Neural Network Approximators for Marginal MAP in Probabilistic Circuits

Shivvrat Arya, Tahrima Rahman, Vibhav Gogate

The University of Texas at Dallas
{shivvrat.arya, tahrima.rahman, vibhav.gogate}@utdallas.edu

Abstract

Probabilistic circuits (PCs) such as sum-product networks efficiently represent large multi-variate probability distributions. They are preferred in practice over other probabilistic representations, such as Bayesian and Markov networks, because PCs can solve marginal inference (MAR) tasks in time that scales linearly in the size of the network. Unfortunately, the most probable explanation (MPE) task and its generalization, the marginal maximum-a-posteriori (MMAP) inference task remain NP-hard in these models. Inspired by the recent work on using neural networks for generating near-optimal solutions to optimization problems such as integer linear programming, we propose an approach that uses neural networks to approximate MMAP inference in PCs. The key idea in our approach is to approximate the cost of an assignment to the query variables using a continuous multilinear function and then use the latter as a loss function. The two main benefits of our new method are that it is self-supervised, and after the neural network is learned, it requires only linear time to output a solution. We evaluate our new approach on several benchmark datasets and show that it outperforms three competing linear time approximations: max-product inference, max-marginal inference, and sequential estimation, which are used in practice to solve MMAP tasks in PCs.

Introduction

Probabilistic circuits (PCs) (Choi, Vergari, and Van den Broeck 2020) such as sum-product networks (SPNs) (Poon and Domingos 2011), arithmetic circuits (Darwiche 2003), AND/OR graphs (Dechter and Mateescu 2007), cutset networks (Rahman, Kothalkar, and Gogate 2014), and probabilistic sentential decision diagrams (Kisa et al. 2014) represent a class of *tractable probabilistic models* which are often used in practice to compactly encode a large multi-dimensional joint probability distribution. Even though all of these models admit linear time computation of marginal probabilities (MAR task), only some of them (Vergari et al. 2021; Peharz 2015), specifically those without any latent variables or having specific structural properties, e.g., cutset networks, selective SPNs (Peharz et al. 2016), AND/OR graphs having small contexts, etc., admit tractable most probable explanation (MPE) inference[1].

However, none of these expressive PCs can efficiently solve the *marginal maximum-a-posteriori (MMAP) task* (Peharz 2015; Vergari et al. 2021), a task that combines MAR and MPE inference. More specifically, the distinction between MPE and MMAP tasks is that, given observations over a subset of variables (evidence), the MPE task aims to find the most likely assignment to all the non-evidence variables. In contrast, in the MMAP task, the goal is to find the most likely assignment to a subset of non-evidence variables known as the query variables, while marginalizing out non-evidence variables that are not part of the query. The MMAP problem has numerous real-world applications, especially in health care, natural language processing, computer vision, linkage analysis and diagnosis where hidden variables are present and need to be marginalized out (Bioucas-Dias and Figueiredo 2016; Kiselev and Poupart 2014; Lee, Marinescu, and Dechter 2014; Ping, Liu, and Ihler 2015).

In terms of computational complexity, both MPE and MMAP tasks are at least NP-hard in SPNs, a popular class of PCs (Peharz 2015; Conaty, de Campos, and Mauá 2017). Moreover, it is also NP-hard to approximate MMAP in SPNs to $2^{n^{\delta}}$ for fixed $0 \leq \delta < 1$, where n is the input size (Conaty, de Campos, and Mauá 2017; Mei, Jiang, and Tu 2018). It is also known that the MMAP task is much harder than the MPE task and is NP-hard even on models such as cutset networks and AND/OR graphs that admit linear time MPE inference (Park and Darwiche 2004; de Campos 2011).

To date, both exact and approximate methods have been proposed in literature for solving the MMAP task in PCs. Notable exact methods include branch-and-bound search (Mei, Jiang, and Tu 2018), reformulation approaches which encode the MMAP task as other combinatorial optimization problems with widely available solvers (Mauá et al. 2020) and circuit transformation and pruning techniques (Choi, Friedman, and Van den Broeck 2022). These methods can be quite slow in practice and are not applicable when fast, real-time inference is desired. As a result, approximate approaches that require only a few passes over the PC are often used in practice. A popular approximate approach is to

[1]The MPE inference task is also called full maximum-a-posteriori (full MAP) inference in literature. In this paper, we adopt the convention of calling it MPE.

compute an MPE solution over both the query and unobserved variables and then project the MPE solution over the query variables (Poon and Domingos 2011; Rahman, Jin, and Gogate 2019). Although this approach can provide fast answers at query time, it often yields MMAP solutions that are far from optimal.

In this paper, we propose to address the limitations of existing approximate methods for MMAP inference in PCs by using neural networks (NNs), leveraging recent work in the *learning to optimize* literature (Li and Malik 2016; Fioretto, Mak, and Hentenryck 2020; Donti, Rolnick, and Kolter 2020; Zamzam and Baker 2020; Park and Hentenryck 2023). In particular, several recent works have shown promising results in using NNs to solve both constrained and unconstrained optimization problems (see Park and Hentenryck (2023) and the references therein).

The high-level idea in these works is the following: given data, train NNs, either in a supervised or self-supervised manner, and then use them at test time to predict high-quality, near-optimal solutions to future optimization problems. A number of reasons have motivated this idea of *learning to optimize* using NNs: 1) NNs are good at approximating complex functions (distributions), 2) once trained, they can be faster at answering queries than search-based approaches, and 3) with ample data, NNs can learn accurate mappings of inputs to corresponding outputs. This has led researchers to employ NNs to approximately answer probabilistic inference queries such as MAR and MPE in Bayesian and Markov networks (Yoon et al. 2019; Cui et al. 2022). To the best of our knowledge, there is no prior work on solving MMAP in BNs, MNs, or PCs using NNs.

