Learning to Solve the Constrained Most Probable Explanation Task in Probabilistic Graphical Models

Shivvrat Arya[*]
The University of Texas at Dallas

Tahrima Rahman[*]
The University of Texas at Dallas

Vibhav Gogate
The University of Texas at Dallas

Abstract

We propose a *self-supervised learning* approach for solving the following constrained optimization task in log-linear models or Markov networks. Let f and g be two log-linear models defined over the sets $\mathbf{X}$ and $\mathbf{Y}$ of random variables respectively. Given an assignment $\mathbf{x}$ to all variables in $\mathbf{X}$ (evidence) and a real number q, the constrained most-probable explanation (CMPE) task seeks to find an assignment $\mathbf{y}$ to all variables in $\mathbf{Y}$ such that $f(\mathbf{x}, \mathbf{y})$ is maximized and $g(\mathbf{x}, \mathbf{y}) \leq q$. In our proposed self-supervised approach, given assignments $\mathbf{x}$ to $\mathbf{X}$ (data), we train a deep neural network that learns to output near-optimal solutions to the CMPE problem without requiring access to any pre-computed solutions. The key idea in our approach is to use first principles and approximate inference methods for CMPE to derive novel loss functions that seek to push infeasible solutions towards feasible ones and feasible solutions towards optimal ones. We analyze the properties of our proposed method and experimentally demonstrate its efficacy on several benchmark problems.

1 INTRODUCTION

Probabilistic graphical models (PGMs) such as Bayesian and Markov networks (Koller and Friedman, 2009; Darwiche, 2009) compactly represent joint probability distributions over random variables by factorizing the distribution according to a graph structure that encodes conditional independence among the variables. Once learned from data, these models can be used to answer various queries, such as computing the marginal probability distribution over a subset of variables (MAR) and finding the most likely assignment to all unobserved variables, which is referred to as the most probable explanation (MPE) task.

Recently, Rouhani et al. (2020) proposed an extension to the MPE task in PGMs by introducing constraints. More specifically, given two PGMs f and g defined over the set of random variables $\mathbf{X}$ and a real number q, the constrained most probable explanation (CMPE) task seeks to find the most likely state $\mathbf{X} = \mathbf{x}$ w.r.t. f such that the constraint $g(\mathbf{x}) \leq q$ is satisfied. Even though both MPE and CMPE are NP-hard in general, CMPE is considerably more difficult to solve in practice than MPE. Notably, CMPE is NP-hard even on PGMs having no edges, such as zero treewidth or independent PGMs, while MPE can be solved in linear time. Rouhani et al. (2020) and later Rahman et al. (2021) showed that several probabilistic inference queries are special cases of CMPE. This includes queries such as finding the decision preserving most probable explanation (Choi et al., 2012), finding the nearest assignment (Rouhani et al., 2018) and robust estimation (Darwiche and Hirth, 2023, 2020).

Our interest in the CMPE task is motivated by its extensive applicability to various *neuro-symbolic inference* tasks. Many of these tasks can be viewed as specific instances of CMPE. Specifically, when $f(\mathbf{x})$ represents a function encoded by a neural network and $g(\mathbf{x}) \leq q$ signifies particular symbolic or weighted constraints that the neural network must adhere to, the neuro-symbolic inference task involves determining the most likely prediction with respect to f while ensuring that the constraint $g(\mathbf{x}) \leq q$ is satisfied. Another notable application of CMPE involves transferring abstract knowledge and inferences from simulations to real-world contexts. For example, in robotics, numerous simulations can be employed to instruct the robot on various aspects, such as object interactions, robot-world interactions, and underlying physical principles, encapsulating this abstract knowledge within the constraint $g(\mathbf{x}) \leq q$. Subsequently, with a neural network f trained on a limited amount of real-world data, characterized by richer feature sets and objectives, g can be used to reinforce the predictions made by f, ensuring that the robot identifies the most likely prediction with respect to f while satisfying the constraint $g(\mathbf{x}) \leq q$. This strategy enhances the reliability of

[*]These authors contributed equally to this work.

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

the robot's predictions and underscores the practical significance of CMPE.

In this paper, we explore novel machine learning (ML) approaches for solving the CMPE task, drawing inspiration from recent developments in *learning to optimize* (Donti et al., 2021; Fioretto et al., 2020; Park and Van Hentenryck, 2022; Zamzam and Baker, 2019). The main idea in these works is to train a deep neural network that takes the parameters, observations, etc. of a constrained optimization problem *as input* and outputs a near-optimal solution to the optimization problem.

In practice, a popular approach for solving optimization problems is to use search-based solvers such as Gurobi and SCIP. However, a drawback of these off-the-shelf solvers is their inability to efficiently solve large problems, especially those with dense global constraints, such as the CMPE problem. In contrast, neural networks are efficient because once trained, the time complexity of solving an optimization problem using them scales linearly with the network's size. This attractive property has also driven their application in solving probabilistic inference tasks such as MAR and MPE inference (Gilmer et al., 2017; Kuck et al., 2020; Zhang et al., 2020; Satorras and Welling, 2021). However, all of these works require access to exact inference techniques in order to train the neural network. As a result, they are feasible only for small graphical models on which exact inference is tractable. Recently, Cui et al. (2022) proposed to solve the MPE task by training a variational distribution that is parameterized by a neural network in a self-supervised manner (without requiring access to exact inference methods). To the best of our knowledge, there is no prior work on using neural networks for solving the CMPE problem.

In this paper, we propose a new self-supervised approach for training neural networks which takes observations or evidence as input and outputs a near optimal solution to the CMPE task. Existing self-supervised approaches (Fioretto et al., 2020; Park and Van Hentenryck, 2022) in the *learning to optimize* literature either relax the constrained objective function using Lagrangian relaxation and then use the Lagrangian dual as a loss function or use the Augmented Lagrangian method. We show that these methods can be easily adapted to solve the CMPE task. Unfortunately, an issue with them is that an optimal solution to the Lagrangian dual is not guaranteed to be an optimal solution to the CMPE task (because of the non-convexity of CMPE, there is a duality gap). To address this issue, we propose a new loss function based on first principles and show that an optimal solution to the loss function is also an optimal solution to the CMPE task. Moreover, our new loss function has several desirable properties, which include: (a) during training, when the constraint is violated, it focuses on decreasing the strength of the violation, and (b) when constraints are not violated, it focuses on increasing the value of the objective function associated with the CMPE task.

We conducted a comprehensive empirical evaluation, comparing several supervised and self-supervised approaches to our proposed method. To the best of our knowledge, these are the first empirical results on using machine learning, either supervised or self-supervised, to solve the CMPE task in PGMs. On a number of benchmark models, our experiments show that neural networks trained using our proposed loss function are more efficient and accurate compared to models trained to minimize competing supervised and self-supervised loss functions from the literature.

2 Notation and Background

We denote random variables by upper-case letters (e.g., X, Y, Z, etc.), their corresponding assignments by lower-case letters (e.g., x, y, z, etc.), sets of random variables by bold upper-case letters (e.g., $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, etc.) and assignments to them by bold lower-case letters (e.g., $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, etc.). $\mathbf{z}_{\mathbf{X}}$ denotes the projection of the complete assignment $\mathbf{z}$ on to the subset $\mathbf{X}$ of $\mathbf{Z}$. For simplicity of exposition, we assume that discrete and continuous random variables take values from the set $\{0,1\}$ and $[0,1]$ respectively.

We use the multilinear polynomial representation (Sherali and Adams, 2009; Sherali and Tuncbilek, 1992; Horst and Tuy, 1996) to concisely describe our proposed method as well as for specifying discrete, continuous, and mixed constrained optimization problems. Let $\mathbf{Z} = \{Z_1, \ldots, Z_n\}$ be a set of random variables. Let $[n] = \{1, \ldots, n\}$ and $i \in [n]$ be an index over the variables of $\mathbf{Z}$. Let $2^{[n]}$ denote the set of subsets of indices of $[n]$; thus each element of $2^{[n]}$ denotes a (unique) subset of $\mathbf{Z}$. Let $\mathcal{I} \subseteq 2^{[n]}$ and let $w_I \in \mathbb{R}$ where $I \in \mathcal{I}$ be a real number (weight) associated with each element I of $\mathcal{I}$. Then, a multilinear polynomial is given by

$$f(\mathbf{z}) = f(z_1, \ldots, z_n) = \sum_{I \in \mathcal{I}} w_I \prod_{i \in I} z_i \qquad (1)$$

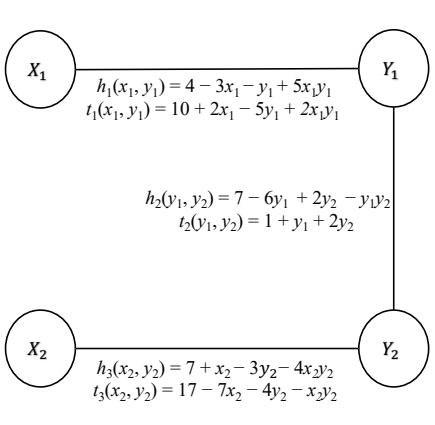
where $\mathbf{z} = (z_1, \ldots, z_n)$ is an assignment to all variables in $\mathbf{Z}$. We will call $f(\mathbf{z})$ the weight of $\mathbf{z}$.

It is known that weighting functions, namely the sum of log of conditional probability tables and log-potentials associated with Bayesian and Markov networks respectively can be expressed as multilinear polynomials (see for example (Koller and Friedman, 2009)).

Example 1. *Figure 1 shows a multilinear representation for a Markov network. The weight of the assignment* $(X_1 = 0, X_2 = 1, Y_1 = 0, Y_2 = 1)$ *is 14 and 18 w.r.t.* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *respectively.*

2.1 Constrained Most Probable Explanation

We are interested in solving the following constrained most probable explanation (CMPE) task. Let $\mathbf{X}$ and $\mathbf{Y}$ be two subsets of $\mathbf{Z}$ such that $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ and $\mathbf{X} \cap \mathbf{Y} = \emptyset$. We



x_1	y_1	h_1	t_1
0	0	4	10
0	1	3	5
1	0	1	12
1	1	5	9

y_1	y_2	h_2	t_2
0	0	7	1
0	1	9	3
1	0	1	2
1	1	2	4

x_2	y_2	h_3	t_3
0	0	7	17
0	1	4	13
1	0	8	10
1	1	1	5

Figure 1: Two Markov networks $\mathcal{M}_1$ and $\mathcal{M}_2$ having the same chain-like structure and defined over the same set $\{X_1, X_2, Y_1, Y_2\}$ of variables . $\mathcal{M}_1$ is defined by the set of log-potentials $\{h_1, h_2, h_3\}$ and $\mathcal{M}_2$ is defined by the set of log-potentials $\{t_1, t_2, t_3\}$. Each log-potential can be expressed as a local multilinear polynomial function. The global multilinear function representing $\mathcal{M}_1$ and $\mathcal{M}_2$ are $h(x_1, x_2, y_1, y_2) = 18 - 3x_1 + x_2 - 7y_1 - y_2 + 5x_1y_1 - 4x_2y_2 - y_1y_2$ and $t(x_1, x_2, y_1, y_2) = 28 + 2x_1 - 7x_2 - 4y_1 - 2y_2 + 2x_1y_1 - x_2y_2$ respectively which are obtained by adding the local functions associated with the respective models and then simplifying, i.e., $h(x_1, x_2, y_1, y_2) = h_1(x_1, y_1) + h_2(y_1, y_2) + h_3(x_2, y_2)$. $t(x_1, x_2, y_1, y_2)$ is obtained similarly.

will refer to $\mathbf{Y}$ as *decision variables* and $\mathbf{X}$ as *evidence variables*. Given assignments $\mathbf{x}$ and $\mathbf{y}$, let $(\mathbf{x}, \mathbf{y})$ denote their composition. Let h and t denote two multilinear polynomials over $\mathbf{Z}$ obtained from two Markov networks $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively that represent two (possibly different) joint probability distributions over $\mathbf{Z}$. Then given a real number q and an assignment $\mathbf{x}$ to all variables in $\mathbf{X}$, the CMPE task is to find an assignment $\mathbf{y}^*$ to all the variables in $\mathbf{Y}$ such that $h(\mathbf{x}, \mathbf{y}^*)$ is maximized (namely the probability of the assignment w.r.t. $\mathcal{M}_1$ is maximized) and $t(\mathbf{x}, \mathbf{y}^*) \leq q$ (namely the probability of the assignment w.r.t. $\mathcal{M}_2$ is bounded by a constant). Formally,

$$\underset{\mathbf{y}}{\text{maximize}} \ h(\mathbf{x}, \mathbf{y}) \ \ s.t. \ \ t(\mathbf{x}, \mathbf{y}) \leq q \tag{2}$$

For brevity, we will abuse notation and use $h_{\mathbf{x}}(\mathbf{y})$ and $t_{\mathbf{x}}(\mathbf{y})$ to denote $h(\mathbf{x}, \mathbf{y})$ and $t(\mathbf{x}, \mathbf{y})$ respectively. The most probable explanation (MPE) task in probabilistic graphical models (Koller and Friedman, 2009) is a special case of CMPE; MPE is just CMPE without the constraint $t_{\mathbf{x}}(\mathbf{y}) \leq q$. The goal in MPE is to find an assignment $\mathbf{y}^*$ to $\mathbf{Y}$ such that the weight $h_{\mathbf{x}}(\mathbf{y}^*)$ of the assignment is maximized given evidence $\mathbf{x}$. Similar to MPE, CMPE is NP-hard in general, with the caveat that CMPE is much harder than MPE. Specifically, CMPE is NP-hard even on independent graphical models (having zero treewidth), where MPE can be solved in linear time by independently maximizing each univariate function (Rouhani et al., 2020).

