NEURAL SOLVERS FOR FAST, ACCURATE PROBABILISTIC INFERENCE

by

Shivvrat Arya

APPROVED BY SUPERVISORY COMMITTEE:

____

Vibhav Gogate, Chair

____

Yu Xiang, Co-Chair

____

Nicholas Ruozzi

____

Sriraam Natarajan

To my parents, Vinay Arya and Smita Arya,

and my partner, Smita Ghosh,

for their unwavering support and encouragement.

NEURAL SOLVERS FOR FAST, ACCURATE PROBABILISTIC INFERENCE

by

SHIVVRAT ARYA

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2025

ACKNOWLEDGMENTS

First and foremost, I extend my deepest gratitude to my advisor, Dr. Vibhav Gogate, whose invaluable guidance and mentorship has been instrumental in shaping this dissertation. His unwavering support, keen insight, and relentless pursuit of excellence has profoundly influenced my research journey. From the moment I joined his lab at The University of Texas at Dallas, he welcomed me with generosity, providing both intellectual and financial support through the DARPA XAI grant (award number W911NF-08-1-0242). His ability to tackle seemingly unsolvable research problems with innovative approaches has been a constant source of inspiration. I am immensely grateful for his patience, encouragement, and belief in my abilities, which has made my PhD experience both rewarding and fulfilling.

I also express my sincere appreciation to my co-advisor, Dr. Yu Xiang, whose insightful feedback, stimulating discussions, and collaborative efforts have played a significant role in my research. His expertise and constructive critiques have been invaluable, greatly enhancing the quality of this dissertation. Additionally, I extend my sincere gratitude to my committee members, Dr. Sriraam Natarajan and Dr. Nicholas Ruozzi, for their valuable guidance, critical feedback, and thought-provoking discussions, all of which have been instrumental in refining my work.

Beyond my advisors and committee members, I am especially grateful to Dr. Tahrima Rahman for her support in my research and for our collaborations on multiple publications. Her insight and contributions have played a pivotal role in my academic growth. I also extend my gratitude to Chiradeep, Vasundhara, Brij, and Akshay for their friendship and collaboration. Working alongside them has been both intellectually enriching and personally rewarding.

My deepest gratitude goes to my parents, Vinay and Smita, whose belief in me and constant encouragement has been my greatest source of strength. They have instilled in me the values of perseverance, ambition, and resilience, shaping me into the person I am today. Their sacrifices and unconditional support have made this journey possible. I am equally grateful to my partner, Smita Ghosh, whose care, support, patience, and encouragement has been my anchor throughout the challenges of my PhD. Her presence has been a constant source of strength and motivation. This achievement is as much a testament to their support as it is to my efforts.

Finally, I acknowledge the financial support provided by various funding agencies, including the DARPA Explainable Artificial Intelligence (XAI) Program (award number N66001-17-2-4032), the DARPA Perceptually-Enabled Task Guidance (PTG) Program (award number HR0011222-0005), the DARPA Assured Neuro Symbolic Learning and Reasoning (ANSR) Program (award number HR001122S0039), the National Science Foundation grant (award number IIS-1652835), and the AFOSR award (award number FA9550-23-1-0239).

These grants have provided me with opportunities to collaborate on exciting projects with research groups led by Dr. Yu Xiang, Dr. Nicholas Ruozzi, and Dr. Sriraam Natarajan at UTD, Dr. Eric Ragan at the University of Florida, and Dr. Guy Van den Broeck at UCLA. I am especially grateful to Dr. Vibhav Gogate, Dr. Yu Xiang and Dr. Nicholas Ruozzi for their unwavering support and encouragement during the intensive months leading up to the DARPA PTG Demo Days. The challenges we tackled, the discussions we engaged in, and the rigorous work during that time have left a lasting impact on me, shaping me into a better researcher. I also appreciate the support from the Department of Computer Science at UTD for funding my early years of graduate studies.

This dissertation is the culmination of years of hard work, challenges, and learning, and it would not have been possible without the contributions and support of these exceptional individuals and institutions. I am deeply thankful to all who have been part of this journey.

March 2025

NEURAL SOLVERS FOR FAST, ACCURATE PROBABILISTIC INFERENCE

Shivvrat Arya, PhD
The University of Texas at Dallas, 2025

Supervising Professors: Vibhav Gogate, Chair
Yu Xiang, Co-Chair

Probabilistic Models (PMs) provide a powerful framework for representing large multivariate probability distributions, but as their expressiveness increases, inference tasks become computationally intractable. In the era of big data, the demand for complex models capable of effectively representing and reasoning over large-scale data has increased significantly. While models like probabilistic circuits allow tractable inference for specific query types, handling a broader range of queries remains challenging. Exact inference in PMs is generally NP-hard, and approximate methods, though viable, often involve trade-offs between accuracy and efficiency, leading to unreliable estimates. Achieving near-optimal solutions typically demands substantial computational resources, posing a barrier to scalability.

This dissertation presents novel algorithms for key inference tasks in PMs, including Most Probable Explanation (MPE), Constrained Most Probable Explanation (CMPE), and Marginal Maximum A Posteriori (MMAP) inference. Our methods generalize across different Probabilistic Models (PMs), such as Probabilistic Graphical Models (PGMs), Probabilistic Circuits (PCs), and Neural Autoregressive Models (NAMs), enabling efficient and scalable probabilistic reasoning. Additionally, we propose a neurosymbolic framework that integrates Dependency Networks (DNs) with deep learning architectures to enhance multi-label classification. Specifically, this dissertation makes the following contributions:

- We develop a self-supervised learning framework for neural network-based solvers to answer predefined MPE and MMAP queries over PCs. Our approach introduces a scalable loss function with a computation cost that scales linearly with the size of the PC, enabling the training of neural solvers that match or surpass state-of-the-art solvers in solution quality while reducing inference time from seconds to microseconds.

- We extend inference capabilities beyond predefined queries, allowing neural solvers to answer *arbitrary* MPE queries over PCs, PGMs, and NAMs. To achieve this, we introduce a novel dual-network learning framework and an enhanced inference scheme that updates neural network parameters at test time, leading to improved solution quality.