This paper makes the following contributions. First, we propose to learn a neural network (NN) approximator for solving the MMAP task in PCs. Second, by leveraging the tractability of PCs, we devise a loss function whose gradient can be computed in time that scales linearly in the size of the PC, allowing fast gradient-based algorithms for learning NNs. Third, our method trains an NN in a self-supervised manner without having to rely on pre-computed solutions to arbitrary MMAP problems, thus circumventing the need to solve intractable MMAP problems in practice. Fourth, we demonstrate via a large-scale experimental evaluation that our proposed NN approximator yields higher quality MMAP solutions as compared to existing approximate schemes.

Preliminaries

We use upper case letters (e.g., X) to denote random variables and corresponding lower case letters (e.g., x) to denote an assignment of a value to a variable. We use bold upper case letters (e.g., $\mathbf{X}$) to denote a set of random variables and corresponding bold lower case letters (e.g., $\mathbf{x}$) to denote an assignment of values to all variables in the set. Given an assignment $\mathbf{x}$ to all variables in $\mathbf{X}$ and a variable $Y \in \mathbf{X}$, let $\mathbf{x}_Y$ denote the projection of $\mathbf{x}$ on Y. We assume that all random variables take values from the set $\{0, 1\}$; although note that it is easy to extend our method to multi-valued variables.

Probabilistic Circuits

A probabilistic circuit (PC) $\mathcal{M}$ (Choi, Vergari, and Van den Broeck 2020) defined over a set of variables $\mathbf{X}$ represents a joint probability distribution over $\mathbf{X}$ using a rooted directed acyclic graph. The graph consists of three types of nodes: internal sum nodes that are labeled by $+$, internal product nodes that are labeled by $\times$, and leaf nodes that are labeled by either X or $\neg X$ where $X \in \mathbf{X}$. Sum nodes represent conditioning, and an edge into a sum node n from its child node m is labeled by a real number $\omega(m, n) > 0$. Given an internal node (either a sum or product node) n, let $\texttt{ch}(n)$ denote the set of children of n. We assume that each sum node n is normalized and satisfies the following property: $\sum_{m \in \texttt{ch}(n)} \omega(m, n) = 1$.

In this paper, we focus on a class of PCs which are *smooth and decomposable* (Choi, Vergari, and Van den Broeck 2020; Vergari et al. 2021). Examples of such PCs include sum-product networks (Poon and Domingos 2011; Rahman and Gogate 2016b), mixtures of cutset networks (Rahman, Kothalkar, and Gogate 2014; Rahman and Gogate 2016a), and arithmetic circuits obtained by compiling probabilistic graphical models (Darwiche 2003). These PCs admit tractable marginal inference, a key property that we leverage in our proposed method.

Definition 1. *We say that a sum or a product node n is defined over a variable X if there exists a directed path from n to a leaf node labeled either by X or $\neg X$. A PC is **smooth** if each sum node is such that its children are defined over the same set of variables. A PC is **decomposable** if each product node is such that its children are defined over disjoint subsets of variables.*

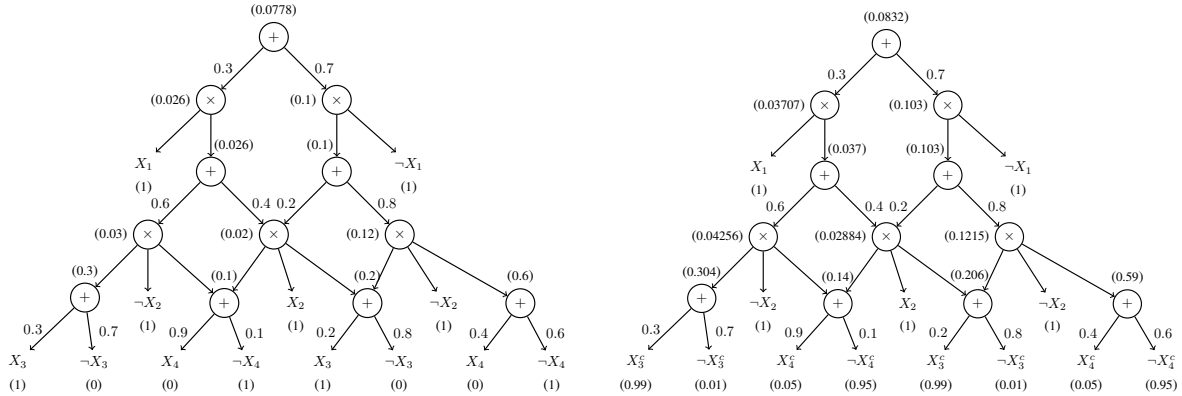
Example 1. *Figure 1(a) shows a smooth and decomposable probabilistic circuit defined over $\mathbf{X} = \{X_1, \ldots, X_4\}$.*

Marginal Inference in PCs

Next, we describe how to compute the probability of an assignment to a subset of variables in a smooth and decomposable PC. This task is called the marginal inference (MAR) task. We begin by describing some additional notation.

Given a PC $\mathcal{M}$ defined over $\mathbf{X}$, let $\mathcal{S}$, $\mathcal{P}$ and $\mathcal{L}$ denote the set of sum, product and leaf nodes of $\mathcal{M}$ respectively. Let $\mathbf{Q} \subseteq \mathbf{X}$. Given a node m and an assignment $\mathbf{q}$, let $v(m, \mathbf{q})$ denote the value of m given $\mathbf{q}$. Given a leaf node n, let $\texttt{var}(n)$ denote the variable associated with n and let $l(n, \mathbf{q})$ be a function, which we call *leaf function*, that is defined as follows. $l(n, \mathbf{q})$ equals 0 if any of the following two conditions are satisfied: (1) the label of n is Q where $Q \in \mathbf{Q}$ and $\mathbf{q}$ contains the assignment $Q = 0$; and (2) if the label of n is $\neg Q$ and $\mathbf{q}$ contains the assignment $Q = 1$. Otherwise, it is equal to 1. Intuitively, the leaf function assigns all leaf nodes that are inconsistent with the assignment $\mathbf{q}$ to 0 and the remaining nodes, namely those that are consistent with $\mathbf{q}$ and those that are not part of the query to 1.