Example 2. *Given $X_1 = 1$, $X_2 = 1$ and $q = 20$, the CMPE solution of the example problem in figure 1 is $(y_1^*, y_2^*) = (0, 1)$ with a value $h(1, 1, 0, 1) = 11$, whereas the MPE solution is $(y_1^*, y_2^*) = (0, 0)$ with value $h(1, 1, 0, 0) = 16$.*

Since we are interested in machine learning approaches to solve the CMPE task and such approaches employ loss functions, it is convenient to express CMPE as a minimization task with a "≤ 0" constraint. This can be accomplished by negating h and subtracting q from t. Formally, let $f_{\mathbf{x}}(\mathbf{y}) = -h_{\mathbf{x}}(\mathbf{y})$ and $g_{\mathbf{x}}(\mathbf{y}) = t_{\mathbf{x}}(\mathbf{y}) - q$. Then Eq. (2) is equivalent to the following minimization problem:

$$\underset{\mathbf{y}}{\text{minimize}} \ f_{\mathbf{x}}(\mathbf{y}) \ \ s.t. \ \ g_{\mathbf{x}}(\mathbf{y}) \leq 0 \tag{3}$$

Let $\mathbf{y}^*$ be the optimal solution to the problem given in Eq. (3) and let $p_{\mathbf{x}}^* = f_{\mathbf{x}}(\mathbf{y}^*)$. Also, without loss of generality, we assume that $f_{\mathbf{x}}$ is strictly positive, i.e., $\forall \mathbf{y}, \ f_{\mathbf{x}}(\mathbf{y}) > 0$.

If all variables in $\mathbf{Y}$ are binary (or discrete in general), Eq. (3) can be formulated as an (equivalent) integer linear programming (ILP) problem by introducing auxiliary integer variables for each multilinear term (e.g., $y_{1,2} = y_1y_2$, $y_{2,3} = y_2y_3$, etc.) and adding appropriate constraints to model the equivalence between the auxiliary variables and multilinear terms (see for example (Koller and Friedman, 2009), Chapter 13). Therefore, in practice, (3) can be solved optimally using mixed integer linear programming (MILP) solvers such as Gurobi (Gurobi Optimization, 2021) and SCIP (Achterberg et al., 2008; Achterberg, 2009).

Unfortunately, due to a presence of a dense global constraint, namely $g_{\mathbf{x}}(\mathbf{y}) \leq 0$ in Eq. (3), the MILP solvers often perform poorly. Instead, in practice, application designers often use efficient, specialized algorithms that exploit problem structure for lower bounding $p_{\mathbf{x}}^*$, and then using these lower bounds in an *anytime* branch-and-bound algorithm to obtain an upper bound on $p_{\mathbf{x}}^*$.

2.2 Specialized Lower Bounding Algorithms

Recently, Rahman et al. (2021) proposed two new approaches for computing upper bounds on the optimal value of the maximization problem given in Eq. (2). These methods can be easily adapted to obtain a lower bound on $p_{\mathbf{x}}^*$; because an upper bound on the maximization problem is a lower bound on the corresponding minimization problem. We present the adaptations of Rahman et al.'s approach next.

The first approach is based on the Lagrangian relaxation method that introduces a *Lagrange multiplier* $\mu \geq 0$ to transform the constrained minimization problem to the following unconstrained problem: $\text{minimize}_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \mu g_{\mathbf{x}}(\mathbf{y})$. Let d_{μ}^* denote the optimal value of the unconstrained problem. Then, it is easy to show that $d_{\mu}^* \leq p_{\mathbf{x}}^*$. The largest upper

bound is obtained by finding a value of μ that maximizes d^*_μ. More formally,

$$\max_{\mu \geq 0} d^*_\mu = \max_{\mu \geq 0} \min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \mu g_{\mathbf{x}}(\mathbf{y}) \leq p^*_{\mathbf{x}} \quad (4)$$

Rahman et al. (2021) proposed to solve the inner minimization problem using exact techniques from the graphical models literature such as variable/bucket elimination (Dechter, 1999), branch and bound search and best-first search (Marinescu and Dechter, 2012, 2009; Wu et al., 2020). When exact inference is not feasible, Rahman et al. (2021) proposed to solve the inner problem using approximate inference techniques such as mini-bucket elimination, dual-decomposition and join-graph based bounding algorithms (Choi and Darwiche, 2011; Dechter and Rish, 2003; Wainwright et al., 2005; Globerson and Jaakkola, 2007; Komodakis et al., 2007; Ihler et al., 2012). The outer maximization problem is solved using sub-gradient ascent.

The second approach by Rahman et al. (2021) uses the Lagrangian decomposition method to transform the problem into a multi-choice knapsack problem (MCKP) and then utilizes off-the-shelf MCKP solvers. In our experiments, we use the Lagrange relaxation approach given in Eq. (4).

If the set $\mathbf{Y}$ contains continuous variables, then it is not possible to reduce it to an equivalent MILP/LP (Horst and Tuy, 1996; Sherali and Tuncbilek, 1992). However, by leveraging linearization methods (Sherali and Tuncbilek, 1992; Sherali and Adams, 2009) and solving the resulting problem using linear programming (LP) solvers, we can still obtain good lower bounds on $p^*_{\mathbf{x}}$.

3 Solving CMPE using Methods from the Learning to Optimize Literature

In this section, we show how techniques developed in the learning to optimize literature (Donti et al., 2021; Fioretto et al., 2020; Park and Van Hentenryck, 2022; Zamzam and Baker, 2019) which seeks to develop machine learning approaches for solving constrained optimization problems can be leveraged to solve the CMPE task. The main idea is to train a deep neural network $\mathcal{F}_\Theta : \mathbf{X} \rightarrow \mathbf{Y}$ parameterized by the set $\Theta \in \mathbb{R}^M$ such that at test time given evidence $\mathbf{x}$, the network is able to predict an (near) optimal solution $\hat{\mathbf{y}}$ to the CMPE problem. Note that as far as we are aware, no prior work exists on solving CMPE using deep neural networks.

3.1 Supervised Methods

In order to train the parameters of $\mathcal{F}_\Theta$ in a supervised manner, we need to acquire labeled data in the form $\mathcal{D} = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}_{i=1}^N$ where each label $\mathbf{y}_i$ is an optimal solution to the problem given in Eq. (3) given $\mathbf{x}_i$. In practice, we can generate the assignments $\{\mathbf{x}_i\}_{i=1}^N$ by sampling them from the graphical model corresponding to f and the labels $\{\mathbf{y}_i\}_{i=1}^N$ by solving the minimization problem given in Eq. (3) using off-the-shelf solvers such as Gurobi and SCIP.

Let $\hat{\mathbf{y}}_i = \mathcal{F}_\Theta(\mathbf{x}_i)$ denote the labels predicted by the neural network for $\mathbf{x}_i$. Following Zamzam and Baker (2019), we propose to train $\mathcal{F}_\Theta$ using the following two loss functions

$$\text{Mean-Squared Error (MSE)} : \frac{1}{N} \sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \quad (5)$$

$$\text{Mean-Absolute-Error (MAE)}: \frac{1}{N} \sum_i |\mathbf{y}_i - \hat{\mathbf{y}}_i| \quad (6)$$

Experimentally (see the supplementary material), we found that neural networks trained using the MAE and MSE loss functions often output infeasible assignments. To address this issue, following prior work (Nellikkath and Chatzivasileiadis, 2021), we propose to add $\lambda_{\mathbf{x}} \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}$ to the loss function where $\lambda_{\mathbf{x}}$ is a penalty coefficient.

In prior work (Fioretto et al., 2020), it was observed that the quality of the solutions greatly depends on the value chosen for $\lambda_{\mathbf{x}}$. Moreover, it is not straightforward to choose it optimally because it varies for each $\mathbf{x}$. To circumvent this issue, we propose to update $\lambda_{\mathbf{x}}$ via a *Lagrangian dual* method (Nocedal and Wright, 2006). More specifically, we propose to use the following subgradient method to optimize the value of $\lambda_{\mathbf{x}}$. While training a neural network, let $\lambda^k_{\mathbf{x}_i}$ and $\hat{\mathbf{y}}^k_i$ denote the values of the penalty co-efficient and the predicted assignment respectively at the k-th epoch and for the i-th example in $\mathcal{D}$ (if the i-th example is part of the current mini-batch), then, we update $\lambda^{k+1}_{\mathbf{x}_i}$ using

$$\lambda^{k+1}_{\mathbf{x}_i} = \lambda^k_{\mathbf{x}_i} + \rho \max\{0, g_{\mathbf{x}_i}(\hat{\mathbf{y}}^k_i)\} \quad (7)$$

where ρ is the Lagrangian step size. In our experiments, we evaluated both the naive and the penalty based supervised loss approaches (for CMPE) and found that the penalty method with MSE loss yields the best results. Therefore, in our experiments, we use it as a strong supervised baseline.

3.2 Self-Supervised Methods

Supervised methods require pre-computed solutions for numerous NP-hard/multilinear problem instances, which are computationally expensive to derive. Therefore, we propose to train the neural network in a self-supervised manner that does not depend on the pre-computed results. Utilizing findings from Kotary et al. (2021) and Park and Van Hentenryck (2022), we introduce two self-supervised approaches: one is grounded in the *penalty method*, and the other builds upon the *augmented Lagrangian method*.

Penalty Method (Donti et al., 2021; Kotary et al., 2021; Fioretto et al., 2020). In the penalty method, we solve the constrained minimization problem by iteratively transforming it into a sequence of unconstrained problems. Each unconstrained problem at iteration k is constructed by adding

a term, which consists of a penalty parameter $\lambda_{\mathbf{x}}^k$ multiplied by a function $\max\{0, g_{\mathbf{x}}(\mathbf{y})\}^2$ that quantifies the constraint violations, to the objective function. Formally, the optimization problem at the k-th step is given by:

$$\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\{0, g_{\mathbf{x}}(\mathbf{y})\}^2 \tag{8}$$

Here, $\lambda_{\mathbf{x}}^k$ is progressively increased either until the constraint is satisfied or a predefined maximum λ_{max} is reached. $\lambda_{\mathbf{x}}^k$ can be updated after a few epochs using simple strategies such as multiplication by a fixed factor (e.g., 2, 10, etc.).

The penalty method can be adapted to learn a neural network in a self-supervised manner as follows. At each epoch k, we sample an assignment $\mathbf{x}$ (or multiple samples for a mini-batch) from the graphical model corresponding to f, predict $\hat{\mathbf{y}}$ using the neural network and then use the following loss function to update its parameters:

$$\mathcal{L}_{\mathbf{x}}^{pen}(\hat{\mathbf{y}}) = f_{\mathbf{x}}(\hat{\mathbf{y}}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2 \tag{9}$$

Determining the optimal $\lambda_{\mathbf{x}}^k$ is crucial. In prior work, Kotary et al. (2021) and Fioretto et al. (2020) proposed to update it via a subgradient method, similar to the update rule given by Eq. (7). More formally, we can update $\lambda_{\mathbf{x}}^k$ using:

$$\lambda_{\mathbf{x}}^{k+1} = \lambda_{\mathbf{x}}^k + \rho \max\left\{0, g_{\mathbf{x}}(\hat{\mathbf{y}}^k)\right\} \tag{10}$$

where ρ is the Lagrangian step size.

Augmented Lagrangian Method (ALM). In this method, we augment the objective used in the penalty method with a Lagrangian term. More formally, the optimization problem at the k-th step is given by (compare with Eq. (8)):

$$\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\{0, g_{\mathbf{x}}(\mathbf{y})\}^2 + \mu_{\mathbf{x}}^k g_{\mathbf{x}}(\mathbf{y}) \tag{11}$$

Here, $\lambda_{\mathbf{x}}^k$ may be progressively increased similar to the penalty method while $\mu_{\mathbf{x}}^k$ is updated using

$$\mu_{\mathbf{x}}^{k+1} = \max\left\{0, \mu_{\mathbf{x}}^k + \lambda_{\mathbf{x}}^k g_{\mathbf{x}}(\mathbf{y}^k)\right\} \tag{12}$$

Recently, Park and Van Hentenryck (2022) proposed a self-supervised primal-dual learning method that leverages two distinct networks to emulate the functionality of ALM: the first (primal) network takes as input $\mathbf{x}$ and outputs $\mathbf{y}$ while the second network focuses on learning the dual aspects; specifically it takes $\mathbf{x}$ as input and outputs $\mu_{\mathbf{x}}^k$. The training process uses a sequential approach, where one network is trained while the other remains frozen to furnish the requisite values for the loss computation.

The primal network uses the following loss function:

$$\mathcal{L}_{\mathbf{x}}^{A,p}(\hat{\mathbf{y}}|\mu, \lambda) = f_{\mathbf{x}}(\hat{\mathbf{y}}) + \frac{\lambda}{2} \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2 + \mu g_{\mathbf{x}}(\mathbf{y})$$

While the dual network uses the following loss function

$$\mathcal{L}_{\mathbf{x}}^{A,d}(\hat{\mu}|\mathbf{y}, \lambda, \mu^k) = ||\hat{\mu} - \max\left\{0, \mu^k + \lambda g_{\mathbf{x}}(\mathbf{y})\right\}||$$

where $\hat{\mu}$ is the predicted value of the Lagrangian multiplier.