- We eliminate the need for costly inference-time optimization in neural solvers for *arbitrary* MPE queries over PGMs by enhancing query encoding, which improves solution quality through richer embeddings. Additionally, we introduce two novel methods for discretizing continuous neural network outputs, further enhancing solution quality.

- We extend neural solvers to constrained inference, specifically the Constrained Most Probable Explanation (CMPE) task. We develop a self-supervised CMPE solver with a loss function satisfying the *consistent loss property*, ensuring alignment with the optimal CMPE solution—unlike existing loss functions for constrained optimization, which lack this guarantee.

- We propose advanced inference schemes for MPE in neurosymbolic models for multi-label classification, specifically improving inference in Deep Dependency Networks (DDNs). While DDNs offer efficient training and an intuitive loss function for multi-label classification, they traditionally rely on Gibbs sampling, which limits inference accuracy and efficiency. To address this limitation, we introduce novel inference techniques based on local search and integer linear programming (ILP), facilitating more accurate and efficient computation of the most probable label assignments.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Deep learning systems have achieved state-of-the-art performance across diverse domains, including vision, speech, language understanding, medical diagnosis, and predictive modeling. However, these models remain error-prone, particularly when faced with out-of-distribution inputs or subtle contextual shifts, raising concerns about their robustness and reliability. In vision, deep models excel at object and scene identification, but often misinterpret physical interactions, causal relationships, or intent. For instance, Figure 1.1 illustrates a captioning model by Karpathy and Li (2015) that correctly identifies objects but fails to capture contextual meaning, leading to incorrect captions (Lake et al., 2017). Similarly, Figure 1.2 shows how occlusions cause the model to misclassify monkeys as humans or a guitar as a bird, highlighting its sensitivity to minor contextual variations (Wang et al., 2018).

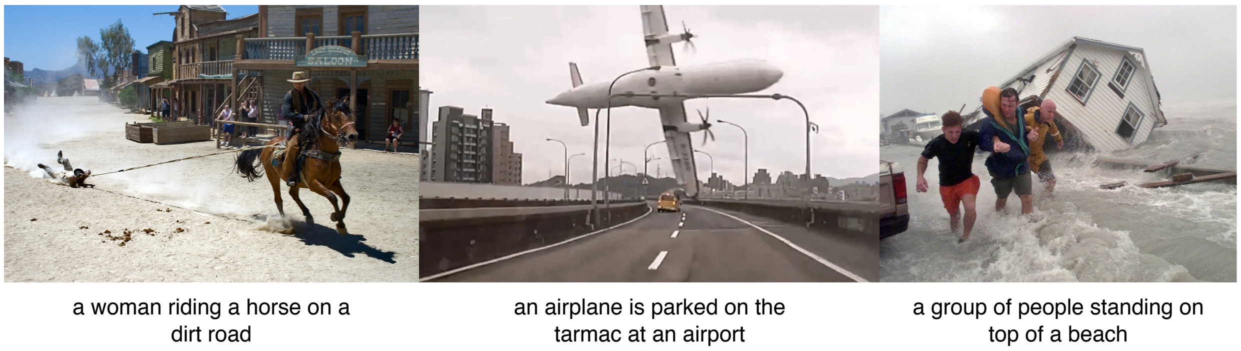

a woman riding a horse on a dirt road
an airplane is parked on the tarmac at an airport
a group of people standing on top of a beach

Figure 1.1: Examples of image captions generated by a deep neural network (Karpathy and Li, 2015), as presented by Lake et al. (2017). Image sources: Gabriel Villena Fernández (left), TVBS Taiwan/Agence France-Presse (center), and AP Photo/Dave Martin (right).

This brittleness extends to distribution shifts in training data. Models trained on ImageNet (Deng et al., 2009) experience accuracy drops of up to 45% (Figure 1.3b) when evaluated on ObjectNet (Barbu et al., 2019), which introduces realistic variations in object poses, backgrounds, and viewpoints (Figure 1.3a). This performance degradation underscores deep models' dependence on training distributions, posing a significant challenge for deployment in safety-critical

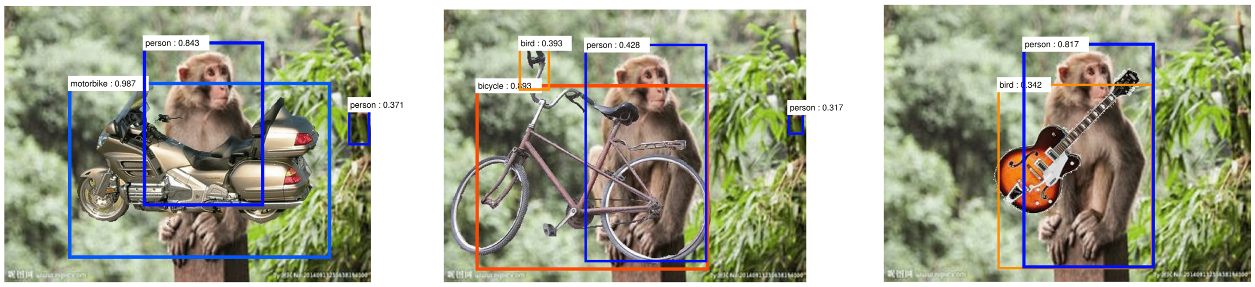
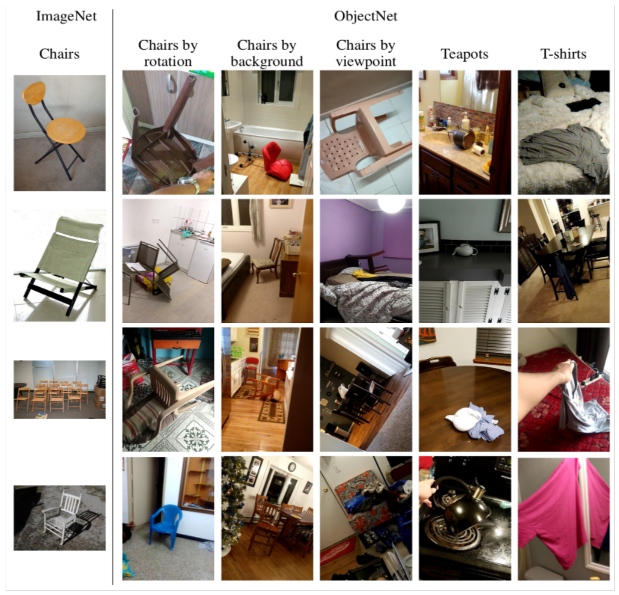
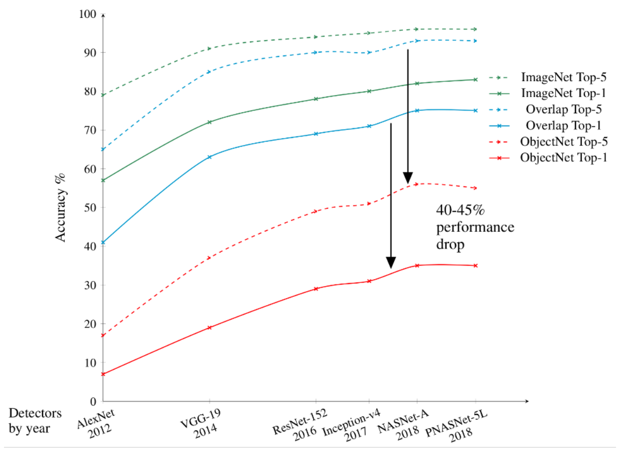