Under this notation, and given a leaf function $l(n, \mathbf{q})$, the marginal probability of any assignment $\mathbf{q}$ w.r.t $\mathcal{M}$, and denoted by $\mathrm{p}_{\mathcal{M}}(\mathbf{q})$ can be computed by performing the follow-



(a) PC (b) QPC

Figure 1: (a) An example smooth and decomposable PC. The figure also shows value computation for answering the query $\mathrm{p}_{\mathcal{M}}(X_3 = 1, X_4 = 0)$. The values of the leaf, sum, and product nodes are given in parentheses on their bottom, top, and left, respectively. The value of the root node is the answer to the query. (b) QPC obtained from the PC given in (a) for query variables $\{X_3, X_4\}$. For simplicity, here, we use an MMAP problem without any evidence. This is because a given evidence can be incorporated into the PC by appropriately setting the leaf nodes. We also show value computations for the following leaf initialization: $X_3^c = 0.99, \neg X_3^c = 0.01, X_4^c = 0.05, \neg X_4^c = 0.95$ and all other leaves are set to 1.

ing recursive *value computations*:

$$v(n, \mathbf{q}) = \begin{cases} l(n, \mathbf{q}) & \text{if } n \in \mathcal{L} \\ \sum_{m \in \mathtt{ch}(n)} \omega(m, n) v(m, \mathbf{q}) & \text{if } n \in \mathcal{S} \\ \prod_{m \in \mathtt{ch}(n)} v(m, \mathbf{q}) & \text{if } n \in \mathcal{P} \end{cases} \quad (1)$$

Let r denote the root node of $\mathcal{M}$. Then, the probability of $\mathbf{q}$ w.r.t. $\mathcal{M}$, denoted by $\mathrm{p}_{\mathcal{M}}(\mathbf{q})$ equals $v(r, \mathbf{q})$. Note that if $\mathbf{Q} = \mathbf{X}$, then $v(r, \mathbf{x})$ denotes the probability of the joint assignment $\mathbf{x}$ to all variables in the PC. Thus

$$v(r, \mathbf{q}) = \sum_{\mathbf{y} \in \{0,1\}^{|\mathbf{Y}|}} v(r, (\mathbf{q}, \mathbf{y}))$$

where $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Q}$ and the notation $(\mathbf{q}, \mathbf{y})$ denotes the *composition* of the assignments to $\mathbf{Q}$ and $\mathbf{Y}$ respectively.

Since the recursive value computations require only one bottom-up pass over the PC, MAR inference is tractable or linear time in smooth and decomposable PCs.

Example 2. *Figure 1(a) shows bottom-up, recursive value computations for computing the probability of the assignment* $(X_3 = 1, X_4 = 0)$ *in our running example. Here, the leaf nodes* $\neg X_3$ *and* X_4 *are assigned to* 0 *and all other leaf nodes are assigned to* 1*. The number in parentheses at the top, left, and bottom of each sum, product and leaf nodes respectively shows the value of the corresponding node. The value of the root node equals* $p_{\mathcal{M}}(X_3 = 1, X_4 = 0)$.

Marginal Maximum-a-Posteriori (MMAP) Inference in PCs

Given a PC $\mathcal{M}$ defined over $\mathbf{X}$, let $\mathbf{E} \subseteq \mathbf{X}$ and $\mathbf{Q} \subseteq \mathbf{X}$ denote the set of evidence and query variables respectively such that $\mathbf{E} \cap \mathbf{Q} = \emptyset$. Let $\mathbf{H} = \mathbf{X} \setminus (\mathbf{Q} \cup \mathbf{E})$ denote the set of hidden variables. Given an assignment $\mathbf{e}$ to the evidence variables (called evidence), the MMAP task seeks to find an assignment $\mathbf{q}$ to $\mathbf{Q}$ such that the probability of the assignment $(\mathbf{e}, \mathbf{q})$ w.r.t. $\mathcal{M}$ is maximized. Mathematically,

$$\mathtt{MMAP}(\mathbf{Q}, \mathbf{e}) = \operatorname*{argmax}_{\mathbf{q}} \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}) \quad (2)$$

$$= \operatorname*{argmax}_{\mathbf{q}} \sum_{\mathbf{h} \in \{0,1\}^{|\mathbf{H}|}} \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}, \mathbf{h}) \quad (3)$$

If $\mathbf{H} = \emptyset$ (namely $\mathbf{Q}$ is the set of non-evidence variables), then MMAP corresponds to the most probable explanation (MPE) task. It is known that both MMAP and MPE tasks are at least NP-hard in smooth and decomposable PCs (Park and Darwiche 2004; de Campos 2011; Peharz 2015), and even NP-hard to approximate (Conaty, de Campos, and Mauá 2017; Mei, Jiang, and Tu 2018).

A popular approach to solve the MMAP task in PCs is to replace the sum ($\sum$) operator with the max operator during bottom-up, recursive value computations and then performing a second top-down pass to find the assignment (Poon and Domingos 2011).

A Neural Optimizer for MMAP in PCs

In this section, we introduce a learning-based approach using deep neural networks (NNs) to approximately solve the MMAP problem in PCs. Formally, the NN represents a function $f_\theta(.)$ that is parameterized by θ, and takes an assignment $\mathbf{e}$ over the evidence variables as input and outputs an assignment $\mathbf{q}$ over the query variables. Our goal is to design *generalizable, continuous loss functions* for updating the parameters of the NN such that once learned, at test time, given an assignment $\mathbf{e}$ to the evidence variables as input, the NN outputs near-optimal solutions to the MMAP problem.

In this paper, we assume that the sets of evidence $\left(\mathbf{E} = \{E_i\}_{i=1}^N\right)$ and query $\left(\mathbf{Q} = \{Q_j\}_{j=1}^M\right)$ variables are known *a priori* and do not change at both training and test

time. We leave as future work the generalization of our approach that can handle variable length, arbitrarily chosen evidence, and query sets. Also, note that our proposed method does not depend on the particular NN architecture used, and we only require that each output node is a continuous quantity in the range $[0, 1]$ and uses a differentiable activation function (e.g., the sigmoid function).