3.2.1 Drawbacks of the Penalty and ALM Methods

A limitation of the penalty-based self-supervised method is that it does not guarantee a global minimum unless specific conditions are met. In particular, the optimal solution w.r.t. the loss function (see Eq. (9)) may be far away from the optimal solution $\mathbf{y}^*$ of the problem given in Eq. (3), unless the penalty co-efficient $\lambda_{\mathbf{x}}^k \to \infty$. Moreover, when $\lambda_{\mathbf{x}}^k$ is large for all $\mathbf{x}$, the gradients will be uninformative. In the case of ALM method (cf. (Nocedal and Wright, 2006)), for global minimization, we require that either $\lambda_{\mathbf{x}}^k \to \infty$ or $\forall \mathbf{x}$ with $g_{\mathbf{x}}(\mathbf{y}) > 0$, $\mu_{\mathbf{x}}^k$ should be such that $\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \mu_{\mathbf{x}}^k g_{\mathbf{x}}(\mathbf{y}) > p_{\mathbf{x}}^*$. Additionally, ALM introduces a dual network, increasing the computational complexity and potentially leading to negative information transfer when the dual network's outputs are inaccurate. These outputs are subsequently utilized in the loss to train the primal network for the following iteration, thereby exerting a negative effect. To address these limitations, next, we introduce a self-supervised method that achieves global minimization without the need for a dual network or infinite penalty coefficients.

4 A NOVEL SELF-SUPERVISED CMPE SOLVER

An appropriately designed loss function should have the following characteristics. For feasible solutions, namely when $g_{\mathbf{x}}(\mathbf{y}) \leq 0$, the loss function should be proportional to $f_{\mathbf{x}}(\mathbf{y})$. While for infeasible assignments, it should equal infinity. This loss function will ensure that once a feasible solution is found, the neural network will only explore the space of feasible solutions. Unfortunately, infinity does not provide any gradient information, and the neural network will get stuck in the infeasible region if the neural network generates an infeasible assignment during training.

An alternative approach is to use g as a loss function when the constraint is not satisfied (i.e., $g_{\mathbf{x}}(\mathbf{y}) > 0$) in order to push the infeasible solutions towards feasible ones (Liu and Cherian, 2023). Unfortunately, this approach will often yield feasible solutions that lie at the boundary $g_{\mathbf{x}}(\mathbf{y}) = 0$. For instance, for a boundary assignment $\mathbf{y}_b$ where $g_{\mathbf{x}}(\mathbf{y}_b) = 0$ but $f_{\mathbf{x}}(\mathbf{y}_b) > 0$ (or decreasing), the sub-gradient will be zero, and the neural network will treat the boundary assignment as an optimal one.

To circumvent this issue, we propose a loss function which has the following two properties: (1) It is proportional to g in the infeasible region with f acting as a control in the boundary region (when g is zero); and (2) It is proportional to f in the feasible region. Formally,

$$\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \begin{cases} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0 \\ \alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}})) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \end{cases} \tag{13}$$

where $\alpha_{\mathbf{x}}$ is a function of the evidence $\mathbf{x}$. Our goal is to find

a bound for $\alpha_{\mathbf{x}}$ such that the following desirable property is satisfied and the bound can be computed in polynomial time for each $\mathbf{x}$ by leveraging bounding methods for CMPE.

Property (Consistent Loss): The loss for all infeasible assignments is higher than the optimal value $p^*_{\mathbf{x}}$.
To satisfy this property, we have to ensure that:

$$\forall \hat{\mathbf{y}} \text{ s.t. } g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0, \quad \alpha_{\mathbf{x}} \left(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}})\right) > p^*_{\mathbf{x}}$$

which implies that the following condition holds.

$$\alpha_{\mathbf{x}} \left(\min_{\hat{\mathbf{y}}} \; f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \;\; s.t. \;\; g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \right) > p^*_{\mathbf{x}}$$

Let $q^*_{\mathbf{x}}$ denote the optimal value of $\min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \;\; s.t. \;\; g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0$. Then, $\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}}$.

Proposition 4.1. *If $\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}})$ is consistent, i.e., $\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}}$ then $\min_{\hat{\mathbf{y}}} \mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = p^*_{\mathbf{x}}$, namely $\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}})$ is an optimal loss function.*

Proof. From equation (13), we have

$$\begin{aligned} \min_{\hat{\mathbf{y}}} \mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \min \Bigg\{ & \min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) \;\; s.t. \;\; g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0, \\ & \alpha_{\mathbf{x}} \left(\min_{\hat{\mathbf{y}}} \; f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \;\; s.t. \;\; g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \right) \Bigg\} \\ & = \min \left\{ p^*_{\mathbf{x}}, \alpha_{\mathbf{x}} q^*_{\mathbf{x}} \right\} \end{aligned} \tag{14}$$

Because $\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}})$ is consistent, namely, $\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}}$, we have

$$\min \left\{ p^*_{\mathbf{x}}, \alpha_{\mathbf{x}} q^*_{\mathbf{x}} \right\} = p^*_{\mathbf{x}} \tag{15}$$

From equations (14) and (15), the proof follows. □

We assume that $f_{\mathbf{x}}(\mathbf{y})$ and $g_{\mathbf{x}}(\mathbf{y})$ are bounded functions, namely for any assignment $(\mathbf{x}, \mathbf{y})$, $l_f \leq f_{\mathbf{x}}(\mathbf{y}) \leq u_f$ and $l_g \leq g_{\mathbf{x}}(\mathbf{y}) \leq u_g$ where $-\infty < s < \infty$ and $s \in \{l_f, u_f, l_g, u_g\}$. Also, for simplicity, we assume that $f_{\mathbf{x}}(\mathbf{y})$ is a strictly positive function, namely $l_f > 0$.

Thus, based on the assumptions given above, we have

$$\frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}} \leq \frac{u_f}{l_f} \text{ and } 0 < \alpha_{\mathbf{x}} \leq \frac{u_f}{l_f}$$

The above assumptions will ensure that the gradients are bounded, because $\alpha_{\mathbf{x}}$, f and g are bounded, and both $p^*_{\mathbf{x}}$ and $q^*_{\mathbf{x}}$ are greater than zero.

Next, we show how to compute an upper bound on $\alpha_{\mathbf{x}}$ using $\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}}$, thus ensuring that we have an optimal loss function. The terms in the numerator ($p^*_{\mathbf{x}}$) and denominator ($q^*_{\mathbf{x}}$) require solving two instances of the CMPE task. Since solving CMPE exactly is impractical and moreover, since we are interested in self-supervised methods where we do not assume access to such a solver, we propose to lower bound $q^*_{\mathbf{x}}$ and upper bound $p^*_{\mathbf{x}}$.

For a given instance $\mathbf{x}$, a lower bound on $q^*_{\mathbf{x}}$ can be obtained using the Lagrangian relaxation method described in section 2.2 (see Eq. (4)) for discrete variables and the Reformulation-Linearization method described in Sherali and Tuncbilek (1992) for continuous variables. On the other hand, any feasible solution can serve as an upper bound for $p^*_{\mathbf{x}}$. A simple yet efficient approach is to begin with a loose upper bound by upper bounding the MPE task: $\max_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y})$, using fast algorithms such as mini-bucket elimination (Dechter and Rish, 2003) or fast linear programming based approximations (Ihler et al., 2012; Globerson and Jaakkola, 2007) and then keep track of feasible solutions during batch-style gradient descent.

In summary, we proposed a new loss function which uses the quantity $\alpha_{\mathbf{x}}$. When the neural network predicts a feasible $\hat{\mathbf{y}}$, the loss equals f, whereas when it predicts an infeasible $\hat{\mathbf{y}}$, the loss is such that the infeasible solution can quickly be pushed towards a feasible solution (because it uses gradients from g). A key advantage of our proposed loss function is that $\alpha_{\mathbf{x}}$ is not treated as an optimization variable, and a bound on it can be pre-computed for each example $\mathbf{x}$.

4.1 Making The Loss Function Smooth and Continuous

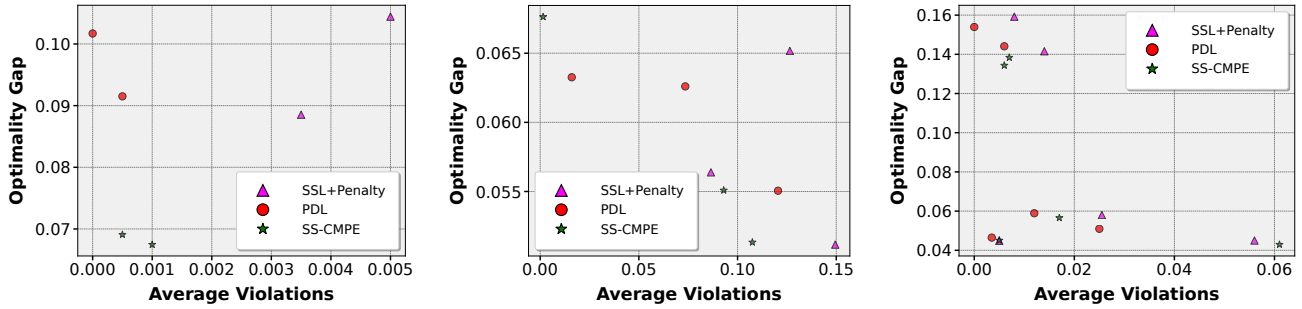
The loss function defined in Eq. (13) is continuous and differential everywhere except at $g_{\mathbf{x}}(\hat{\mathbf{y}}) = 0$. There is a *jump discontinuity* at $g_{\mathbf{x}}(\hat{\mathbf{y}}) = 0$ since $\lim_{g_{\mathbf{x}}(\hat{\mathbf{y}}) \to 0^-} f_{\mathbf{x}}(\hat{\mathbf{y}}) \neq \lim_{g_{\mathbf{x}}(\hat{\mathbf{y}}) \to 0^+} \alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}))$. To address this issue, we propose the following continuous approximation

$$\begin{aligned} \widetilde{\mathcal{L}}_{\mathbf{x}}(\hat{\mathbf{y}}) = & \Big((1 - \sigma(\beta g_{\mathbf{x}}(\hat{\mathbf{y}}))) \cdot [f_{\mathbf{x}}(\hat{\mathbf{y}})] \Big) + \\ & \Big(\sigma(\beta g_{\mathbf{x}}(\hat{\mathbf{y}})) \cdot [\alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\})] \Big) \end{aligned} \tag{16}$$

where $\sigma(.)$ is the sigmoid function and $\beta \geq 0$ is a hyper-parameter that controls the steepness of the sigmoid. At a high level, the above continuous approximation uses a sigmoid function to approximate a Heaviside step function.

5 EXPERIMENTAL EVALUATION

In this section, we thoroughly evaluate the effectiveness of our proposed neural networks based solvers for CMPE. We evaluate the competing methods on several test problems using three criteria: optimality gap (relative difference between the optimal solution and the one found by the method), constraint violations (percentage of time the method outputs an infeasible solution), and training and inference times.



(a) Grids (b) Segmentation (c) Tractable Models

Figure 2: Optimality Gap (avg %) and Average Violations for Self-Supervised methods

Table 1: Average gap and constraint violations over test samples for models from the UAI competition. ± denotes standard deviation. **Bold** values indicate the methods with the highest performance. Underlined values denote significant violations, particularly those exceeding a threshold of 0.15. For these methods, the gap values are not considered in our analysis.

Methods		Segment12	Segment14	Segment15	Grids17	Grids18
ILP Obj.		463.454	471.205	514.287	2879.469	4160.196
$\texttt{SL}_{pen}$	Gap	0.053 ± 0.043	0.053 ± 0.043	0.053 ± 0.041	0.092 ± 0.070	0.082 ± 0.065
	Violations	0.238 ± 0.426	0.248 ± 0.432	0.153 ± 0.361	0.054 ± 0.226	0.053 ± 0.224
$\texttt{SSL}_{pen}$	Gap	**0.051 ± 0.042**	0.065 ± 0.048	**0.056 ± 0.044**	0.089 ± 0.055	0.104 ± 0.062
	Violations	0.149 ± 0.357	0.127 ± 0.332	0.086 ± 0.281	0.004 ± 0.059	0.005 ± 0.071
PDL	Gap	0.063 ± 0.050	0.055 ± 0.042	0.063 ± 0.049	0.102 ± 0.059	0.092 ± 0.061
	Violations	**0.073 ± 0.261**	0.120 ± 0.326	0.016 ± 0.126	**0.000 ± 0.000**	**0.001 ± 0.022**
SS-CMPE	Gap	0.055 ± 0.045	**0.051 ± 0.040**	0.068 ± 0.051	**0.067 ± 0.049**	**0.069 ± 0.051**
	Violations	0.093 ± 0.291	**0.107 ± 0.415**	**0.002 ± 0.039**	0.001 ± 0.032	**0.001 ± 0.022**

5.1 The Loss Functions: Competing Methods

We trained several neural networks to minimize both supervised and self-supervised loss functions. We evaluated both MSE and MAE supervised losses with and without penalty coefficients (see section 3). In the main paper, we show results on the best performing supervised loss, which is MSE with penalty, denoted by $\texttt{SL}_{pen}$ (results for other supervised loss functions are provided in the supplement).