Figure 1.2: Occluders disrupt deep networks, creating adversarial context examples (Wang et al., 2018). Left: The motorbike causes the monkey to be misclassified as a human. Center: The bicycle leads to a similar misclassification, with the jungle turning the handle into a tropical bird. Right: The guitar causes misclassification, while the jungle alters it into a bird.



(a) ImageNet (Deng et al., 2009) (left) shows objects in typical settings with limited rotations and viewpoints. ObjectNet (Barbu et al., 2019) (right) introduces significant variability in rotation, background, and viewpoint, with most detectors failing on its examples.



(b) Performance on ObjectNet (Barbu et al., 2019) test dataset for state of the art image classifiers trained on ImageNet, including AlexNet (Krizhevsky et al., 2012), VGG-19 (Simonyan and Zisserman, 2015), ResNet-152 (He et al., 2016), Inception-v4 (Szegedy et al., 2017), NASNET-A (Zoph et al., 2018), and PNASNet-5 Large (Liu et al., 2017).

Figure 1.3: Comparison between ImageNet and ObjectNet datasets (a) and performance of leading ImageNet-trained classifiers on ObjectNet (b).

domains such as autonomous systems and medical diagnosis, where modeling distribution shifts is inherently difficult.

GSM-NoOp

Oliver picks 44 kiwis on Friday. Then he picks 58 kiwis on Saturday. On Sunday, he picks double the number of kiwis he did on Friday, but five of them were a bit smaller than average. How many kiwis does Oliver have?

---

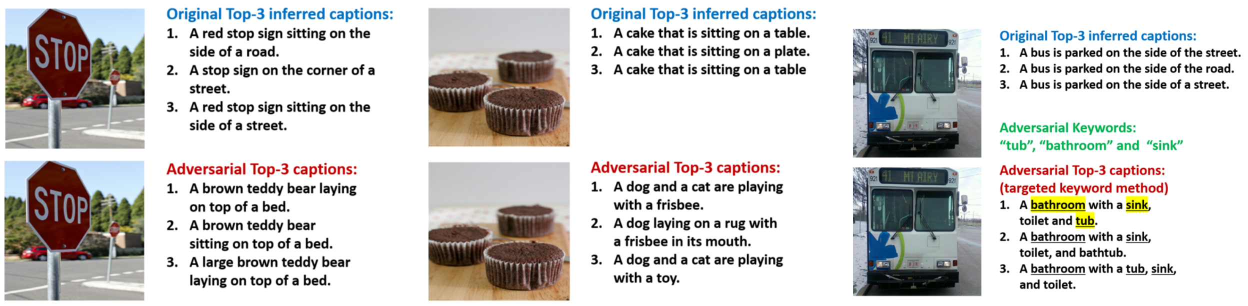
o1-mini: Oliver picks kiwis over three days as follows:
- Friday: 44 kiwis
- Saturday: 58 kiwis
- Sunday: Double the number he picked on Friday, which is $2 \times 44 = 88$ kiwis

However, on Sunday, 5 of these kiwis were smaller than average. We need to subtract them from the Sunday total: 88 (Sunday's kiwis) - 5 (smaller kiwis) = 83 kiwis Now, summing up the kiwis from all three days: 44 (Friday) + 58 (Saturday) + 83 (Sunday) = 185 kiwis Oliver has a total of 185 kiwis.

Llama3-8B: Let's break it down step by step:
Oliver picks 44 kiwis on Friday. He picks 58 kiwis on Saturday. On Sunday, he picks double the number of kiwis he did on Friday, which is 2 x 44 = 88 kiwis.
Five of the kiwis picked on Sunday are a bit smaller than average, so we subtract 5 from the total number of kiwis picked on Sunday: 88 - 5 = 83 kiwis. Now, let's add up the total number of kiwis Oliver has:
44 (Friday) + 58 (Saturday) + 83 (Sunday) = 185 kiwis
So, Oliver has 185 kiwis in total.

Figure 1.4: An example from the GSM-NoOp (Mirzadeh et al., 2025) dataset: Irrelevant yet seemingly relevant statements are added to questions. Most models fail to disregard these distractions, erroneously converting them into operations, resulting in mistakes.

Furthermore, deep learning models lack logical reasoning capabilities. Figure 1.4 illustrates how Large Language Models, such as o1-mini and LLaMA3-8B (Grattafiori et al., 2024), fail to disregard irrelevant distractors in arithmetic problems from the GSM-NoOp dataset (Mirzadeh et al., 2025), exposing fundamental reasoning deficits. Compounding these limitations, deep models are highly susceptible to adversarial attacks. As shown in Figure 1.5, imperceptible perturbations lead captioning models (Vinyals et al., 2015) to generate nonsensical outputs (Chen et al., 2018). Similarly, Figure 1.6 demonstrates how CLIP (Radford et al., 2021) can be deceived by simple typographic attacks (Goh et al., 2021), misclassifying a Granny Smith apple as an iPod with high confidence.