We can learn the parameters of the given NN either in a *supervised* manner or in a *self-supervised* manner. However, the supervised approach is impractical, as described below.

In the supervised setting, we assume that we are given training data $\mathcal{D} = \{\langle \mathbf{e}_1, \mathbf{q}_1^* \rangle, \ldots, \langle \mathbf{e}_d, \mathbf{q}_d^* \rangle\}$, where each input $\mathbf{e}_i$ is an assignment to the evidence variables, and each (label) $\mathbf{q}_i^*$ is an optimal solution to the corresponding MMAP task, namely $\mathbf{q}_i^* = \texttt{MMAP}(\mathbf{Q}, \mathbf{e}_i)$. We then use supervised loss functions such as the mean-squared-error (MSE) $\sum_{i=1}^{d} \|\mathbf{q}_i^* - \mathbf{q}_i^c)\|_2^2/d$ and the mean-absolute-error (MAE) $\sum_{i=1}^{d} \|\mathbf{q}_i^* - \mathbf{q}_i^c\|_1/d$ where $\mathbf{q}_i^c$ is the predicted assignment (note that $\mathbf{q}_i^c$ is continuous), and standard gradient-based methods to learn the parameters. Although supervised approaches allow us to use simple-to-implement loss functions, they are *impractical* if the number of query variables is large because they require access to the exact solutions to several intractable MMAP problems[2]. We therefore propose to use a *self-supervised approach*.

A Self-Supervised Loss Function for PCs

In the self-supervised setting, we need access to training data in the form of assignments to the evidence variables, i.e., $\mathcal{D}' = \{\mathbf{e}_1, \ldots, \mathbf{e}_d\}$. Since smooth and decomposable PCs admit perfect sampling, these assignments can be easily sampled from the PC via top-down AND/OR sampling (Gogate and Dechter 2012). The latter yields an assignment $\mathbf{x}$ over all the random variables in the PC. Then we simply project $\mathbf{x}$ on the evidence variables $\mathbf{E}$ to yield a training example $\mathbf{e}$. Because each training example can be generated in time that scales linearly with the size of the PC, in practice, our proposed self-supervised approach is likely to have access to much larger number of training examples compared to the supervised approach.

Let $\mathbf{q}^c$ denote the MMAP assignment predicted by the NN given evidence $\mathbf{e} \in \mathcal{D}'$ where $\mathbf{q}^c \in [0, 1]^M$. In MMAP inference, given $\mathbf{e}$, we want to find an assignment $\mathbf{q}$ such that $\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$ is maximized, namely, $-\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$ is minimized. Thus, a natural loss function that we can use is $-\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$. Unfortunately, the NN outputs a continuous vector $\mathbf{q}^c$ and as a result $\mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}^c)$ is not defined. Therefore, we cannot use $-\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}^c)$ as a loss function.

One approach to circumvent this issue is to use a threshold (say 0.5) to convert each continuous quantity in the range [0,1] to a binary one. A problem with this approach is that the threshold function is not differentiable.

Therefore, we propose to construct a smooth, differentiable loss function that given $\mathbf{q}^c = (q_1^c, \ldots, q_M^c)$ approximates $-\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$ where $\mathbf{q} = (q_1 = [q_1^c > 0.5], \ldots, q_M = [q_M^c > 0.5])$ and $[q_i^c > 0.5]$ is an indicator function which is 1 if $q_i^c > 0.5$ and 0 otherwise. The key idea in our approach is to construct a new PC, which we call *Query-specific PC* (QPC) by replacing all binary leaf nodes associated with the query variables in the original PC, namely those labeled by Q and $\neg Q$ where $Q \in \mathbf{Q}$, with continuous nodes $Q^c \in [0, 1]$ and $\neg Q^c \in [0, 1]$. Then our proposed loss function is obtained using value computations (at the *root node* of the QPC) via a simple modification of the *leaf function* of the PC. At a high level, our new leaf function assigns each leaf node labeled by Q_j^c such that $Q_j \in \mathbf{Q}$ to its corresponding estimate q_j^c, obtained from the NN and each leaf node labeled by $\neg Q_j^c$ such that $Q_j \in \mathbf{Q}$ to $1 - q_j^c$.

Formally, for the QPC, we propose to use leaf function $l'(n, (\mathbf{e}, \mathbf{q}^c))$ defined as follows:

1. If the label of n is Q_j^c such that $Q_j \in \mathbf{Q}$ then $l'(n, (\mathbf{e}, \mathbf{q}^c)) = q_j^c$.
2. If n is labeled by $\neg Q_j^c$ such that $Q_j \in \mathbf{Q}$ then $l'(n, (\mathbf{e}, \mathbf{q}^c)) = 1 - q_j^c$.
3. If n is labeled by E_k such that $E_k \in \mathbf{E}$ and the assignment $E_k = 0$ is in $\mathbf{e}$ then $l'(n, (\mathbf{e}, \mathbf{q}^c)) = 0$.
4. If n is labeled by $\neg E_k$ such that $E_k \in \mathbf{E}$ and the assignment $E_k = 1$ is in $\mathbf{e}$ then $l'(n, (\mathbf{e}, \mathbf{q}^c)) = 0$.
5. If conditions (1)-(4) are not met then $l'(n, (\mathbf{e}, \mathbf{q}^c)) = 1$.

The value of each node n in the QPC, denoted by $v'(n, (\mathbf{e}, \mathbf{q}^c))$ is given by a similar recursion to the one given in Eq. (1) for PCs, except that the leaf function $l(n, \mathbf{q})$ is replaced by the new (continuous) leaf function $l'(n, (\mathbf{e}, \mathbf{q}^c))$. Formally, $v'(n, (\mathbf{e}, \mathbf{q}^c))$ is given by

$$v'(n, (\mathbf{e}, \mathbf{q}^c)) = \begin{cases} l'(n, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{L} \\ \sum_{m \in \mathrm{ch}(n)} \omega(m, n) v'(m, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{S} \\ \prod_{m \in \mathrm{ch}(n)} v'(n, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{P} \end{cases} \quad (4)$$

Let r denote the root node of $\mathcal{M}$, then we propose to use $-\ln v'(r, (\mathbf{e}, \mathbf{q}^c))$ as a loss function.