For self-supervised loss, we experimented with the following three approaches: (1) penalty-based method, (2) ALM, which uses a primal-dual loss (PDL), and the approach described in section 4. We will refer to these three schemes as $\texttt{SSL}_{pen}$, PDL, and SS-CMPE, respectively. We used the experimental setup described by Park and Van Hentenryck (2022) for tuning the hyperparameters of PDL and $\texttt{SSL}_{pen}$. For the SS-CMPE method, we employed a grid search approach to determine the optimal values for β. The range of values considered for β was $\{0.1, 1.0, 2.0, 5.0, 10.0, 20.0\}$.

Note that all methods used the same neural network architecture (described in the supplement) except PDL, which uses two neural networks. We obtained the ground-truth for the supervised training by solving the original ILP problem using SCIP (Achterberg, 2009) and Gurobi (Gurobi Optimization, 2021). We report the objective values of the ILP solutions in each table (see Tables 1, 2, and 3).

5.2 Datasets and Benchmarks

We evaluate the competing algorithms ($\texttt{SL}_{pen}$, $\texttt{SSL}_{pen}$, PDL, and SS-CMPE) on a number of log-linear Markov networks ranging from simple models (low treewidth) to high treewidth models. The simple models comprise of learned tractable probabilistic circuits (Choi et al., 2020) without latent variables, specifically, cutset networks (Rahman et al., 2014) from benchmark datasets used in the literature. The complex, high treewidth models are sourced from past UAI inference competitions (Elidan and Globerson, 2010). Finally, we evaluated all methods on the task of generating adversarial examples for neural network classifiers.

5.3 High Tree-Width Markov Networks and Tractable Probabilistic Circuits

Our initial series of experiments focus on high treewidth Grids and Image Segmentation Markov networks from the UAI inference competitions (Gogate, 2014, 2016). In this investigation, we generated CMPE problems by utilizing the model employed in the UAI competitions, denoted as M_1. Subsequently, M_2 was created by adjusting the parameters of M_1 while incorporating a noise parameter ϵ drawn from

Table 2: Average gap and constraint violations over test samples from tractable probabilistic models. ± denotes standard deviation. **Bold** values indicate the methods with the highest performance. Underlined values denote significant violations, particularly those exceeding a threshold of 0.15. For these methods, the gap values are not considered in our analysis.

Methods		AD	BBC	DNA	20 NewsGroup	WebKB1
ILP Obj.		2519.128	871.567	221.119	921.702	824.493
$\texttt{SL}_{pen}$	Gap	0.156 ± 0.057	0.036 ± 0.027	0.143 ± 0.113	**0.041 ± 0.031**	**0.044 ± 0.035**
	Violations	0.135 ± 0.341	0.237 ± 0.425	0.151 ± 0.358	0.084 ± 0.277	0.070 ± 0.254
$\texttt{SSL}_{pen}$	Gap	0.159 ± 0.055	0.045 ± 0.033	0.142 ± 0.116	0.045 ± 0.036	0.058 ± 0.043
	Violations	0.008 ± 0.089	0.056 ± 0.230	0.014 ± 0.118	0.005 ± 0.071	0.025 ± 0.158
PDL	Gap	0.154 ± 0.055	0.051 ± 0.036	0.144 ± 0.117	0.046 ± 0.035	0.059 ± 0.043
	Violations	**0.000 ± 0.000**	**0.025 ± 0.156**	**0.006 ± 0.077**	**0.004 ± 0.059**	**0.012 ± 0.109**
SS-CMPE	Gap	**0.134 ± 0.054**	**0.043 ± 0.033**	**0.138 ± 0.112**	0.045 ± 0.035	0.057 ± 0.043
	Violations	0.006 ± 0.077	0.056 ± 0.230	0.007 ± 0.083	0.005 ± 0.071	0.016 ± 0.126

a normal distribution with mean 0 and variance $\sigma^2 = 0.1$. To select q, we randomly generated 100 samples, sorted them based on their weight, and then selected the weight of the 10th, 30th, 60th, 80th, and 90th sample as a value for q. We assess the impact of changing the value of q in the supplement. Experiments in the main body of the paper use q equal to the weight of the 80th random sample. For each network, we randomly chose 60% of variables as evidence ($\mathbf{X}$) and the remaining as query variables ($\mathbf{Y}$). For both the UAI models and tractable probabilistic circuits, we generated 10K samples from M_1, and used 9K for training and 1K for testing. We used 5-fold cross validation for selecting the hyperparameters.

The results for the UAI datasets are shown in Table 1. We see that for the majority of the datasets, our method produces solutions with superior gap values compared to the other methods. Even in situations where our methods do not achieve better gap values, they exhibit fewer violations. This demonstrates the effectiveness and robustness of our methods in generating solutions that strike a balance between optimizing the objective function and maintaining constraint adherence. For certain datasets, our methods exhibit significantly lower constraint violations, even up to 10 times less than supervised methods.

In the next phase of our study, we employed MPE (Most Probable Explanation) tractable models, which were learned on five high-dimensional datasets (see Lowd and Davis (2010) for details in the datasets): DNA, NewsGroup (c20ng), WebKB1 (cwebkb), AD, and BBC. These learned models served as $\mathcal{M}_1$. We then applied Gaussian noise as described earlier to generate $\mathcal{M}_2$ based on $\mathcal{M}_1$. A similar trend can be observed for tractable probabilistic models in Table 2, where our method consistently outperforms the other self-supervised methods across all datasets. Not only does our approach exhibit superior performance in terms of gap values, but it also demonstrates comparable constraint violations. When comparing with the supervised method, our proposed algorithm exhibits significantly fewer constraint violations while maintaining a better or comparable gap value. This emphasizes the strength of our method in effectively balancing the optimization objectives and constraint adherence, thereby offering improved overall performance compared to both the self-supervised and supervised approaches in the context of tractable probabilistic models. In Figure 2, we present the average optimality gap and average violations for different dataset groups. It is important to note that results closer to the origin indicate better performance.

5.4 Adversarial Modification on the MNIST Dataset

Table 3: Performance comparison of supervised and self-supervised methods. The table presents the average objective value, gap, and constraint violations over the test examples, along with the training and inference time required for each method for adversarial example generation. **Bold** values signify the methods that achieved the best scores.

Methods	Obj. Value	Gap	Violation	Time in seconds	
				Train	Inf.
ILP	30.794	0.000	0.000	NA	5.730
$\texttt{SL}_{pen}$	63.670	1.069	0.071	57534.4	0.003
$\texttt{SSL}_{pen}$	76.316	1.480	0.052	**469.540**	0.003
PDL	66.400	1.158	0.055	839.025	0.003
SS-CMPE	**62.400**	**1.028**	**0.021**	520.149	0.003

We also evaluated our approach on the task of adversarial example generation for discriminative classifiers, specifically neural networks. Adversarial examples play a crucial role in assessing the robustness of models and facilitating the training of more resilient models. The task of Adversarial Example Generation involves producing new images by making minimal modifications to input images that are mis-classified by the model. Rahman et al. (2021) showed that this problem can be reduced to CMPE. Formally, let $\mathcal{G}$ be a differentiable, continuous function defined over a set of inputs $\mathcal{X}$. Given an assignment $\mathcal{X} = x$, we introduce the decision variable $\mathcal{D}$, which takes the value d when $\mathcal{G} > 0$ and $\bar{d}$ otherwise. In the context of adversarial attacks, given

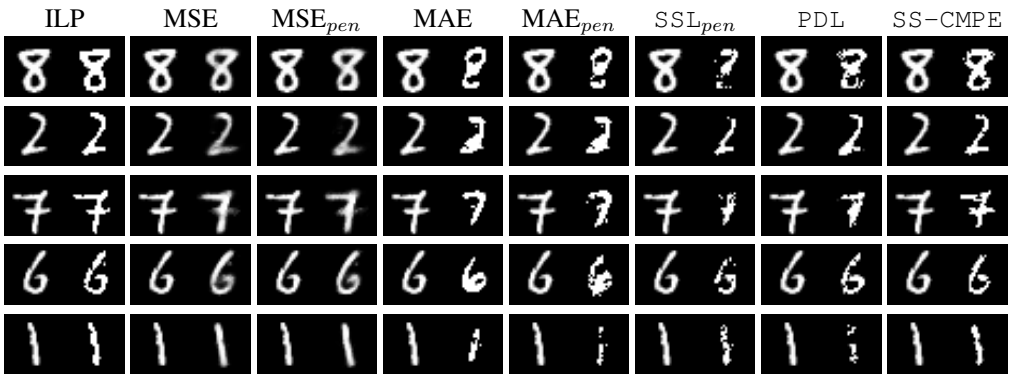


Figure 3: Qualitative results on the adversarially generated MNIST digits. Each row represents an original image followed by a corresponding image generated adversarially by 8 different methods: ILP, MSE, SL+Penalty, MAE, MAE+Penalty, SSL$_{pen}$, PDL, and SS−CMPE .

an image x, our objective is to generate a new image x' such that the distance between x and x' is minimized and the decision is flipped (namely $\mathcal{G} < 0$). We used a log-linear model $\mathcal{F}$ to represent the sum absolute distance between the pixels. Then the task of adversarial example generation can be formulated as the following CMPE problem: maximize $\sum_{f \in \mathcal{F}} f\left(x'|x\right)$ s.t. $\mathcal{G}\left(x'|x\right) \leq 0$.

We evaluated the algorithms using the MNIST handwritten digit dataset (LeCun and Cortes, 2010). We trained a multi-layered neural network having >95% test accuracy and used it as our $\mathcal{G}$ function. To generate adversarial examples corresponding to a given test example, we trained an autoencoder $A : X \longrightarrow X'$ using four loss functions corresponding to SL$_{pen}$, SSL$_{pen}$, PDL and SS−CMPE. We used the default train-test split (10K examples for testing and 60K for training).

Table 3 shows quantitative results comparing our proposed SS−CMPE method with other competing methods. We can clearly see that SS−CMPE is superior to competing self-supervised (SSL$_{pen}$ and PDL) and supervised methods (SL$_{pen}$) in terms of both constraint violations and optimality gap. The second best method in terms of optimality gap is SL$_{pen}$. However, its constraint violations are much higher, and its training time is significantly larger because it needs access to labeled data, which in turn requires using computationally expensive ILP solvers. The training time of SS−CMPE is much smaller than PDL (because the latter uses two networks) and is only slightly larger than SSL$_{pen}$.

Figure 3 shows qualitative results on adversarial modification to the MNIST digits $\{1, 2, 6, 7, 8\}$ by all the eight methods. The CMPE task minimally changes an input image such that the corresponding class is flipped according to a discriminative classifier. MSE and our proposed method SS−CMPE are very competitive and were able to generate visually indistinguishable, high-quality modifications whereas the other methods struggled to do so.

Summary: Our experiments show that SS−CMPE consistently outperforms competing self-supervised methods, PDL and SSL$_{pen}$, in terms of optimality gap and is comparable to PDL in terms of constraint violations. The training time of SS−CMPE is smaller than PDL (by half as much) and is slightly larger than SSL$_{pen}$. However, it is considerably better than SSL$_{pen}$ in terms of constraint violations. SS−CMPE also employs fewer hyperparameters as compared to PDL.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a new self-supervised learning algorithm for solving the constrained most probable explanation task which at a high level is the task of optimizing a multilinear polynomial subject to a multilinear constraint. Our main contribution is a new loss function for self-supervised learning which is derived from first principles, has the same set of global optima as the CMPE task, and operates exclusively on the primal variables. It also uses only one hyperparameter in the continuous case and two hyperparameters in the discrete case. Experimentally, we evaluated our new self-supervised method with penalty-based and Lagrangian duality-based methods proposed in literature and found that our method is often superior in terms of optimality gap and training time (also requires less hyperparameter tuning) to the Lagrangian duality-based methods and also superior in terms of optimality gap and the number of constraint violations to the penalty-based methods.

Our proposed method has several limitations and we will address them in future work. First, it requires a bound for $\alpha_{\mathbf{x}}$. This bound is easy to obtain for graphical models/multilinear objectives but may not be straightforward to obtain for arbitrary non-convex functions. Second, the ideal objective in the infeasible region should be proportional to $g_{\mathbf{x}}(\mathbf{y})$ but our method uses $\alpha_{\mathbf{x}}(f_{\mathbf{x}}(\mathbf{y}) + g_{\mathbf{x}}(\mathbf{y}))$.

ACKNOWLEDGMENTS

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) under contract number HR001122S0039 and by the National Science Foundation grant IIS-1652835.

Bibliography

Achterberg, T. (2009). Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41.

Achterberg, T., Berthold, T., Koch, T., and Wolter, K. (2008). Constraint integer programming: A new approach to integrate cp and mip. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 5th International Conference, CPAIOR 2008 Paris, France, May 20-23, 2008 Proceedings 5*, pages 6–20. Springer.

Choi, A. and Darwiche, A. (2011). Relax, compensate and then recover. In Onada, T., Bekki, D., and McCready, E., editors, *New Frontiers in Artificial Intelligence*, pages 167–180, Berlin, Heidelberg. Springer Berlin Heidelberg.

Choi, A., Xue, Y., and Darwiche, A. (2012). Same-decision probability: A confidence measure for threshold-based decisions. *International Journal of Approximate Reasoning*, 53(9):1415–1428.

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*.