Original Top-3 inferred captions:
1. A red stop sign sitting on the side of a road.
2. A stop sign on the corner of a street.
3. A red stop sign sitting on the side of a street.

Adversarial Top-3 captions:
1. A brown teddy bear laying on top of a bed.
2. A brown teddy bear sitting on top of a bed.
3. A large brown teddy bear laying on top of a bed.

Original Top-3 inferred captions:
1. A cake that is sitting on a table.
2. A cake that is sitting on a plate.
3. A cake that is sitting on a table

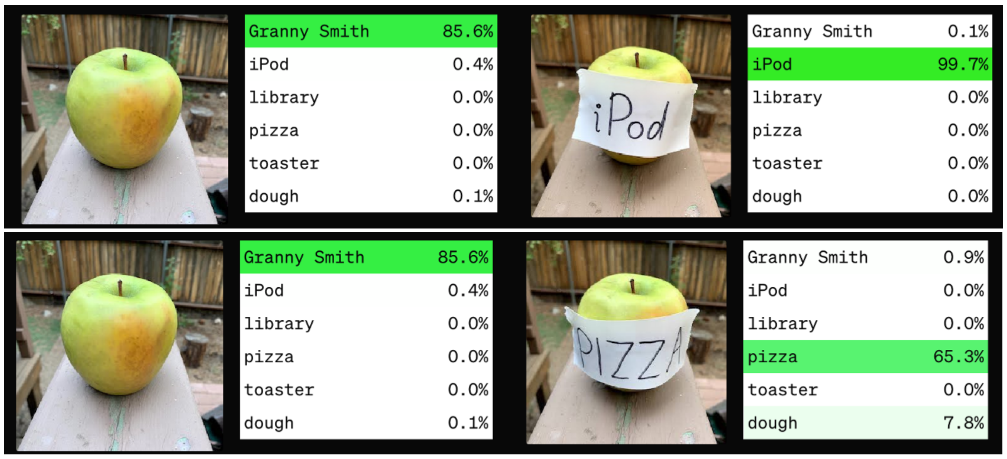
Adversarial Top-3 captions:
1. A dog and a cat are playing with a frisbee.
2. A dog laying on a rug with a frisbee in its mouth.
3. A dog and a cat are playing with a toy.

Original Top-3 inferred captions:
1. A bus is parked on the side of the street.
2. A bus is parked on the side of the road.
3. A bus is parked on the side of a street.

Adversarial Keywords: "tub", "bathroom" and "sink"

Adversarial Top-3 captions: (targeted keyword method)
1. A bathroom with a sink, toilet and tub.
2. A bathroom with a sink, toilet, and bathtub.
3. A bathroom with a tub, sink, and toilet.

Figure 1.5: Adversarial examples generated by Show-and-Fool (Chen et al., 2018) using targeted caption and keyword methods. The target model is Show-and-Tell (Vinyals et al., 2015), with original images from the MS-COCO validation set.



Granny Smith	85.6%
iPod	0.4%
library	0.0%
pizza	0.0%
toaster	0.0%
dough	0.1%

Granny Smith	0.1%
iPod	99.7%
library	0.0%
pizza	0.0%
toaster	0.0%
dough	0.0%

Granny Smith	85.6%
iPod	0.4%
library	0.0%
pizza	0.0%
toaster	0.0%
dough	0.1%

Granny Smith	0.9%
iPod	0.0%
library	0.0%
pizza	65.3%
toaster	0.0%
dough	7.8%

Figure 1.6: Typographical attack (Goh et al., 2021) on CLIP (Radford et al., 2021): Exploiting the model's robust text recognition, even hand-written text in photographs can often deceive the model.

These findings highlight three fundamental limitations of current deep learning models: susceptibility to out-of-distribution data, sensitivity to small perturbations, and deficiencies in robust reasoning. These weaknesses significantly constrain their applicability in domains that demand reliability, interpretability, and adaptability.

In contrast, symbolic AI explicitly encodes logical relationships and properties, providing an interpretable framework compared to deep learning models, which rely on implicitly learned rep-

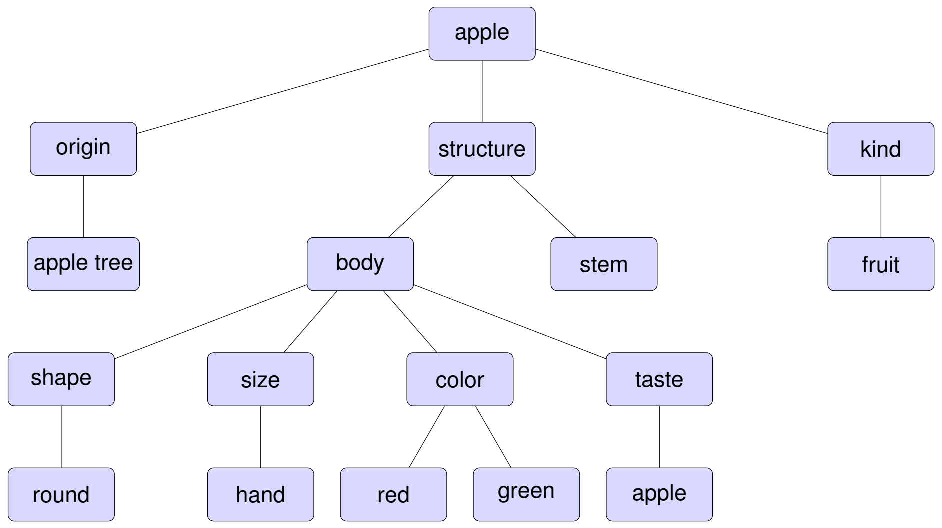


Figure 1.7: Knowledge graph from Minsky (1991) illustrating how details about an apple are decomposed into intuitive and logical units.

resentations. As shown in Figure 1.7, symbolic knowledge graphs decompose concepts, such as an apple, into structured logical units, enabling systematic and generalizable reasoning (Minsky, 1991). Symbolic systems ensure deterministic, logically consistent outputs and operate efficiently with limited data, making them well-suited for tasks demanding explainability and formal reasoning. However, they struggle with unstructured inputs such as images, text, and speech—domains where deep learning excels at extracting patterns directly from raw data.