Example 3. *Figure 1(b) shows the QPC corresponding to the PC shown in Figure 1(a). We also show value computations for the assignment* $(X_3^c = 0.99, X_4^c = 0.05)$.

Tractable Gradient Computation

Our proposed loss function is smooth and continuous because by construction, it is a negative logarithm of a *multilinear function* over $\mathbf{q}^c$. Next, we show that the partial derivative of the function w.r.t. q_j^c can be computed in linear time in the size of the QPC[3]. More specifically, in order to compute the partial derivative of QPC with respect to $\mathbf{q}_j^c$, we simply have to use a new leaf function which is identical to l' except that if the label of a leaf node n is Q_j^c then we set its value to 1 (instead of q_j^c) and if it is $\neg Q_j^c$ then we set its value -1 (instead of $1 - q_j^c$). We then perform bottom-up recursive

[2]Note that the training data used to train the NN in the supervised setting is different from the training data used to learn the PC. In particular, in the data used to train the PC, the assignments to the query variables $\mathbf{Q}$ may not be optimal solutions of $\texttt{MMAP}(\mathbf{Q}, \mathbf{e})$.

[3]Recall that q_j^c is an output node of the NN and therefore back-propagation over the NN can be performed in time that scales linearly with the size of the NN and the QPC

value computations over the QPC and the value of the root node is the partial derivative of the QPC with respect to $\mathbf{q}_j^c$. In summary, it is straight-forward to show that:

Proposition 1. *The gradient of the loss function* $-\ln v'(r,(\mathbf{e},\mathbf{q}^c))$ *w.r.t.* $\mathbf{q}_j^c$ *can be computed in time and space that scales linearly with the size of* $\mathcal{M}$.

Example 4. *The partial derivative of the QPC given in figure 1(b) w.r.t.* x_3^c *given* $(X_3^c = 0.99, X_4^c = 0.05)$ *can be obtained by setting the leaf nodes* X_3^c *to* 1 *and* $\neg X_3^c$ *to* -1*, assigning all other leaves to the values shown in Figure 1(b) and then performing value computations. After the value computation phase, the value of the root node will equal the partial derivative of the QPC w.r.t.* x_3^c.

Improving the Loss Function

As mentioned earlier, our proposed loss function is a continuous approximation of the discrete function $-\ln v(r,(\mathbf{e},\mathbf{q}))$ where $\mathbf{q} = (q_1 = [q_1^c > 0.5], \ldots, q_M = [q_M^c > 0.5])$ and the difference between the two is minimized iff $\mathbf{q} = \mathbf{q}^c$. Moreover, since the set of continuous assignments includes the discrete assignments, it follows that:

$$\min_{\mathbf{q}^c}\{-\ln v'(r,(\mathbf{e},\mathbf{q}^c))\} \leq \min_{\mathbf{q}}\{-\ln v(r,(\mathbf{e},\mathbf{q}))\} \quad (5)$$

Since the right-hand side of the inequality given in (5) solves the MMAP task, we can improve our loss function by tightening the lower bound. This can be accomplished using an entropy-based penalty, controlled by a hyper-parameter $\alpha > 0$, yielding the loss function

$$\ell(\mathbf{q}^c) = -\ln v'(r,(\mathbf{e},\mathbf{q}^c)) - \alpha\sum_{j=1}^{M} q_j^c\log(q_j^c) + (1-q_j^c)\log(1-q_j^c) \quad (6)$$

The second term in the expression given above is minimized when each q_j^c is closer to 0 or 1 and is maximized when $q_j^c = 0.5$. Therefore, it encourages 0/1 (discrete) solutions. The hyperparameter α controls the magnitude of the penalty. When $\alpha = 0$, the above expression finds an assignment based on the continuous approximation $-\ln v'(r,(\mathbf{e},\mathbf{q}^c))$. On the other hand, when $\alpha = \infty$ then only discrete solutions are possible yielding a non-smooth loss function. α thus helps us trade the smoothness of our proposed loss function with its distance to the true loss.

Experiments

In this section, we describe and analyze the results of our comprehensive experimental evaluation for assessing the performance of our novel Self-Supervised learning based MMAP solver for PCs, referred to as SSMP hereafter. We begin by describing our experimental setup including competing methods, evaluation criteria, as well as NN architectures, datasets, and PCs used in our study.

Competing Methods

We use *three polytime baseline methods* from the PC and probabilistic graphical models literature (Park and Darwiche 2004; Poon and Domingos 2011). We also compared the impact of using the solutions computed by the three baseline schemes as well our method SSMP as initial state for stochastic hill climbing search.

Baseline 1: MAX Approximation (`Max`). In this scheme (Poon and Domingos 2011), the MMAP assignment is derived by substituting sum nodes with max nodes. During the upward pass, a max node produces the maximum weighted value from its children instead of their weighted sum. Subsequently, the downward pass begins from the root and iteratively selects the highest-valued child of a max node (or one of them), along with all children of a product node.

Baseline 2: Maximum Likelihood Approximation (`ML`) (Park and Darwiche 2004) For each variable $Q \in \mathbf{Q}$, we first compute the marginal distribution $\mathrm{p}_{\mathcal{M}}(Q|\mathbf{e})$ and then set Q to $\mathrm{argmax}_{j\in\{0,1\}}\,\mathrm{p}_{\mathcal{M}}(Q=j|\mathbf{e})$.

Baseline 3: Sequential Approximation (`Seq`) In this scheme (Park and Darwiche 2004), we assign the query variables one by one until no query variables remain unassigned. At each step, we choose an unassigned query variable $Q_j \in \mathbf{Q}$ that maximizes the probability $\mathrm{p}_{\mathcal{M}}(q_j|\mathbf{e},\mathbf{y})$ for one of its values q_j and assign it to q_j where $\mathbf{y}$ represents the assignment to the previously considered query variables.