Cui, Z., Wang, H., Gao, T., Talamadupula, K., and Ji, Q. (2022). Variational message passing neural network for maximum-a-posteriori (map) inference. In *Uncertainty in Artificial Intelligence*, pages 464–474. PMLR.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A. and Hirth, A. (2020). On the reasons behind decisions. In *Twenty Fourth European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 712–720. IOS Press.

Darwiche, A. and Hirth, A. (2023). On the (complete) reasons behind decisions. *Journal of Logic, Language and Information*, 32(1):63–88.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85.

Dechter, R. and Rish, I. (2003). Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153.

Donti, P. L., Rolnick, D., and Kolter, J. Z. (2021). Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*.

Elidan, G. and Globerson, A. (2010). *The 2010 UAI Approximate Inference Challenge*. Published: Available online at: http://www.cs.huji.ac.il/project/UAI10/index.php.

Fioretto, F., Mak, T. W., and Van Hentenryck, P. (2020). Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 630–637.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Globerson, A. and Jaakkola, T. (2007). Fixing max-product: Convergent message passing algorithms for map lp-relaxations. *Advances in neural information processing systems*, 20:553–560.

Gogate, V. (2014). Results of the 2014 UAI competition. https://personal.utdallas.edu/~vibhav.gogate/uai14-competition/index.html.

Gogate, V. (2016). Results of the 2016 UAI competition. https://personal.utdallas.edu/~vibhav.gogate/uai16-competition/index.html.

Gurobi Optimization, L. (2021). Gurobi optimizer reference manual.

Horst, R. and Tuy, H. (1996). *Global Optimization: Deterministic Approaches*. Springer Berlin Heidelberg.

Ihler, A. T., Flerova, N., Dechter, R., and Otten, L. (2012). Join-graph based cost-shifting schemes. *arXiv preprint arXiv:1210.4878*.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv*.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Komodakis, N., Paragios, N., and Tziritas, G. (2007). Mrf optimization via dual decomposition: Message-passing revisited. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE.

Kotary, J., Fioretto, F., and Hentenryck, P. V. (2021). Learning hard optimization problems: A data generation perspective. *arXiv preprint arXiv: Arxiv-2106.02601*.

Kuck, J., Chakraborty, S., Tang, H., Luo, R., Song, J., Sabharwal, A., and Ermon, S. (2020). Belief propagation neural networks. *Advances in Neural Information Processing Systems*, 33:667–678.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Liu, T. and Cherian, A. (2023). Learning a constrained optimizer: A primal method. In *AAAI 2023 Bridge on Constraint Programming and Machine Learning*.

Lowd, D. and Davis, J. (2010). Learning Markov Network Structure with Decision Trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE.

Marinescu, R. and Dechter, R. (2009). Memory intensive AND/OR search for combinatorial optimization in graphical models. *AI Journal*, 173(16-17):1492–1524.

Marinescu, R. and Dechter, R. (2012). Best-First AND/OR Search for Most Probable Explanations. *CoRR*, abs/1206.5268.

Nellikkath, R. and Chatzivasileiadis, S. (2021). Physics-informed neural networks for ac optimal power flow. *arXiv preprint arXiv: Arxiv-2110.02672*.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, NY, USA, 2e edition.

Park, S. and Van Hentenryck, P. (2022). Self-supervised primal-dual learning for constrained optimization. *arXiv preprint arXiv:2208.09046*.

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.

Rahman, T., Rouhani, S., and Gogate, V. (2021). Novel upper bounds for the constrained most probable explanation task. *Advances in Neural Information Processing Systems*, 34:9613–9624.

Rouhani, S., Rahman, T., and Gogate, V. (2018). Algorithms for the nearest assignment problem. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5096–5102. ijcai.org.

Rouhani, S., Rahman, T., and Gogate, V. (2020). A novel approach for constrained optimization in graphical models. *Advances in Neural Information Processing Systems*, 33:11949–11960.

Satorras, V. G. and Welling, M. (2021). Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 685–693. PMLR.

Sherali, H. D. and Adams, W. P. (2009). A reformulation-linearization technique (rlt) for semi-infinite and convex programs under mixed 0-1 and general discrete restrictions. *Discrete Applied Mathematics*, 157(6):1319–1333. Reformulation Techniques and Mathematical Programming.

Sherali, H. D. and Tuncbilek, C. H. (1992). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2:101–112.

Ucla-Starai (2023). Density-Estimation-Datasets. [Online; accessed 17. May 2023].

Van Haaren, J. and Davis, J. (2012). Markov Network Structure Learning: A Randomized Feature Generation Approach. *AAAI*, 26(1):1148–1154.

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005). Map estimation via agreement on trees: message-passing and linear programming. *IEEE transactions on information theory*, 51(11):3697–3717.

Wu, B., Shen, L., Zhang, T., and Ghanem, B. (2020). Map inference via l2 sphere linear program reformulation. *International Journal of Computer Vision*, 128(7):1913–1936.

Zamzam, A. and Baker, K. (2019). Learning optimal solutions for extremely fast ac optimal power flow. *arXiv preprint arXiv: Arxiv-1910.01213*.

Zhang, Z., Wu, F., and Lee, W. S. (2020). Factor graph neural networks. *Advances in Neural Information Processing Systems*, 33:8577–8587.

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]
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). [Yes]
   (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. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (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]

A DERIVING UPPER BOUNDS FOR $p^*_{\mathbf{x}}$ AND LOWER BOUNDS FOR $q^*_{\mathbf{x}}$

To compute the value of $\alpha_{\mathbf{x}}$, we utilize the following equation:

$$\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}} \tag{17}$$

We choose to set a lower bound for the denominator $q^*_{\mathbf{x}}$ and an upper bound for the numerator $p^*_{\mathbf{x}}$ due to the computational challenges of finding exact solutions for the CMPE task.

To estimate an upper bound for the optimal value ($p^*_{\mathbf{x}}$) of the constrained optimization problem

$$\min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ s.t. } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0, \tag{18}$$

we begin by seeking a loose upper bound through solving the unconstrained task $\max_{\mathbf{y}} \ f_{\mathbf{x}}(\hat{\mathbf{y}})$ by utilizing mini-bucket elimination (Dechter and Rish, 2003). Subsequently, feasible solutions are tracked during batch-style gradient descent to refine the initial upper bound (note that the weight of any feasible solution is an upper bound on $p^*_{\mathbf{x}}$). For each iteration, the feasible solution with the optimal objective value for each example is stored and subsequently utilized.

To derive a lower bound for $q^*_{\mathbf{x}}$, which represents the optimal solution for the following constrained optimization problem,

$$\min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \quad \text{s.t.} \quad g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0, \tag{19}$$

we can employ the methodologies delineated in Rahman et al. (2021). These techniques provide a mechanism for either upper bounding or lower bounding the CMPE task, contingent on whether it is formulated as a maximization or minimization problem, respectively.

To establish a lower bound for $q^*_{\mathbf{x}}$, the constrained optimization task is initially transformed into an unconstrained formulation via Lagrange Relaxation. This results in the following optimization problem:

$$\max_{\mu \geq 0} \min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) + (1 - \mu) \times g_{\mathbf{x}}(\hat{\mathbf{y}}) \tag{20}$$

Here, μ denotes the Lagrangian multiplier. By addressing this dual optimization problem, we enhance the precision of the lower bound for $q^*_{\mathbf{x}}$. For the inner minimization task, the mini-bucket elimination method is employed. The outer maximization is solved through the utilization of sub-gradient descent.

B EXTENSIONS

Adding a Penalty for Constraint Violations. A penalty of the form $\max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2$ can be easily added to the loss function as described by the following equation

$$\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \begin{cases} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0 \\ \alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}})) + \rho \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2 \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \end{cases}$$

where $\rho \geq 0$ is a hyperparameter.

C EXPERIMENTAL SETUP AND DETAILS

C.A Dataset and Model Description

Table ST4 provides a comprehensive overview of the characteristics of each binary dataset, including the number of variables and functions present in each dataset. These datasets were specifically chosen to provide diverse and representative examples for evaluating the performance and scalability of our algorithms.

We used the following two classes of Markov networks from the UAI competitions (Elidan and Globerson, 2010; Gogate, 2014, 2016): Ising models (Grids) and Image Segmentation networks. Specifically, we used the Grids_17 and Grids_18 networks and Segmentation_12, Segmentation_14 and Segmentation_15 networks.

We learned MPE tractable cutset networks without latent variables using the scheme of Rahman et al. (2014) on five high-dimensional datasets: DNA (Van Haaren and Davis, 2012; Ucla-Starai, 2023), NewsGroup (c20ng) (Lowd and Davis, 2010; Ucla-Starai, 2023), WebKB1 (cwebkb) (Lowd and Davis, 2010; Ucla-Starai, 2023), AD (Van Haaren and Davis, 2012; Ucla-Starai, 2023), and BBC (Van Haaren and Davis, 2012; Ucla-Starai, 2023). These datasets are widely used in the probabilistic circuits literature (Lowd and Davis, 2010). Note that CMPE is intractable on these models even though MPE is tractable.

Table ST4: Dataset and Model Descriptions

Dataset	**Number of Variables**	**Number of Functions**
Tractable Probabilistic Circuits		
AD	1556	1556
BBC	1058	1058
20NewsGroup	910	910
WebKB	839	839
DNA	180	180
High Tree-Width Markov Networks		
Grids17	400	1160
Grids18	400	1160
Segmentation12	229	851
Segmentation14	226	845
Segmentation15	232	863

C.B Data Generation

Recall that the CMPE problem uses two Markov networks M_1 and M_2, and a value of q. We used the original Markov networks (chosen from the UAI competitions or learned from data) as M_1 and generated M_2 by adding a value v, which was randomly sampled from a normal distribution with mean 0 and variance 0.1, to each entry in each potential in M_1. We used the following strategy to generate q. We generated 100 random samples from M_2, sorted them according to their weight w.r.t. M_2, and then chose the 10th, 30th, 60th, and 90th sample as a value for q. At a high level, as we go from the 10th sample to the 90th sample, namely as q increases, the constraint (weight w.r.t. M_2 is less than or equal to q) becomes less restrictive. In other words, as we increase q, the set of feasible solutions increases (or stays the same). For each value of q, we use 60% of the variables as evidence variables $\mathbf{X}$ and the remaining as $\mathbf{Y}$.

For each CMPE problem, we generated 10000 samples, used the first 9000 samples for training and the remaining 1000 samples for testing. For the supervised methods, we generated the optimal assignment to $\mathbf{Y}$ using an integer linear programming solver called SCIP (Achterberg, 2009).

For our proposed scheme, which we call `SS-CMPE`, we used approach described in Section A to find the upper bound of $p^*_{\mathbf{x}}$ and the lower bound of $q^*_{\mathbf{x}}$

Note that CMPE is a much harder task than MPE. Our scheme can be easily adapted to MPE, all we have to do is use f to yield a supervised scheme.

C.C Architecture Design and Training Procedure

In our experimental evaluations, we employed a Multi-Layer Perceptron (MLP) with a Rectified Linear Unit (ReLU) activation function for all hidden layers. The final layer of the MLP utilized a sigmoid activation function, as it was necessary to obtain outputs within the range of [0, 1] for all our experiments. Each fully connected neural network in our study consisted of three hidden layers with respective sizes of [128, 256, and 512]. We maintained this consistent architecture across all our supervised (Zamzam and Baker, 2019; Nellikkath and Chatzivasileiadis, 2021) and self-supervised (Park and Van Hentenryck, 2022; Donti et al., 2021) methods. It is important to highlight that in the adversarial modification

experiments, the neural network possessed an equal number of inputs and outputs, specifically set to 28×28 (size of an image in MNIST). However, in the remaining two experiments concerning probabilistic graphical models, the input size was the number of evidence variables ($|\mathbf{X}|$), while the output size was $|\mathbf{Y}|$.

For PDL (Park and Van Hentenryck, 2022), the dual network had one hidden layer with 128 nodes. The number of outputs of the dual network corresponds to the number of constraints in the optimization problem. It is worth emphasizing that our method is not constrained to the usage of Multi-Layer Perceptrons (MLPs) exclusively, and we have the flexibility to explore various neural network architectures. This flexibility allows us to consider and utilize alternative architectures that may better suit the requirements and objectives of other optimization tasks.

Regarding the training process, all methods underwent 300 epochs using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $1e^{-3}$. We employed a Learning Rate Scheduler to dynamically adapt the learning rate as the loss reaches a plateau. The training and testing processes for all models were conducted on a single NVIDIA A40 GPU.

C.D Hyper-parameters

The number of instances in the minibatch was set to 128 for all the experiments. We decay the learning rate in all the experiments by 0.9 when the loss becomes a plateau. Given the empirical observations that learning rate decay often leads to early convergence in most cases and does not yield beneficial results for the supervised baselines, we have made the decision not to apply learning rate decay to these methods. This choice is based on the understanding that the baselines perform optimally without this particular form of learning rate adjustment. For detailed information regarding the hyper-parameters utilized in the benchmarking methods, we refer readers to the corresponding papers associated with each method. As stated in the main text, the optimal hyperparameters were determined using a grid search approach. For the SS−CMPE method, for each dataset, the hyperparameters were selected from the following available options -

- β - $\{0.1, 1.0, 2.0, 5.0, 10.0, 20.0\}$
- ρ - $\{0.01, 0.1, 1, 10, 100\}$

In the optimization problem of SS−CMPE, the parameter ρ is employed to penalize the violation of constraints. The methodology for this approach, denoted as SS−CMPE $_{pen}$, is explained in detail in Section B. The corresponding experiments are presented in tables ST8 through ST17.

