values (R, G, B), we first normalize them to the range $[0, 1]$ by

$$r = R/255,\ g = G/255,\ b = B/255.$$

We then apply the following linear transformation:

$$co = r - b,\ tmp = b + co/2,\ cg = g - tmp,\ y = tmp * 2 + cg + 1,$$

where y, co, and cg are all in the range $[-1, 1]$. Finally, we quantize the interval $[-1, 1]$ into 256 bins uniformly and convert y, co, and cg to their quantized version Y, Co, and Cg, respectively. Note that Y, Co, and Cg are all categorical variables with 256 categories.

And the other transformation, YCoCg-R, maps 8-bit integer RGB values to YCoCg values without information loss. Similarly, the forward transformation is given by:

$$co = r - b,\ tmp = b + co/2,\ cg = g - tmp,\ y = tmp + cg/2,$$

where y, co, and cg are also in the range $[-1, 1]$. The resulting integer channels are treated as categorical variables with 512 categories.

We train several deep PC architectures on the ImageNet32 dataset Deng et al. (2009). These include Hidden Chow-Liu Trees (HCLT) (Liu and Van den Broeck, 2021) with hidden size 512 and 1024, and Partitioned Data HCLTs (PDHCLTs) with 256 and 512 latents.[9] The PDHCLT models are configured to partition the input data, which has a shape of (3, 16, 16). They use a maximum of 8 connections between product blocks. Please refer to

WikiText and The Corresponding PCs. We also extend our empirical evaluation to language modeling using the WikiText-103 dataset by Merity et al. (2017). The raw text data is preprocessed into a format suitable for sequence modeling. Specifically, we firstly tokenize the entire corpus using the standard GPT-2 tokenizer by Radford et al. (2019b). All tokenized documents are then concatenated into a single continuous stream of tokens. Finally, this stream is partitioned into non-overlapping sequences of a fixed length of 128 tokens, with any remaining tokens at the end discarded to ensure uniformity across samples. On this preprocessed data, we trained Hidden Markov Models (HMMs), with 256, 512, and 1024 hidden states, and Monarch HMM with a size of 1024 (Zhang et al., 2025).

Biobank dataset and The Corresponding PCs. For our bioinformatics experiments, we use genetic data sourced from the UK Biobank (UKBB) dataset (Sudlow et al., 2015). This specific version focuses on a single Linkage Disequilibrium (LD) block located on chromosome 6. The authors of this dataset remove SNPs with more than 1% missingness and those that deviate significantly from Hardy-Weinberg Equilibrium (1×10^{-7} significance). Further, only individuals with no genetic relatedness to other individuals are considered. We have gotten the approval from the UK BioBank to access this dataset.

D.2 Details about the Optimizers

Full-Batch EM. The full-batch EM implementation follows prior work (e.g., Choi et al. (2021); Peharz et al. (2020)).

Anemone. For notation simplicity, we define $\alpha = \eta/(\eta-1)$. Therefore, we can rewrite Equation (7) equivalently as

$$\theta'_{n,c} = \left((1-\alpha) \cdot \texttt{TD}_{\boldsymbol{\phi}}(n) \cdot \theta_{n,c} + \alpha \cdot \texttt{F}^{\mathcal{D}}_{\boldsymbol{\phi}}(n, c)\right)/Z, \tag{19}$$

which makes it more consistent with the baseline mini-batch EM algorithm.

We performed a preliminary hyperparameter search where we experimented with batch sizes including 512 and 16384. For the learning rate, we tested fixed values of $\alpha \in \{0.1, 0.2, 0.4, 0.6\}$ and also employed a cosine decay schedule. The schedules included decreasing the rate from a base of $\alpha = 0.4$ to a final rate of $\alpha = 0.2$, and from $\alpha = 0.8$ down to $\alpha = 0.6$. We also set a momentum of 0.9 in pratice. We select the final hyperparameter setting based on performance after the first 100 epochs.

Gradient-Based. Following Loconte et al. (2025, 2024), we adopt the Adam optimizer (Kingma, 2014). We selected hyperparameters using a similar search criterion as our EM experiments, testing learning rates of $\{1 \times 10^{-2}, 3 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-3}\}$ and batch sizes of 512 and 1024. On the ImageNet32 YCC dataset, we found a learning rate of 1×10^{-2} performed the best, which aligns with the observations in Loconte et al. (2024). To ensure correctness, we first validated our implementation by reproducing prior results on the MNIST dataset, achieving a log-likelihood of -661.6 after 30 epochs.

[9] The PDHCLT structure is described in Appendix E.

D.3 Details about computing resources

We ran all the experiments included on NVIDIA A40s and NVIDIA GeForce RTX 4090.

D.4 Convergence experiments

We provide a similar experiment to Table 2, to demonstrate the consistency of faster convergence speed on various datasets, as shown in Table 5.

Table 5: **Convergence speed (epochs) for HMM 256 on WikiText.** The table reports epochs to reach specific LL thresholds, with Δ representing the difference from the best LL of -722 (Table 4). Lower is better. Bold marks the best result per column; ∞ indicates failure to reach the threshold in time.

Method	**LL $\geq$ -730** ($\Delta \approx 7.8$)	**LL $\geq$ -724** ($\Delta \approx 1.8$)	**LL $\geq$ -723** ($\Delta \approx 0.8$)
Full EM	60	230	375
Adam	∞	∞	∞
Mini EM	45	∞	∞
An**em**one	**30**	**115**	**275**

E The PDHCLT Structure

We adopt the PDHCLT structure implemented in the PyJuice (Liu et al., 2024) package (`pyjuice.structures.PDHCLT`). In the case of images with size $(3, 16, 16)$, we define a "split interval" to be $(3, 4, 4)$, which means that we partition the image into chunks of size $3 \times 4 \times 4$, resulting in 4×4 chunks. For each chunk, we adopt the HCLT structure, and the PD structure is used to connect the different chunks.

For the BioBank dataset, the sequences have length 1167, and we partition them into chunks of size 128 (the last chunk has a smaller size). Again, HCLT is used to model intra-chunk dependencies and the PD structure is used to capture inter-chunk dependencies.