Stochastic Hill Climbing Search. We used the three baselines and our SSMP method as the initial state in stochastic hill climbing search for MMAP inference described in (Park and Darwiche 2004). The primary goal of this experiment is to assess whether our scheme can assist local search-based *anytime methods* in reaching better solutions than other heuristic methods for initialization. In our experiments, we ran stochastic hill climbing for 100 iterations for each MMAP problem.

Evaluation Criteria

We evaluated the performance of the competing schemes along two dimensions: log-likelihood scores and inference times. Given evidence $\mathbf{e}$ and query answer $\mathbf{q}$, the log-likelihood score is given by $\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e},\mathbf{q})$.

Datasets and Probabilistic Circuits

We use twenty-two widely used binary datasets from the tractable probabilistic models' literature (Lowd and Davis 2010; Haaren and Davis 2012; Larochelle and Murray 2011; Bekker et al. 2015) (we call them TPM datasets) as well as the binarized MNIST (Salakhutdinov and Murray 2008), EMNIST (Cohen et al. 2017) and CIFAR-10 (Krizhevsky, Nair, and Hinton 2009) datasets. We used the DeeProb-kit library (Loconte and Gala 2022) to learn a sum-product network (our choice of PC) for each dataset. The number of nodes in these learned PCs ranges from 46 to 22027.

For each PC and each test example in the 22 TPM datasets, we generated two types of MMAP instances: MPE instances in which $\mathbf{H}$ is empty and MMAP instances in which $\mathbf{H}$ is not empty. We define query ratio, denoted by qr, as the fraction of variables that are part of the query set. For MPE, we selected qr from $\{0.1, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9\}$, and for MMAP, we replaced 0.9 with 0.4 to avoid small $\mathbf{H}$ and $\mathbf{E}$. For generating MMAP instances, we used 50% of

	Initial								Hill Climbing Search							
	MPE				MMAP				MPE				MMAP			
	Max	SSMP	ML	Seq	Max	SSMP	ML	Seq	Max	SSMP	ML	Seq	Max	SSMP	ML	Seq
Max	0	64	33	14	0	46	23	10	0	40	13	9	0	27	10	16
SSMP	88	0	96	77	97	0	102	82	93	0	99	87	98	0	100	86
ML	6	49	0	15	3	34	0	10	19	37	0	14	12	26	0	17
Seq	105	63	105	0	117	53	117	0	85	44	82	0	89	39	90	0

Table 1: Contingency tables for competing methods across MPE and MMAP Problems, including initial and Hill Climbing Search comparisons. Highlighted values represent results for SSMP.

the remaining variables as evidence variables (and for MPE instances all remaining variables are evidence variables).

For the MNIST, EMNIST, and CIFAR-10 datasets, we used $qr = 0.7$ and generated MPE instances only. More specifically, we used the top 30% portion of the image as evidence, leaving the bottom 70% portion as query variables. Also, in order to reduce the training time for PCs, note that for these datasets, we learned a PC for each class, yielding a total of ten PCs for each dataset.

Neural Network Optimizers

For each PC and query ratio combination, we trained a corresponding neural network (NN) using the loss function described in the previous section. Because we have 22 TPM datasets and 7 query ratios for them, we trained 154 NNs for the MPE task and 154 for the MMAP task. For the CIFAR-10, MNIST and EMNIST datasets, we trained 10 NNs, one for each PC (recall that we learned a PC for each class).

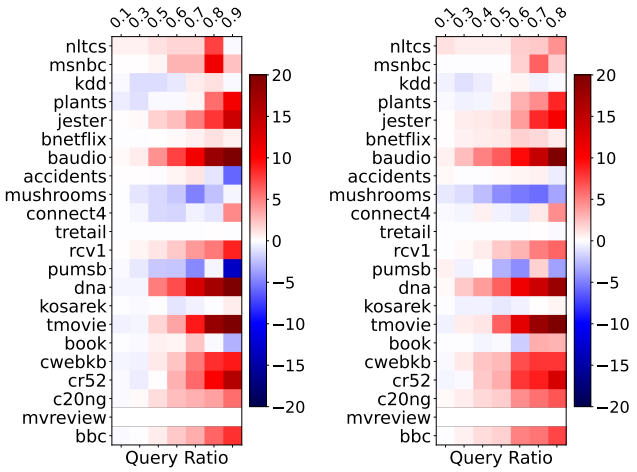
Because our learning method does not depend on the specific choice of neural network architectures, we use a fixed neural network architecture across all experiments: fully connected with four hidden layers having 128, 256, 512, and 1024 nodes respectively. We used ReLU activation in the hidden layers, sigmoid in the output layer, dropout for regularization (Srivastava et al. 2014) and Adam optimizer (Kingma and Ba 2017) with a standard learning rate scheduler for 50 epochs. All NNs were trained using PyTorch (Paszke et al. 2019) on a single NVIDIA A40 GPU. We select a value for the hyperparameter α used in our loss function (see equation (6)) via 5-fold cross validation.

Results on the TPM Datasets

We summarize our results for the competing schemes (3 baselines and SSMP) on the 22 TPM datasets using the first two contingency tables given in Table 1, one for MPE and one for MMAP. Detailed results are provided in the supplementary material. Recall that we generated 154 test datasets each for MPE and MMAP (22 PCs $\times$ 7 qr values). In all contingency tables, the number in the cell (i, j) equals the number of times (out of 154) that the scheme in the i-th row was better in terms of average log-likelihood score than the scheme in the j-th column. The difference between 154 and the sum of the numbers in the cells (i, j) and (j, i) equals the number of times the scheme in the i-th row and j-th column had identical log-likelihood scores.

From the MPE contingency table given in Table 1, we observe that SSMP is superior to Max, ML, and Seq approximations. The Seq approximation is slightly better than the Max approximation, and ML is the worst-performing scheme. For the harder MMAP task, we see a similar ordering among the competing schemes (see Table 1) with SSMP dominating other schemes. In particular, SSMP outperforms the Max and ML approximations in almost two-thirds of the cases and the Seq method in more than half of the cases.