C.E The Loss Function: Competing Methods

We evaluated eight different loss functions, including our method to train a deep neural network to solve our CMPE tasks. The loss functions used for supervised training are 1) Mean-Squared-Error (MSE), and 2) Mean-Absolute-Error (MAE). Both of these losses were then extended to incorporate penalty terms as suggested by (Nellikkath and Chatzivasileiadis, 2021). We denote them as SL+Penalty and MAE+Penalty. We evaluated the self-supervised loss proposed by (Donti et al., 2021) (SSL_{pen}) and by (Park and Van Hentenryck, 2022) (PDL). Finally, we extend our self-supervised CMPE loss function to incorporate the penalty term $\max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2$ (see section B). We denote it as SS−CMPE $_{pen}$.

D EXAMINING THE INFLUENCE OF q: EVALUATING THE PERFORMANCE OF OUR PROPOSED METHOD FOR CHALLENGING PROBLEM INSTANCES

To determine the value of q, a total of 100 random samples were generated, and their weights were calculated. Subsequently, the samples were sorted based on their weight in ascending order. The weight values corresponding to the 10th, 30th, 60th, and 90th samples were then chosen as the values for q. For each value of q, we compare the average gap and violations obtained by our method (SS−CMPE and SS−CMPE $_{pen}$) against six other supervised and self-supervised methods. Tables ST8 through ST17 show the scores obtained by each of the eight methods along with their standard deviations on the generated test problems. This study investigates the performance of each method in finding near-optimal feasible solutions for difficult problems which is directly controlled by the percentile rank of q; problems with a q value in the 10th and 30th percentile are considered harder problems to solve as the size of the feasible region is considerably smaller than the size of the infeasible region. As a result all methods have higher violations on these problems than the problems with a q value in the 60th and 90th percentile.

Table ST5 presents a summary of the performances of SS−CMPE and other *supervised* methods based on their average gap and the average number of violations on the test data for different values of q. We compute the minimum gap and violations achieved among the four supervised methods MSE, SL+Penalty, MAE, and MAE+Penalty and label them as the best supervised method. We choose the minimum gap and violations among SS−CMPE and SS−CMPE $_{pen}$ and label them under the unified term best SS−CMPE. We observe that best SS−CMPE consistently has significantly lower violations than the best supervised method in all the problem instances, and its gap is often comparable to the gap achieved by the best supervised method, winning when compared to the average gap.

Table ST6 presents a similar summary of the performances of SS−CMPE and other *self-supervised* methods. We compute the minimum gap and violations achieved among the 2 self-supervised methods SSL$_{pen}$ and PDL and label them as the best SSL method. As before, we choose the minimum gap and violations among SS−CMPE and SS−CMPE $_{pen}$ and label them under the unified term best SS−CMPE. Although self-supervised methods have larger gaps compared to supervised methods but lesser violations, we observe that SS−CMPE continues to consistently achieve significantly lower violations than the best performing self-supervised method in all the problem instances, and its gap is often comparable to the gap achieved by the best supervised method, winning when compared to the average gap.

Finally, in table ST7, we present a quick summary of the performances of our best SS−CMPE method vs all other methods. We choose the minimum gap and violations achieved among the 6 other supervised and self-supervised methods and the minimum gap and violations among SS−CMPE and SS−CMPE $_{pen}$. In all problems and q values, the best SS−CMPE has significantly lower violations compared to other methods while having a very competitive gap.

D.A The Feasible-Only Optimality Gaps: Comparing Self-Supervised Approaches

From the results presented in tables ST5 through ST17, we observe that self-supervised approaches produce more feasible solutions compared to supervised approaches. In this section, we present the results of a controlled study that shows how each of the self-supervised approaches perform in terms of finding optimal solutions in the feasible region.

We selected a subset of problems from the test set on which all self-supervised methods, namely, SSL$_{pen}$ (Donti et al., 2021), PDL (Park and Van Hentenryck, 2022) and our method SS−CMPE and SS−CMPE $_{pen}$, obtained feasible solutions and this was done for each possible value of q. We then computed their gaps and compare them via figure SF4. Among the three methods analyzed, SS−CMPE and SS−CMPE $_{pen}$ consistently exhibits superior performance across the majority of cases. Its optimality gaps are significantly smaller compared to the other two methods. This finding suggests that SS−CMPE is more effective in minimizing the objective value and achieving solutions closer to optimality for the given examples.

Table ST5: Summary: best SS-CMPE vs other supervised methods including MSE, SL+Penalty, MAE, and MAE+Penalty. Bold represents the minimum gap, while underlined means the least violations

q		10		30		60		90	
Models /Dataset	Gap / Violations	best SS-CMPE	best supervised	best SS-CMPE	best supervised	best SS-CMPE	best supervised	best SS-CMPE	best supervised
Segment-12	gap	0.057	**0.054**	**0.051**	0.052	0.053	**0.051**	**0.050**	0.051
	violations	0.152	0.511	0.166	0.500	0.084	0.348	0.021	0.131
Segment-14	gap	0.050	**0.049**	**0.047**	0.049	0.048	0.048	0.051	0.051
	violations	0.134	0.691	0.088	0.616	0.066	0.410	0.046	0.207
Segment-15	gap	0.051	0.051	**0.050**	0.051	0.052	**0.051**	**0.051**	0.052
	violations	0.060	0.570	0.060	0.417	0.049	0.248	0.001	0.061
Grids-17	gap	0.072	**0.054**	0.069	**0.057**	**0.066**	0.067	0.063	**0.058**
	violations	0.035	0.304	0.013	0.125	0.002	0.044	0.001	0.002
Grids-18	gap	0.064	**0.056**	0.067	**0.060**	**0.060**	0.065	0.065	**0.064**
	violations	0.017	0.210	0.019	0.087	0.000	0.025	0.000	0.015
DNA	gap	0.138	**0.135**	0.138	**0.136**	0.137	**0.136**	0.139	**0.137**
	violations	0.013	0.434	0.002	0.448	0.001	0.281	0.001	0.089
20NewsGr.	gap	**0.043**	0.044	**0.045**	0.046	**0.044**	0.046	0.044	0.044
	violations	0.069	0.455	0.054	0.176	0.007	0.046	0.001	0.001
WebKB	gap	0.059	**0.054**	0.058	**0.054**	0.056	**0.054**	**0.053**	0.054
	violations	0.074	0.471	0.029	0.378	0.001	0.174	0.001	0.018
BBC	gap	0.038	**0.036**	0.043	**0.036**	0.042	**0.037**	0.040	**0.037**
	violations	0.074	0.657	0.067	0.557	0.056	0.384	0.002	0.151
Ad	gap	**0.129**	0.204	**0.131**	0.201	**0.130**	0.204	**0.131**	0.213
	violations	0.017	0.085	0.004	0.041	0.000	0.021	0.000	0.005
Average	gap	**0.070**	0.074	**0.070**	0.074	**0.069**	0.076	**0.069**	0.076
	violations	0.065	0.439	0.050	0.335	0.027	0.198	0.007	0.068

Table ST6: Summary: best SS-CMPE vs other self-supervised methods including SL_{pen}, and PDL. Bold represents the minimum gap, while underlined means the least violations

q		10		30		60		90	
Models/ Dataset	Gap/ Violations	best SS-CMPE	best SSL	best SS-CMPE	best SSL	best SS-CMPE	best SSL	best SS-CMPE	best SSL
Segment-12	gap	0.057	**0.054**	**0.051**	0.052	0.053	**0.051**	**0.050**	0.051
	violations	0.152	0.545	0.166	0.622	0.084	0.486	0.021	0.163
Segment-14	gap	**0.050**	0.058	**0.047**	0.050	**0.048**	0.050	**0.051**	0.053
	violations	0.134	0.507	0.088	0.414	0.066	0.394	0.046	0.207
Segment-15	gap	0.051	0.051	**0.050**	0.053	0.052	**0.051**	**0.051**	0.054
	violations	0.060	0.676	0.060	0.360	0.049	0.274	0.001	0.086
Grids-17	gap	**0.072**	0.086	**0.069**	0.079	**0.066**	0.093	**0.063**	0.087
	violations	0.035	0.043	0.013	0.003	0.002	0.001	0.001	0.014
Grids-18	gap	**0.064**	0.105	**0.067**	0.074	**0.060**	0.093	**0.065**	0.118
	violations	0.017	0.060	0.019	0.001	0.000	0.003	0.000	0.005
DNA	gap	**0.138**	0.140	**0.138**	0.141	**0.137**	0.139	**0.139**	0.143
	violations	0.013	0.048	0.002	0.062	0.001	0.003	0.001	0.004
20NewsGr	gap	0.043	0.043	**0.045**	0.046	**0.044**	0.045	**0.044**	0.046
	violations	0.069	0.278	0.054	0.129	0.007	0.024	0.001	0.001
WebKB	gap	0.059	**0.058**	0.058	**0.057**	**0.056**	0.057	**0.053**	0.057
	violations	0.074	0.149	0.029	0.096	0.001	0.056	0.001	0.013
BBC	gap	**0.038**	0.041	**0.043**	0.042	0.042	**0.038**	0.040	**0.039**
	violations	0.074	0.336	0.067	0.160	0.056	0.149	0.002	0.029
Ad	gap	**0.129**	0.135	**0.131**	0.140	**0.130**	0.142	**0.131**	0.134
	violations	0.017	0.055	0.004	0.006	0.000	0.013	0.000	0.004
Average	gap	**0.070**	0.077	**0.070**	0.073	**0.069**	0.076	**0.069**	0.078
	violations	0.065	0.270	0.050	0.185	0.027	0.140	0.007	0.053

Table ST7: Summary: best SS-CMPE has significantly lower violations compared to other methods on all the problems and over all the chosen q values. It has comparable gap to the other methods.

q		10		30		60		90	
Models /Datasets	Gap /Violations	best SS-CMPE	others	best SS-CMPE	others	best SS-CMPE	others	best SS-CMPE	others
Segment-12	Gap	0.057	**0.054**	**0.051**	0.052	0.053	**0.051**	**0.050**	0.051
	Violations	**0.152**	0.511	**0.166**	0.500	**0.084**	0.348	**0.021**	0.131
Segment-14	Gap	0.050	**0.049**	**0.047**	0.049	0.048	0.048	0.051	0.051
	Violations	**0.134**	0.507	**0.088**	0.414	**0.066**	0.394	**0.046**	0.207
Segment-15	Gap	0.051	0.051	**0.050**	0.051	0.052	**0.051**	**0.051**	0.052
	Violations	**0.060**	0.570	**0.060**	0.360	**0.049**	0.248	**0.001**	0.061
Grids-17	Gap	0.072	**0.054**	0.069	**0.057**	**0.066**	0.067	0.063	**0.058**
	Violations	**0.035**	0.043	**0.013**	0.003	**0.002**	0.001	**0.001**	0.002
Grids-18	Gap	0.064	**0.056**	0.067	**0.060**	**0.060**	0.065	0.065	**0.064**
	Violations	**0.017**	0.060	**0.019**	0.001	**0.000**	0.003	**0.000**	0.005
DNA	Gap	0.138	**0.135**	0.138	**0.136**	0.137	**0.136**	0.139	**0.137**
	Violations	**0.013**	0.048	**0.002**	0.062	**0.001**	0.003	**0.001**	0.004
20NewsGr	Gap	0.043	0.043	**0.045**	0.046	**0.044**	0.045	**0.044**	0.044
	Violations	**0.069**	0.278	**0.054**	0.129	**0.007**	0.024	**0.001**	0.001
WebKB	Gap	0.059	**0.054**	0.058	**0.054**	0.056	**0.054**	**0.053**	0.054
	Violations	**0.074**	0.149	**0.029**	0.096	**0.001**	0.056	**0.001**	0.013
BBC	Gap	0.038	**0.036**	0.043	**0.036**	0.042	**0.037**	0.040	**0.037**
	Violations	**0.074**	0.336	**0.067**	0.160	**0.056**	0.149	**0.002**	0.029
Ad	Gap	**0.129**	0.135	**0.131**	0.140	**0.130**	0.142	**0.131**	0.134
	Violations	**0.017**	0.055	**0.004**	0.006	**0.000**	0.013	**0.000**	0.004
Average	Gap	0.070	**0.067**	0.070	**0.068**	**0.069**	0.070	0.069	**0.068**
	Violations	**0.065**	0.256	**0.050**	0.173	**0.027**	0.124	**0.007**	0.046

Figure SF4: Illustration of the optimality gap for self-supervised methods (on feasible examples only) for all approaches. Lower is better.