Neurosymbolic AI addresses these complementary limitations by integrating deep learning's perceptual capabilities with symbolic AI's reasoning strengths. Neural networks process unstructured data, while symbolic components perform high-level reasoning using structured representations. This hybrid approach offers a promising path toward robust, interpretable, and generalizable AI systems, particularly in high-stakes domains where both accurate perception and logical reasoning are essential.

This dissertation investigates the use of Probabilistic Models (PMs) as the symbolic component of neurosymbolic AI, referring to these models as neurosymbolic probabilistic models. Probabilistic Models (PMs) provide a compact representation of large, complex multivariate probability distributions and facilitate efficient query answering. PMs offer a principled framework for encoding random variables and their conditional independencies, enabling structured reasoning under uncertainty.

1.1 Contributions

Although Probabilistic Models (PMs) enable probabilistic reasoning, their computational complexity scales with expressiveness, posing significant challenges for efficient inference. Exact inference in PMs is NP-hard (Cooper, 1990; Peharz, 2015; Conaty et al., 2017), rendering many probabilistic reasoning tasks intractable. In particular, *Constrained Most Probable Explanation* (CMPE) (Rouhani et al., 2020) and *Marginal maximum-a-posteriori* (MMAP) (Peharz, 2015; Conaty et al., 2017; Mei et al., 2018) are computationally intractable. Furthermore, the *Most Probable Explanation* (MPE) task is NP-hard and challenging to approximate (Cooper, 1990; Park and Darwiche, 2004; de Campos, 2011; Conaty et al., 2017; Peharz, 2015). Similarly, approximating MMAP in SPNs, a widely used class of PCs, remains NP-hard with a complexity bound of 2^{n^δ} for fixed $0 \leq \delta < 1$, where n is the input size (Conaty et al., 2017; Mei et al., 2018). As a result, real-world applications rely on approximate inference methods that trade accuracy for computational efficiency, often producing suboptimal solutions.

While PMs enable probabilistic reasoning, their computational complexity grows with expressiveness, posing significant challenges for scalable inference. Exact inference in Probabilistic Models (PMs) is NP-hard (Cooper, 1990; Peharz, 2015; Conaty et al., 2017), rendering many probabilistic reasoning tasks intractable. In particular, both CMPE (Rouhani et al., 2020) and MMAP (Peharz, 2015; Conaty et al., 2017; Mei et al., 2018) are computationally infeasible. Moreover, MPE is not only NP-hard but also difficult to approximate (Cooper, 1990; Park and Darwiche, 2004; de Campos, 2011; Conaty et al., 2017; Peharz, 2015). Similarly, approximating MMAP in SPNs remains NP-hard, with a complexity bound of 2^{n^δ} for fixed $0 \leq \delta < 1$, where n is the input size (Conaty et al., 2017; Mei et al., 2018). Consequently, real-world applications depend on approximate inference methods that trade accuracy for efficiency, often yielding suboptimal or unreliable solutions.

To address these challenges, we propose scalable neural solvers for *Most Probable Explanation* (MPE), *Constrained Most Probable Explanation* (CMPE), and *Marginal maximum-a-posteriori* (MMAP), enabling efficient query answering. Additionally, we develop multi-linear integer programming techniques to improve inference in neurosymbolic models. These methods improve both tractability and accuracy, enabling real-time inference within the symbolic layer of neurosymbolic systems. In the following chapters, we systematically examine each of these inference tasks in the context of Probabilistic Models (PMs) and neurosymbolic models, outlining our contributions:

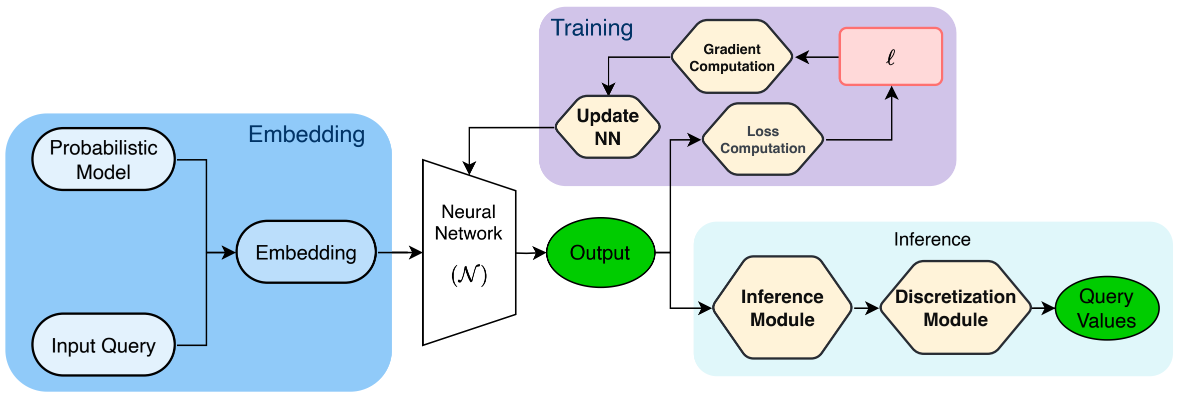


Figure 1.8: Pipeline for training and performing inference using neural-based solvers for probabilistic models. The framework supports both fixed-partition and any-partition inference tasks, including *Most Probable Explanation* (MPE), *Constrained Most Probable Explanation* (CMPE), and *Marginal maximum-a-posteriori* (MMAP). It enables performing these inference tasks over various probabilistic representations, such as Probabilistic Circuits (PCs), Probabilistic Graphical Models (PGMs), and Neural Autoregressive Models (NAMs). Given a probabilistic model, a neural network is trained to predict the most probable query variable assignments conditioned on the evidence. The pipeline comprises six key components: (1) NN Architecture, where a neural solver ($\mathcal{N}$) is trained to perform inference over probabilistic models; (2) Embeddings, which transform input queries and probabilistic model information into structured representations optimized for neural network processing; (3) Loss Function, which defines a tractable, differentiable objective (l) aligned with the target inference task; (4) Training, where gradient-based optimization updates the neural network using the defined loss function; (5) Inference, where the trained model is used at test time to infer query variable values given an input query; and (6) Discretization, which converts continuous neural outputs into discrete solutions.