We also investigate the effectiveness of SSMP and other baseline approaches when employed as initialization strategies for Hill Climbing Search. These findings are illustrated in the last two contingency tables given in Table 1. Notably, SSMP outperforms all other baseline approaches in nearly two-thirds of the experiments for both MPE and MMAP tasks. These results demonstrate that SSMP can serve as an effective initialization technique for anytime local search-based algorithms.



(a) MPE (b) MMAP

Figure 2: Heat map showing the % difference in log-likelihood scores between SSMP and Max approximation. Blue represents Max's superiority (negative values) and red indicates SSMP better performance (positive values).

In Figure 2, via a heat-map representation, we show a more detailed performance comparison between SSMP and the Max approximation, which is a widely used baseline for MPE and MMAP inference in PCs. In the heat-map representation, the y-axis represents the datasets (ordered by the number of variables), while the x-axis shows the query ratio. The values in each cell represent the percentage difference between the mean log-likelihood scores of SSMP and

	CIFAR				MNIST				EMNIST			
	Max	SSMP	ML	Seq	Max	SSMP	ML	Seq	Max	SSMP	ML	Seq
Max	0	0	0	2	0	1	0	1	0	1	0	5
SSMP	9	0	9	9	9	0	9	9	7	0	7	7
ML	0	0	0	2	0	1	0	1	0	1	0	5
Seq	7	0	7	0	9	1	9	0	3	1	3	0

Table 2: Contingency tables comparing competing methods for MPE on CIFAR, MNIST and EMNIST datasets. Highlighted values represent results for SSMP.

the Max approximation. Formally, let ll_{ssmp} and ll_{max} denote the mean LL scores of SSMP and Max approximation respectively, then the percentage difference is given by

$$\%\text{Diff.} = \frac{ll_{ssmp} - ll_{max}}{|ll_{max}|} \times 100 \tag{7}$$

From the heatmap for MPE given in Figure 2(a), we observe that SSMP is competitive with the Max approximation when the size of the query set is small. However, as the number of query variables increases, signaling a more challenging problem, SSMP consistently outperforms or has similar performance to the Max method across all datasets, except for accidents, pumsb-star, and book.

The heatmaps for MMAP are illustrated in Figure 2(b). We see a similar trend as the one for MPE; SSMP remains competitive with the Max approximation, particularly when the number of query variables is small. While SSMP outperforms (with some exceptions) the Max approximation when the number of query variables is large.

Finally, we present inference times in the supplement. On average SSMP requires in the order of 7-10 micro-seconds for MMAP inference on an A40 GPU. The Max approximation takes 7 milli-seconds (namely, SSMP is almost 1000 times faster). In comparison, as expected, the Seq and ML approximations are quite slow, requiring roughly 400 to 600 milliseconds to answer MPE and MMAP queries. In the case of our proposed method (SSMP), during the inference process, the size of the SPN holds no relevance; its time complexity is linear in the size of the neural network. On the contrary, for the alternative methods, the inference time is intricately dependent on the size of the SPN.

Results on the CIFAR-10 Dataset

We binarized the CIFAR-10 dataset using a variational autoencoder having 512 bits. We then learned a PC for each of the 10 classes; namely, we learned a PC conditioned on the class variable. As mentioned earlier, we randomly set 70% of the variables as query variables. The contingency table for CIFAR-10 is shown in Table 2. We observe that SSMP dominates all competing methods while the Seq approximation is the second-best performing scheme (although note that Seq is computationally expensive).

Results on the MNIST and EMNIST Datasets

Finally, we evaluated SSMP on the image completion task using the Binarized MNIST (Salakhutdinov and Murray 2008) and the EMNIST datasets (Cohen et al. 2017). As mentioned earlier, we used the top 30% of the image as evidence and estimated the bottom 70% by solving the MPE task over PCs using various competing methods. The contingency tables for the MNIST and EMNIST datasets are shown in Table 2. We observe that on the MNIST dataset, SSMP is better than all competing schemes on 9 out of the 10 PCs, while it is inferior to all on one of them. On the EMNIST dataset, SSMP is better than all competing schemes on 7 out of the 10 PCs and inferior to all on one of the PCs. Detailed results on the image datasets, including qualitative comparisons, are provided in the supplement.

In summary, we find that, on average, our proposed method (SSMP) is better than other baseline MPE/MMAP approximations in terms of log-likelihood score. Moreover, it is substantially better than the baseline methods when the number of query variables is large. Also, once learned from data, it is also significantly faster than competing schemes.

Conclusion and Future Work

In this paper, we introduced a novel self-supervised learning algorithm for solving MMAP queries in PCs. Our contributions comprise a neural network approximator and a self-supervised loss function which leverages the tractability of PCs for achieving scalability. Notably, our method employs minimal hyperparameters, requiring only one in the discrete case. We conducted a comprehensive empirical evaluation across various benchmarks; specifically, we experimented with 22 binary datasets used in tractable probabilistic models community and three classic image datasets, MNIST, EMNIST, and CIFAR-10. We compared our proposed neural approximator to polytime baseline techniques and observed that it is superior to the baseline methods in terms of log-likelihood scores and is significantly better in terms of computational efficiency. Additionally, we evaluated how our approach performs when used as an initialization scheme in stochastic hill climbing (local) search and found that it improves the quality of solutions output by anytime local search schemes. Our empirical results clearly demonstrated the efficacy of our approach in both accuracy and speed.

Future work includes compiling PCs to neural networks for answering more complex queries that involve constrained optimization; developing sophisticated self-supervised loss functions; learning better NN architecture for the given PC; generalizing our approach to arbitrarily chosen query and evidence subsets; etc.

Acknowledgements

This work was supported in part by the DARPA Perceptually-Enabled Task Guidance (PTG) Program under contract number HR00112220005, by the DARPA Assured

Neuro Symbolic Learning and Reasoning (ANSR) Program under contract number HR001122S0039, by the National Science Foundation grant IIS-1652835 and by the AFOSR award FA9550-23-1-0239.