Table ST8: Average gap and constraint violations over test samples for models applied to the Segmentation12 Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±

q		10	30	60	90
ILP Obj		491.150	476.654	467.913	461.967
MAE	Gap	0.064 ± 0.051	0.061 ± 0.049	0.053 ± 0.042	0.052 ± 0.040
	Violations	0.569 ± 0.495	0.545 ± 0.498	0.430 ± 0.495	0.131 ± 0.337
MSE	Gap	0.054 ± 0.042	0.053 ± 0.042	0.051 ± 0.041	0.051 ± 0.041
	Violations	0.776 ± 0.417	0.580 ± 0.494	0.486 ± 0.500	0.186 ± 0.390
MAE+Penalty	Gap	0.064 ± 0.050	0.061 ± 0.049	0.059 ± 0.046	0.052 ± 0.043
	Violations	0.511 ± 0.500	0.500 ± 0.500	0.348 ± 0.476	0.140 ± 0.347
SL+Penalty	Gap	0.054 ± 0.042	0.052 ± 0.041	0.051 ± 0.040	0.051 ± 0.042
	Violations	0.651 ± 0.477	0.505 ± 0.500	0.486 ± 0.500	0.186 ± 0.390
$\texttt{SSL}_{pen}$	Gap	0.054 ± 0.043	0.052 ± 0.040	0.051 ± 0.041	0.051 ± 0.040
	Violations	0.790 ± 0.407	0.622 ± 0.485	0.486 ± 0.500	0.186 ± 0.390
PDL	Gap	0.063 ± 0.050	0.052 ± 0.041	0.052 ± 0.041	0.051 ± 0.041
	Violations	0.545 ± 0.498	0.622 ± 0.485	0.517 ± 0.500	0.163 ± 0.369
SS-CMPE	Gap	0.057 ± 0.044	0.051 ± 0.040	0.053 ± 0.043	0.050 ± 0.041
	Violations	0.503 ± 0.293	0.346 ± 0.485	0.257 ± 0.437	0.104 ± 0.305
SS-CMPE $_{pen}$	Gap	0.058 ± 0.043	0.052 ± 0.040	0.053 ± 0.043	0.051 ± 0.041
	Violations	0.152 ± 0.359	0.166 ± 0.372	0.084 ± 0.277	0.021 ± 0.143

Table ST9: Average gap and constraint violations over test samples for models applied to the Segmentation14 Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		493.647	482.837	476.145	470.485
MAE	Gap	0.067 ± 0.051	0.062 ± 0.048	0.062 ± 0.047	0.051 ± 0.040
	Violations	0.691 ± 0.462	0.623 ± 0.485	0.435 ± 0.496	0.271 ± 0.444
MSE	Gap	0.051 ± 0.039	0.049 ± 0.038	0.048 ± 0.037	0.053 ± 0.041
	Violations	0.810 ± 0.392	0.818 ± 0.386	0.606 ± 0.489	0.252 ± 0.434
MAE+Penalty	Gap	0.065 ± 0.051	0.061 ± 0.048	0.060 ± 0.046	0.054 ± 0.043
	Violations	0.693 ± 0.461	0.616 ± 0.487	0.410 ± 0.492	0.207 ± 0.405
SL+Penalty	Gap	0.049 ± 0.039	0.049 ± 0.039	0.049 ± 0.037	0.054 ± 0.041
	Violations	0.810 ± 0.392	0.803 ± 0.397	0.601 ± 0.490	0.215 ± 0.411
$\texttt{SSL}_{pen}$	Gap	0.061 ± 0.047	0.050 ± 0.038	0.050 ± 0.039	0.053 ± 0.042
	Violations	0.590 ± 0.492	0.618 ± 0.486	0.394 ± 0.489	0.207 ± 0.405
PDL	Gap	0.058 ± 0.045	0.068 ± 0.051	0.056 ± 0.043	0.054 ± 0.043
	Violations	0.507 ± 0.500	0.414 ± 0.493	0.403 ± 0.491	0.207 ± 0.405
SS-CMPE	Gap	0.050 ± 0.038	0.047 ± 0.037	0.048 ± 0.037	0.051 ± 0.040
	Violations	0.502 ± 0.295	0.444 ± 0.401	0.309 ± 0.497	0.150 ± 0.358
SS-CMPE $_{pen}$	Gap	0.050 ± 0.039	0.048 ± 0.038	0.050 ± 0.037	0.052 ± 0.042
	Violations	0.134 ± 0.340	0.088 ± 0.284	0.066 ± 0.248	0.046 ± 0.211

Table ST10: Average gap and constraint violations over test samples for models applied to the Segmentation15 Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		531.436	520.647	516.797	514.276
MAE	Gap	0.053 ± 0.043	0.054 ± 0.043	0.053 ± 0.042	0.052 ± 0.041
	Violations	0.570 ± 0.495	0.417 ± 0.493	0.248 ± 0.432	0.061 ± 0.239
MSE	Gap	0.051 ± 0.038	0.052 ± 0.041	0.052 ± 0.040	0.053 ± 0.041
	Violations	0.833 ± 0.373	0.715 ± 0.452	0.450 ± 0.498	0.076 ± 0.265
MAE+Penalty	Gap	0.056 ± 0.043	0.054 ± 0.042	0.053 ± 0.041	0.053 ± 0.042
	Violations	0.616 ± 0.486	0.457 ± 0.498	0.265 ± 0.441	0.075 ± 0.263
SL+Penalty	Gap	0.052 ± 0.040	0.051 ± 0.040	0.051 ± 0.040	0.052 ± 0.041
	Violations	0.833 ± 0.373	0.715 ± 0.452	0.450 ± 0.498	0.076 ± 0.265
$\texttt{SSL}_{pen}$	Gap	0.053 ± 0.042	0.059 ± 0.047	0.052 ± 0.041	0.054 ± 0.043
	Violations	0.676 ± 0.468	0.360 ± 0.480	0.274 ± 0.446	0.097 ± 0.295
PDL	Gap	0.051 ± 0.040	0.053 ± 0.043	0.051 ± 0.040	0.054 ± 0.043
	Violations	0.698 ± 0.459	0.461 ± 0.499	0.392 ± 0.488	0.086 ± 0.280
SS-CMPE	Gap	0.051 ± 0.040	0.050 ± 0.041	0.052 ± 0.040	0.051 ± 0.040
	Violations	0.366 ± 0.474	0.298 ± 0.499	0.225 ± 0.417	0.059 ± 0.236
SS-CMPE $_{pen}$	Gap	0.052 ± 0.040	0.052 ± 0.042	0.052 ± 0.041	0.051 ± 0.041
	Violations	0.060 ± 0.238	0.060 ± 0.238	0.049 ± 0.215	0.001 ± 0.032

Table ST11: Average gap and constraint violations over test samples for models applied to the Grids17 Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		2892.585191	2884.506703	2877.311872	2878.147272
MAE	Gap	0.119 ± 0.078	0.133 ± 0.088	0.125 ± 0.084	0.114 ± 0.078
	Violations	0.565 ± 0.496	0.376 ± 0.485	0.122 ± 0.328	0.020 ± 0.140
MSE	Gap	0.054 ± 0.041	0.057 ± 0.045	0.067 ± 0.054	0.075 ± 0.056
	Violations	0.314 ± 0.464	0.152 ± 0.359	0.044 ± 0.205	0.013 ± 0.111
MAE+Penalty	Gap	0.129 ± 0.081	0.144 ± 0.089	0.137 ± 0.087	0.127 ± 0.085
	Violations	0.534 ± 0.499	0.376 ± 0.485	0.142 ± 0.350	0.019 ± 0.137
SL+Penalty	Gap	0.054 ± 0.041	0.059 ± 0.045	0.069 ± 0.054	0.058 ± 0.044
	Violations	0.304 ± 0.460	0.125 ± 0.331	0.044 ± 0.205	0.002 ± 0.045
$\texttt{SSL}_{pen}$	Gap	0.104 ± 0.060	0.079 ± 0.056	0.093 ± 0.059	0.087 ± 0.058
	Violations	0.149 ± 0.357	0.013 ± 0.113	0.004 ± 0.067	0.026 ± 0.159
PDL	Gap	0.086 ± 0.056	0.087 ± 0.058	0.109 ± 0.063	0.112 ± 0.062
	Violations	0.043 ± 0.202	0.003 ± 0.055	0.001 ± 0.022	0.014 ± 0.118
SS-CMPE	Gap	0.072 ± 0.051	0.071 ± 0.052	0.066 ± 0.048	0.063 ± 0.049
	Violations	0.146 ± 0.353	0.032 ± 0.176	0.024 ± 0.152	0.001 ± 0.022
SS-CMPE $_{pen}$	Gap	0.073 ± 0.052	0.069 ± 0.051	0.073 ± 0.052	0.066 ± 0.050
	Violations	0.035 ± 0.185	0.013 ± 0.113	0.002 ± 0.039	0.001 ± 0.022

Table ST12: Average gap and constraint violations over test samples for models applied to the Grids18 Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		4185.600003	4166.310635	4158.737261	4167.11386
MAE	Gap	0.138 ± 0.087	0.142 ± 0.091	0.122 ± 0.081	0.102 ± 0.073
	Violations	0.606 ± 0.489	0.389 ± 0.488	0.178 ± 0.383	0.019 ± 0.138
MSE	Gap	0.056 ± 0.044	0.060 ± 0.047	0.069 ± 0.056	0.064 ± 0.050
	Violations	0.332 ± 0.471	0.194 ± 0.395	0.178 ± 0.383	0.015 ± 0.122
MAE+Penalty	Gap	0.143 ± 0.087	0.154 ± 0.093	0.145 ± 0.092	0.114 ± 0.080
	Violations	0.551 ± 0.498	0.364 ± 0.481	0.172 ± 0.378	0.021 ± 0.145
SL+Penalty	Gap	0.061 ± 0.047	0.064 ± 0.049	0.065 ± 0.050	0.133 ± 0.082
	Violations	0.210 ± 0.407	0.087 ± 0.282	0.025 ± 0.158	0.025 ± 0.158
$\texttt{SSL}_{pen}$	Gap	0.115 ± 0.065	0.074 ± 0.055	0.093 ± 0.060	0.123 ± 0.065
	Violations	0.060 ± 0.238	0.182 ± 0.386	0.013 ± 0.115	0.005 ± 0.074
PDL	Gap	0.105 ± 0.063	0.126 ± 0.067	0.101 ± 0.062	0.118 ± 0.064
	Violations	0.097 ± 0.295	0.001 ± 0.032	0.003 ± 0.050	0.005 ± 0.071
SS-CMPE	Gap	0.064 ± 0.047	0.072 ± 0.053	0.060 ± 0.046	0.065 ± 0.049
	Violations	0.200 ± 0.400	0.029 ± 0.169	0.000 ± 0.000	0.001 ± 0.032
SS-CMPE $_{pen}$	Gap	0.067 ± 0.051	0.067 ± 0.051	0.078 ± 0.055	0.075 ± 0.053
	Violations	0.017 ± 0.129	0.019 ± 0.137	0.002 ± 0.039	0.000 ± 0.000

Table ST13: Average gap and constraint violations over test samples for models applied to the AD Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		2535.424	2526.023	2521.957	2519.917
MAE	Gap	0.288 ± 0.061	0.276 ± 0.063	0.283 ± 0.061	0.270 ± 0.063
	Violations	0.671 ± 0.470	0.457 ± 0.498	0.257 ± 0.437	0.046 ± 0.210
MSE	Gap	0.204 ± 0.061	0.201 ± 0.063	0.204 ± 0.061	0.213 ± 0.059
	Violations	0.336 ± 0.472	0.063 ± 0.243	0.069 ± 0.254	0.013 ± 0.111
MAE +Penalty	Gap	0.294 ± 0.062	0.290 ± 0.066	0.277 ± 0.061	0.274 ± 0.061
	Violations	0.600 ± 0.490	0.468 ± 0.499	0.271 ± 0.444	0.039 ± 0.194
SL+Penalty	Gap	0.216 ± 0.061	0.213 ± 0.063	0.220 ± 0.061	0.229 ± 0.060
	Violations	0.085 ± 0.278	0.041 ± 0.197	0.021 ± 0.142	0.005 ± 0.074
$\texttt{SSL}_{pen}$	Gap	0.135 ± 0.055	0.140 ± 0.057	0.142 ± 0.054	0.134 ± 0.055
	Violations	0.244 ± 0.430	0.143 ± 0.350	0.054 ± 0.226	0.005 ± 0.074
PDL	Gap	0.148 ± 0.056	0.152 ± 0.056	0.146 ± 0.055	0.139 ± 0.054
	Violations	0.055 ± 0.228	0.006 ± 0.080	0.013 ± 0.113	0.004 ± 0.063
SS-CMPE	Gap	0.135 ± 0.055	0.131 ± 0.057	0.131 ± 0.055	0.131 ± 0.054
	Violations	0.102 ± 0.302	0.025 ± 0.155	0.005 ± 0.071	0.003 ± 0.055
SS-CMPE $_{pen}$	Gap	0.129 ± 0.054	0.136 ± 0.057	0.130 ± 0.054	0.133 ± 0.054
	Violations	0.017 ± 0.127	0.004 ± 0.063	0.000 ± 0.000	0.000 ± 0.000