- **Scalable and Accurate Unconstrained Inference in Probabilistic Models With Neural Solvers**

  Probabilistic representations, such as Probabilistic Circuits (PCs) (Choi et al., 2020a), graphical models (Koller and Friedman, 2009) including Bayesian Networks (BNs) and Markov Networks (MNs), and Neural Autoregressive Models (NAMs) (Larochelle and Murray, 2011; Germain et al., 2015), are widely used to model large, multi-dimensional probability distributions. However, as distributional complexity increases, solving NP-hard inference tasks like *Most Probable Explanation* (MPE) and *Marginal maximum-a-posteriori* (MMAP) through exact methods (Otten, 2012; Otten and Dechter, 2012) becomes computationally infeasible. While both exact and approximate solvers exist for PCs, BNs, and MNs, exact solvers are often too slow for real-time inference, while approximate solvers typically trade accuracy for efficiency. This issue is particularly pronounced in autoregressive models, where existing methods rely on inefficient search techniques, such as hill climbing and beam search, for inference.

  To address these challenges, we propose neural solvers that leverage insights from learning-to-optimize literature (Li and Malik, 2017; Fioretto et al., 2020; Donti et al., 2020; Zamzam and Baker, 2020; Park and Hentenryck, 2023). Given a Probabilistic Model (PM), we train a neural network—following the pipeline outlined in Figure 1.8—to predict the most probable query variable assignments conditioned on the evidence. Training follows either a supervised or self-supervised approach—supervised training requires access to exact inference, whereas self-supervised training circumvents this need, making it more practical for complex real-world models.

  In Chapter 3, we introduce the training procedure for fixed-partition inference, where evidence and query sets are predefined, resulting in an exponential number of possible input configurations. In Chapter 4, we extend this approach to any-partition inference, enabling arbitrary selection of evidence and query sets. This increased flexibility significantly increases the number of input configurations and variable partitions. In the any-partition setting, improving solution quality requires updating neural network parameters during inference, which increases inference time.

To eliminate the need for inference-time optimization in achieving near-optimal solutions for the any-partition task, we propose enhanced embedding schemes for PGMs and introduce more efficient discretization strategies. As detailed in Chapter 5, these advancements improve both solution quality and runtime performance, enabling scalable and accurate inference.

We conduct a comprehensive experimental evaluation of our methods across various PCs, PGMs, and NAMs, demonstrating that our neural solvers outperform state-of-the-art approximate inference techniques in both accuracy and computational efficiency.

- **Scalable and Accurate Constrained Inference in Probabilistic Models With Neural Solvers**

  In Chapter 6, we extend our neural solvers to handle inference queries that incorporate constraints, a critical requirement for modeling real-world scenarios. Domain-specific constraints guide optimization by embedding expert knowledge, improving solution relevance, and reducing the search space. Rouhani et al. (2020) introduced Constrained Most Probable Explanation (CMPE), an extension of the MPE task that integrates such constraints into the optimization process. Given two PGMs f and g defined over a set of random variables $\mathbf{X}$ and a real value q, the CMPE task seeks the most probable state $\mathbf{X} = \mathbf{x}$ with respect to f, subject to the constraint $g(\mathbf{x}) \leq q$. While both MPE and CMPE are NP-hard, CMPE is significantly more challenging. Notably, CMPE remains NP-hard even for zero-treewidth models (independent PGMs), where MPE can be solved in linear time. Rouhani et al. (2020) and

Rahman et al. (2021) showed that several probabilistic inference queries, including decision-preserving MPE (Choi et al., 2012), nearest assignment queries (Rouhani et al., 2018), and robust estimation (Darwiche and Hirth, 2023, 2020), can all be formulated as instances of CMPE.

We explore novel machine learning approaches for solving the CMPE task, drawing inspiration from recent progress in *learning to optimize* (Donti et al., 2020; Fioretto et al., 2020; Park and Hentenryck, 2023; Zamzam and Baker, 2020). These methods train deep neural networks to take as input the parameters and observations of a constrained optimization problem and directly output near-optimal solutions.

Our study represents the first empirical results in applying machine learning, both supervised and self-supervised, to solve the CMPE task in PGMs. Our experiments show that neural networks trained using the proposed self-supervised loss function consistently outperform models trained with existing supervised and self-supervised loss functions across several benchmark problems, demonstrating both improved efficiency and accuracy.

- **Advanced Inference Schemes for Neurosymbolic Models**

  Multi-Label Classification (MLC) assigns multiple labels to each instance, allowing the modeling of complex dependencies between inputs and outputs. It is fundamental in tasks such as text categorization, image classification, and bioinformatics, where instances can belong to multiple classes simultaneously. For example, an image may depict multiple objects, and a document may address several topics.

  Motivated by *neurosymbolic* architectures that combine probabilistic models (PMs) with neural networks (NNs) (Krishnan et al., 2015; Johnson et al., 2016), we adopt the Deep Dependency Network (DDN) (Guo and Weng, 2020) as introduced in Chapter 7. In this framework, a neural network extracts input features, while a dependency network (Heckerman et al., 2000) models the conditional distribution of each label based on these features and other labels.

However, a key limitation of DDNs is their reliance on basic inference methods, such as Gibbs sampling and mean-field approximation (Lowd and Shamaei, 2011; Lowd, 2012), which restricts their reasoning capabilities. To overcome this limitation and fully leverage the knowledge learned by the dependency network, we propose two advanced inference methods for the Most Probable Explanation (MPE) task in DDNs: a random-walk-based local search method and an approximation that reformulates MPE as an integer linear program, solvable using off-the-shelf commercial solvers such as Gurobi (Gurobi Optimization, LLC, 2023).

Our experiments on six datasets—Charades, TACoS, Wetlab, MS-COCO, PASCAL VOC 2007, and NUS-WIDE—demonstrate that DDNs equipped with these enhanced inference techniques outperform standalone neural networks and existing hybrid models, achieving superior reasoning and learning performance.