References

Bekker, J.; Davis, J.; Choi, A.; Darwiche, A.; and Van den Broeck, G. 2015. Tractable Learning for Complex Probability Queries. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Bioucas-Dias, J.; and Figueiredo, M. 2016. Bayesian Image Segmentation Using Hidden Fields: Supervised, Unsupervised, and Semi-Supervised Formulations. In *2016 24th European Signal Processing Conference (EUSIPCO)*, 523–527.

Choi, Y.; Friedman, T.; and Van den Broeck, G. 2022. Solving marginal map exactly by probabilistic circuit transformations. In *International Conference on Artificial Intelligence and Statistics*, 10196–10208. PMLR.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models. Technical report, University of California, Los Angeles.

Cohen, G.; Afshar, S.; Tapson, J.; and van Schaik, A. 2017. EMNIST: An Extension of MNIST to Handwritten Letters. arxiv:1702.05373.

Conaty, D.; de Campos, C. P.; and Mauá, D. D. 2017. Approximation Complexity of Maximum A Posteriori Inference in Sum-Product Networks. In Elidan, G.; Kersting, K.; and Ihler, A., eds., *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press.

Cui, Z.; Wang, H.; Gao, T.; Talamadupula, K.; and Ji, Q. 2022. Variational Message Passing Neural Network for Maximum-A-Posteriori (MAP) Inference. In Cussens, J.; and Zhang, K., eds., *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, the Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, 464–474. PMLR.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, 50(3): 280–305.

de Campos, C. P. 2011. New Complexity Results for MAP in Bayesian Networks. *IJCAI International Joint Conference on Artificial Intelligence*, 2100–2106.

Dechter, R.; and Mateescu, R. 2007. AND/OR search spaces for graphical models. *Artificial intelligence*, 171(2-3): 73–106.

Donti, P. L.; Rolnick, D.; and Kolter, J. Z. 2020. DC3: A Learning Method for Optimization with Hard Constraints. In *International Conference on Learning Representations*.

Fioretto, F.; Mak, T. W. K.; and Hentenryck, P. V. 2020. Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01): 630–637.

Gogate, V.; and Dechter, R. 2012. Importance sampling-based estimation over AND/OR search spaces for graphical models. *Artificial Intelligence*, 184-185: 38–77.

Haaren, J. V.; and Davis, J. 2012. Markov Network Structure Learning: A Randomized Feature Generation Approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1): 1148–1154.

Kingma, D. P.; and Ba, J. 2017. Adam: A Method for Stochastic Optimization. arxiv:1412.6980.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Kiselev, I.; and Poupart, P. 2014. POMDP Planning by Marginal-MAP Probabilistic Inference in Generative Models. In *Proceedings of the 2014 AAMAS Workshop on Adaptive Learning Agents*.

Krizhevsky, A.; Nair, V.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Technical Report, University of Toronto.

Larochelle, H.; and Murray, I. 2011. The Neural Autoregressive Distribution Estimator. In Gordon, G.; Dunson, D.; and Dudík, M., eds., *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, 29–37. Fort Lauderdale, FL, USA: PMLR.

Lee, J.; Marinescu, R.; and Dechter, R. 2014. Applying Marginal MAP Search to Probabilistic Conformant Planning: Initial Results. In *Statistical Relational Artificial Intelligence, Papers from the 2014 AAAI Workshop, Québec City, Québec, Canada, July 27, 2014*, volume WS-14-13 of *AAAI Technical Report*. AAAI.

Li, K.; and Malik, J. 2016. Learning to Optimize. arXiv:1606.01885.

Loconte, L.; and Gala, G. 2022. DeeProb-kit: a Python Library for Deep Probabilistic Modelling.

Lowd, D.; and Davis, J. 2010. Learning Markov Network Structure with Decision Trees. In *2010 IEEE International Conference on Data Mining*, 334–343. IEEE. ISBN 978-1-4244-9131-5.

Mauá, D. D.; Reis, H. R.; Katague, G. P.; and Antonucci, A. 2020. Two reformulation approaches to maximum-a-posteriori inference in sum-product networks. In *International Conference on Probabilistic Graphical Models*, 293–304. PMLR.

Mei, J.; Jiang, Y.; and Tu, K. 2018. Maximum a posteriori inference in sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Park, J. D.; and Darwiche, A. 2004. Complexity Results and Approximation Strategies for MAP Explanations. *J. Artif. Int. Res.*, 21(1): 101–133.

Park, S.; and Hentenryck, P. V. 2023. Self-Supervised Primal-Dual Learning for Constrained Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4): 4052–4060.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Peharz, R. 2015. *Foundations of sum-product networks for probabilistic modeling*. Ph.D. 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.

Ping, W.; Liu, Q.; and Ihler, A. T. 2015. Decomposition Bounds for Marginal MAP. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Poon, H.; and Domingos, P. 2011. Sum-Product Networks: A New Deep Architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 337–346. AUAI Press.

Rahman, T.; and Gogate, V. 2016a. Learning Ensembles of Cutset Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Rahman, T.; and Gogate, V. 2016b. Merging Strategies for Sum-Product Networks: From Trees to Graphs. In *Proceedings of the Thirty-Second Conference Conference on Uncertainty in Artificial Intelligence*, 617–626.

Rahman, T.; Jin, S.; and Gogate, V. 2019. Look ma, no latent variables: Accurate cutset networks via compilation. In *International Conference on Machine Learning*, 5311–5320. PMLR.

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*, 630–645. Springer.

Salakhutdinov, R.; and Murray, I. 2008. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, 872–879. ACM.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 13189–13201. Curran Associates, Inc.

Yoon, K.; Liao, R.; Xiong, Y.; Zhang, L.; Fetaya, E.; Urtasun, R.; Zemel, R.; and Pitkow, X. 2019. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 868–875. IEEE.

Zamzam, A. S.; and Baker, K. 2020. Learning Optimal Solutions for Extremely Fast AC Optimal Power Flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 1–6.