Table ST14: Average gap and constraint violations over test samples for models applied to the BBC Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		890.289	880.270	875.668	872.118
MAE	Gap	0.053 ± 0.037	0.046 ± 0.034	0.043 ± 0.032	0.045 ± 0.034
	Violations	0.779 ± 0.415	0.624 ± 0.485	0.414 ± 0.493	0.165 ± 0.371
MSE	Gap	0.036 ± 0.027	0.036 ± 0.028	0.037 ± 0.028	0.037 ± 0.029
	Violations	0.924 ± 0.265	0.854 ± 0.354	0.578 ± 0.494	0.204 ± 0.403
MAE+Penalty	Gap	0.047 ± 0.036	0.044 ± 0.034	0.041 ± 0.031	0.040 ± 0.031
	Violations	0.657 ± 0.475	0.557 ± 0.497	0.384 ± 0.486	0.151 ± 0.358
SL+Penalty	Gap	0.036 ± 0.028	0.036 ± 0.028	0.037 ± 0.030	0.037 ± 0.029
	Violations	0.919 ± 0.272	0.854 ± 0.354	0.578 ± 0.494	0.204 ± 0.403
$\texttt{SSL}_{pen}$	Gap	0.041 ± 0.032	0.042 ± 0.032	0.038 ± 0.030	0.039 ± 0.030
	Violations	0.516 ± 0.500	0.393 ± 0.488	0.408 ± 0.492	0.130 ± 0.336
PDL	Gap	0.043 ± 0.033	0.051 ± 0.036	0.045 ± 0.034	0.044 ± 0.033
	Violations	0.336 ± 0.472	0.160 ± 0.366	0.149 ± 0.356	0.029 ± 0.169
SS-CMPE	Gap	0.038 ± 0.029	0.043 ± 0.032	0.042 ± 0.033	0.040 ± 0.031
	Violations	0.316 ± 0.495	0.239 ± 0.427	0.108 ± 0.310	0.044 ± 0.206
SS-CMPE $_{pen}$	Gap	0.038 ± 0.030	0.043 ± 0.032	0.044 ± 0.033	0.040 ± 0.031
	Violations	0.074 ± 0.263	0.067 ± 0.250	0.056 ± 0.229	0.002 ± 0.045

Table ST15: Average gap and constraint violations over test samples for models applied to the 20 Newsgroup Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		928.386	924.439	923.173	921.754
MAE	Gap	0.044 ± 0.034	0.046 ± 0.036	0.047 ± 0.035	0.048 ± 0.037
	Violations	0.470 ± 0.499	0.176 ± 0.381	0.049 ± 0.215	0.001 ± 0.022
MSE	Gap	0.050 ± 0.038	0.053 ± 0.039	0.051 ± 0.038	0.051 ± 0.037
	Violations	0.639 ± 0.480	0.403 ± 0.491	0.142 ± 0.349	0.008 ± 0.089
MAE+Penalty	Gap	0.044 ± 0.035	0.047 ± 0.036	0.047 ± 0.036	0.047 ± 0.035
	Violations	0.455 ± 0.498	0.181 ± 0.386	0.046 ± 0.210	0.001 ± 0.022
SL+Penalty	Gap	0.046 ± 0.036	0.047 ± 0.035	0.046 ± 0.036	0.044 ± 0.034
	Violations	0.573 ± 0.495	0.384 ± 0.486	0.161 ± 0.367	0.015 ± 0.122
$\texttt{SSL}_{pen}$	Gap	0.045 ± 0.036	0.046 ± 0.036	0.045 ± 0.035	0.046 ± 0.035
	Violations	0.386 ± 0.487	0.139 ± 0.346	0.024 ± 0.152	0.002 ± 0.039
PDL	Gap	0.043 ± 0.035	0.046 ± 0.036	0.046 ± 0.036	0.046 ± 0.036
	Violations	0.278 ± 0.448	0.129 ± 0.335	0.028 ± 0.165	0.001 ± 0.032
SS-CMPE	Gap	0.043 ± 0.034	0.045 ± 0.035	0.044 ± 0.034	0.044 ± 0.034
	Violations	0.317 ± 0.465	0.086 ± 0.280	0.019 ± 0.137	0.001 ± 0.032
SS-CMPE $_{pen}$	Gap	0.044 ± 0.033	0.045 ± 0.035	0.046 ± 0.035	0.045 ± 0.034
	Violations	0.069 ± 0.254	0.054 ± 0.227	0.007 ± 0.083	0.001 ± 0.022

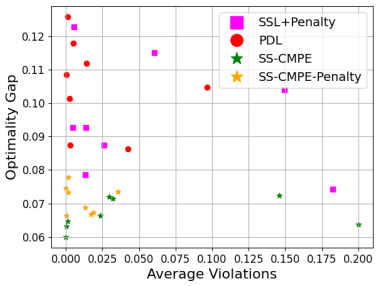
Table ST16: Average gap and constraint violations over test samples for models applied to the Webkb Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		828.463	824.361	825.517	823.917
MAE	Gap	0.057 ± 0.043	0.057 ± 0.043	0.058 ± 0.043	0.062 ± 0.044
	Violations	0.613 ± 0.487	0.502 ± 0.500	0.249 ± 0.433	0.046 ± 0.210
MSE	Gap	0.065 ± 0.046	0.069 ± 0.046	0.066 ± 0.045	0.065 ± 0.045
	Violations	0.695 ± 0.461	0.473 ± 0.499	0.210 ± 0.407	0.018 ± 0.133
MAE+Penalty	Gap	0.058 ± 0.042	0.058 ± 0.042	0.059 ± 0.043	0.061 ± 0.044
	Violations	0.471 ± 0.499	0.395 ± 0.489	0.211 ± 0.408	0.041 ± 0.197
SL+Penalty	Gap	0.054 ± 0.042	0.054 ± 0.041	0.054 ± 0.040	0.054 ± 0.040
	Violations	0.584 ± 0.493	0.378 ± 0.485	0.174 ± 0.380	0.018 ± 0.131
$\texttt{SSL}_{pen}$	Gap	0.058 ± 0.043	0.057 ± 0.041	0.057 ± 0.042	0.057 ± 0.042
	Violations	0.360 ± 0.480	0.217 ± 0.413	0.082 ± 0.274	0.015 ± 0.120
PDL	Gap	0.063 ± 0.045	0.060 ± 0.044	0.058 ± 0.042	0.057 ± 0.042
	Violations	0.149 ± 0.357	0.096 ± 0.295	0.056 ± 0.229	0.013 ± 0.115
SS-CMPE	Gap	0.059 ± 0.043	0.058 ± 0.043	0.056 ± 0.042	0.056 ± 0.042
	Violations	0.169 ± 0.374	0.050 ± 0.218	0.026 ± 0.159	0.004 ± 0.063
SS-CMPE $_{pen}$	Gap	0.062 ± 0.045	0.061 ± 0.044	0.057 ± 0.042	0.053 ± 0.040
	Violations	0.074 ± 0.263	0.029 ± 0.169	0.001 ± 0.032	0.001 ± 0.022

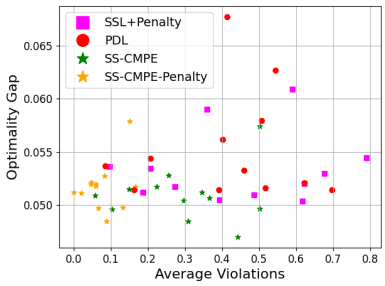
Table ST17: Average gap and constraint violations over test samples for models applied to the DNA Dataset for different q values. The plot displays the mean values of the average gap and constraint violations, with standard deviations denoted by ±.

q		10	30	60	90
ILP Obj		222.848	221.635	221.114	220.625
MAE	Gap	0.138 ± 0.109	0.142 ± 0.109	0.136 ± 0.109	0.141 ± 0.111
	Violations	0.444 ± 0.497	0.448 ± 0.497	0.286 ± 0.452	0.114 ± 0.317
MSE	Gap	0.138 ± 0.112	0.140 ± 0.112	0.139 ± 0.111	0.139 ± 0.110
	Violations	0.506 ± 0.500	0.565 ± 0.496	0.322 ± 0.467	0.113 ± 0.317
MAE+Penalty	Gap	0.140 ± 0.111	0.136 ± 0.106	0.143 ± 0.111	0.137 ± 0.113
	Violations	0.444 ± 0.497	0.448 ± 0.497	0.286 ± 0.452	0.114 ± 0.317
SL+Penalty	Gap	0.135 ± 0.109	0.140 ± 0.112	0.141 ± 0.111	0.143 ± 0.115
	Violations	0.434 ± 0.496	0.494 ± 0.500	0.281 ± 0.450	0.089 ± 0.285
$\texttt{SSL}_{pen}$	Gap	0.140 ± 0.115	0.141 ± 0.111	0.146 ± 0.116	0.143 ± 0.118
	Violations	0.048 ± 0.214	0.062 ± 0.241	0.014 ± 0.118	0.004 ± 0.067
PDL	Gap	0.140 ± 0.113	0.141 ± 0.113	0.139 ± 0.112	0.144 ± 0.120
	Violations	0.287 ± 0.452	0.129 ± 0.335	0.003 ± 0.055	0.006 ± 0.077
SS-CMPE	Gap	0.138 ± 0.113	0.138 ± 0.108	0.137 ± 0.106	0.139 ± 0.109
	Violations	0.046 ± 0.210	0.017 ± 0.129	0.012 ± 0.109	0.008 ± 0.089
SS-CMPE $_{pen}$	Gap	0.139 ± 0.116	0.139 ± 0.113	0.140 ± 0.112	0.139 ± 0.113
	Violations	0.013 ± 0.115	0.002 ± 0.045	0.001 ± 0.022	0.001 ± 0.022

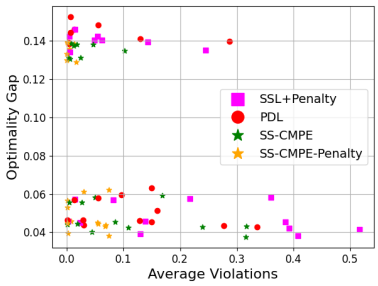
D.B Optimality Gap And Violations in Self-Supervised Methods for Different q Values



(a) Optimality Gap (avg %) and Average Violations for Grids UAI networks



(b) Opt. Gap (avg %) and Avg. Violations for Segmentation UAI networks



(c) Optimality Gap (avg %) and Average Violations for Tractable Models

Figure SF5: Visualization of Optimality Gap (average %) and Average Violations for Self-Supervised Methods across different q values. Points closer to the origin indicate better performance.

In the scatter plots depicted in Figure SF5, three distinct evaluations of the optimality gap against the average violations for various self-supervised methods across different q values are visualized. Points positioned closer to the origin indicate better performance, with reduced optimality gaps and fewer violations. In Figure SF5(a), focused on Grids UAI networks, the $\texttt{SS-CMPE}_{pen}$ method generally occupies a position near the origin, indicating its commendable performance in this setting. The SS-CMPE method exhibits a comparable performance to the $\texttt{SS-CMPE}_{pen}$ method, with occasional high levels of violations observed in two instances.

In Figure SF5(b), showcasing the Segmentation UAI networks, the SS-CMPE and $\texttt{SS-CMPE}_{pen}$ methods again demonstrate superiority, particularly evident by their prevalence near the origin. Finally, in Figure SF5(c) related to tractable models, the $\texttt{SS-CMPE}_{pen}$ method often achieves optimal placement close to the origin, reflecting a balanced performance. These evaluations provide critical insights into the effectiveness and robustness of the proposed self-supervised methods across different problems.

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.*

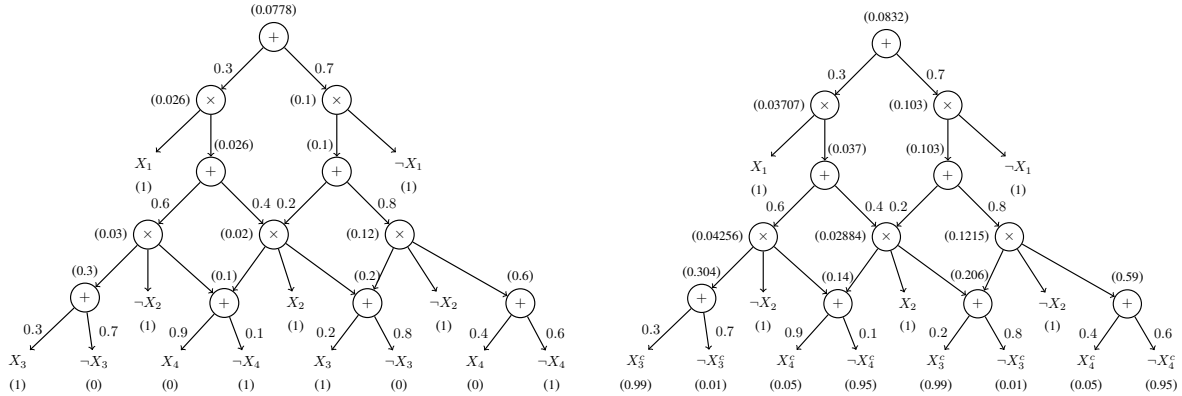
Example 1. *Figure 1(a) shows a smooth and decomposable probabilistic circuit defined over $\mathbf{X} = \{X_1, \ldots, X_4\}$.*

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).

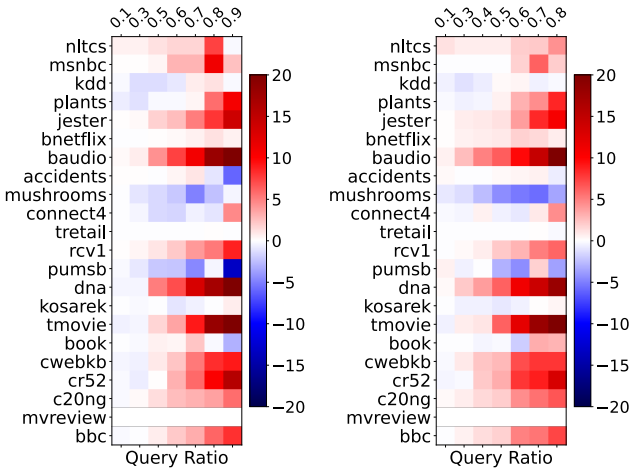
Because our learning method does not depend on the specific choice of neural network architectures, we use a fixed neural network architecture across all experiments: fully connected with four hidden layers having 128, 256, 512, and 1024 nodes respectively. We used ReLU activation in the hidden layers, 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.