In the final chapter, we summarize the key research contributions of this dissertation and discuss potential directions for future work.

CHAPTER 2

BACKGROUND

This chapter provides an in-depth analysis of Probabilistic Models (PMs), their associated inference tasks, and both exact and approximate inference techniques applicable to these tasks. For more details on Dependency Networks, we refer readers to Heckerman et al. (2000). A comprehensive discussion of the Constrained Most Probable Explanation (CMPE) task is presented by Rouhani et al. (2020). For general background on probabilistic graphical models, we recommend Koller and Friedman (2009); Darwiche (2009), while Choi et al. (2020a) offers an extensive review of Probabilistic Circuits. For hypergraph neural networks, we recommend Feng et al. (2019), which also introduces the HGNN model.

2.1 Notation

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.

2.2 Probabilistic Model Representations

Throughout this dissertation, we use the term Probabilistic Models (PMs) to denote a broad class of models in which computing the likelihood[1] of a computing variable assignment can be performed

[1]Or a value proportional to it, such as the unnormalized probability in Markov networks or a quantity proportional to the pseudo-likelihood.

in polynomial time, preferably linear, in the size of the model. This class includes, but is not limited to, Bayesian and Markov networks, collectively referred to as Probabilistic Graphical Models (PGMs) (Koller and Friedman, 2009); smooth and decomposable Probabilistic Circuits (PCs) (Choi et al., 2020a), such as sum-product networks (SPNs) (Poon and Domingos, 2011), arithmetic circuits (Darwiche, 2003), AND/OR graphs (Dechter and Mateescu, 2007), cutset networks (Rahman et al., 2014), and probabilistic sentential decision diagrams (Kisa et al., 2014); and Neural Autoregressive Models (NAMs), including Neural Autoregressive Distribution Estimations (NADEs) (Larochelle and Murray, 2011) and Masked Autoencoder for Distribution Estimations (MADEs) (Germain et al., 2015).

2.2.1 Probabilistic Circuits (PCs)

A probabilistic circuit (PC) $\mathcal{M}$ (Choi et al., 2020a) 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: $\sum_{m \in \texttt{ch}(n)} \omega(m, n) = 1$.

We focus on a class of PCs which are *smooth and decomposable* (Choi et al., 2020a; 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 et al., 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 methods.

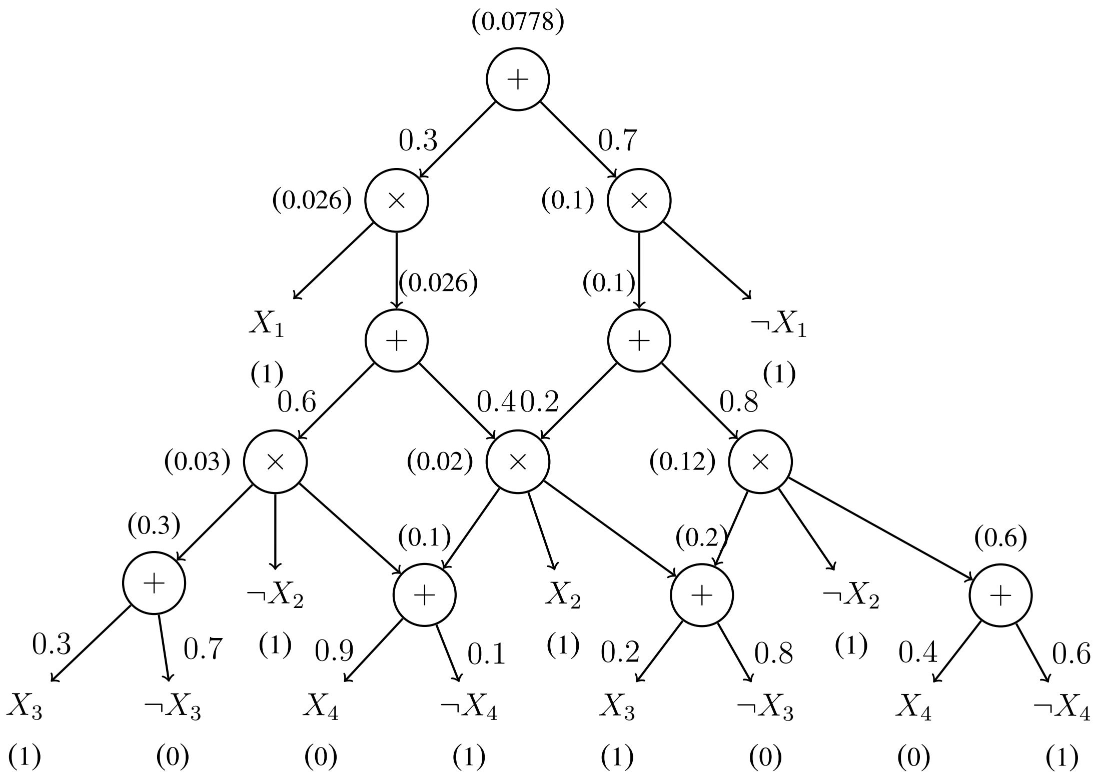


Figure 2.1: An example smooth and decomposable PC. The figure also shows value computation for answering the query $\text{p}_{\mathcal{M}}(X_3 = 1, X_4 = 0)$. The values of the leaf, sum, and product nodes are given in parentheses on their bottom, top, and left, respectively. The value of the root node is the answer to the query.

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 2.1 shows a smooth and decomposable probabilistic circuit defined over $\mathbf{X} = \{X_1, \ldots, X_4\}$.*

2.2.2 Neural Autoregressive Models (NAMs)

Neural Autoregressive Models (NAMs) are neural network-based models designed for estimating probability distributions. These models follow the product rule, where each variable X_i depends only on its predecessors in a predefined ordering, denoted by $\mathbf{X}_{<i}$. To enforce this autoregressive structure, techniques such as weight sharing and masking are employed, ensuring that each variable remains conditionally independent of its successors given its predecessors. Notable NAMs include the Masked Autoencoder for Distribution Estimation (MADE) (Germain et al., 2015) and Neural Autoregressive Distribution Estimation (NADE) (Larochelle and Murray, 2011). Both models enable efficient sampling and have demonstrated competitive performance in modeling binary and real-valued data. However, for inference tasks such as MPE, existing approaches are often restricted to computationally expensive and potentially inaccurate methods, such as hill-climbing and beam search.

2.2.3 Probabilistic Graphical Models (PGMs)

In this section, we focus on PGMs (Koller and Friedman, 2009), with particular emphasis on Markov networks and Bayesian networks, and demonstrate how they can be represented using multilinear polynomial representations.

Markov Networks

A **log-linear model**, also known as a **Markov random field** (MRF) or a **Markov network** (MN), denoted by $\mathcal{M}$, is an undirected probabilistic graphical model (Koller and Friedman, 2009) that is widely used in many real-world domains for representing and reasoning about uncertainty. Binary MRFs are defined as a triple $\langle \mathbf{X}, \mathcal{F}, \Theta \rangle$ where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of Boolean random variables, $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a set of features such that each feature f_i (we assume that a feature is a Boolean formula) is defined over a subset $\mathbf{D}_i$ of $\mathbf{X}$, and $\Theta = (\theta_1, \ldots, \theta_m)$ are real-valued weights or parameters, namely $\forall \theta_i \in \Theta;\ \theta_i \in \mathbb{R}$ such that each feature f_i is associated with a parameter θ_i. $\mathcal{M}$ represents the following probability distribution:

$$P(\mathbf{x}) = \frac{1}{Z(\Theta)} \exp\left\{ \sum_{i=1}^{m} \theta_i f_i\left(\mathbf{x}_{\mathbf{D}_i}\right) \right\} \tag{2.1}$$

where $\mathbf{x}_{\mathbf{D}_i}$ is the projection of $\mathbf{x}$ on the variables $\mathbf{D}_i$ of f_i, $f_i(\mathbf{x}_{\mathbf{D}_i})$ is an *indicator function* that equals 1 when $\mathbf{x}_{\mathbf{D}_i}$ evaluates f_i to `True` and is 0 otherwise, and $Z(\Theta)$ is the normalization constant called the *partition function*.

We focus on three tasks over MRFs: (1) structure learning; (2) posterior marginal inference; and (3) finding the most likely assignment to all the non-evidence variables given evidence (MPE inference). All of these tasks are at least NP-hard in general, and therefore approximate methods are often preferred over exact ones in practice.

A popular and fast method for structure learning is to learn binary pairwise MRFs (MRFs in which each feature is defined over at most two variables) by training an ℓ_1-regularized logistic regression classifier for each variable given all other variables as features (Wainwright et al., 2006; Lee et al., 2006). ℓ_1-regularization induces sparsity in that it encourages many weights to take the value zero. All non-zero weights are then converted into conjunctive features. Each conjunctive feature evaluates to `True` if both variables are assigned the value 1 and to `False` otherwise. Popular approaches for posterior marginal inference are the Gibbs sampling algorithm and generalized Belief propagation (Yedidia et al., 2000) techniques such as Iterative Join Graph Propagation (Mateescu et al., 2010). For MPE inference, a popular approach is to encode it as an integer linear programming (ILP) problem (Koller and Friedman, 2009) and then use off-the-shelf approaches such as Gurobi Optimization, LLC (2023) to solve the ILP.

Bayesian networks

Bayesian networks (BNs) are directed acyclic graphs that represent conditional dependencies among variables. They efficiently encode joint distributions via factorization, exploiting conditional independence relationships encoded in the graph structure. In BNs, the joint distribution factorizes as:

$$\mathrm{p}_{\mathcal{M}}(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathrm{Pa}(X_i))$$

where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is the set of random variables, and $\text{Pa}(X_i)$ denotes the parents of X_i in the directed graph. $\mathcal{M}$ denotes the Bayesian network.

Throughout this dissertation, we refer to both conditional probability tables (in BNs) and potential functions (in MNs) as *factors*, denoted by $\mathcal{F}_c$, where c represents the clique induced by the factor (in BNs, this is the clique formed by a node and its parents).

Multilinear Polynomial Representation

We use the multilinear polynomial representation (Sherali and Adams, 2009; Sherali and Tuncbilek, 1992; Horst and Tuy, 1996) to concisely describe our proposed methods 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

$$f(\mathbf{z}) = f(z_1, \ldots, z_n) = \sum_{I \in \mathcal{I}} w_I \prod_{i \in I} z_i \tag{2.2}$$

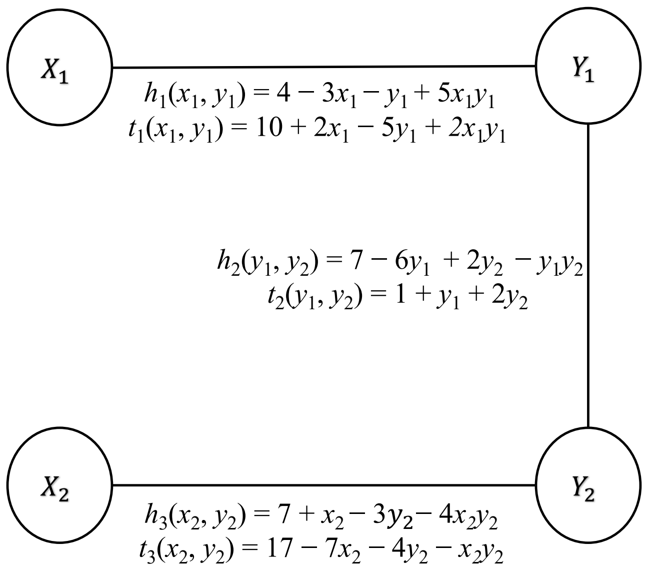
where $\mathbf{z} = (z_1, \ldots, z_n)$ is an assignment to all variables in $\mathbf{Z}$. We will call $f(\mathbf{z})$ the weight of $\mathbf{z}$.

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 2. *Figure 2.2 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.*

2.2.4 Dependency Networks (DNs)

Heckerman et al. (2000) introduced Dependency Networks (DNs), which represent the joint distribution using a set of local conditional probability distributions—one for each variable. Each


X_1
Y_1
$h_1(x_1, y_1) = 4 - 3x_1 - y_1 + 5x_1y_1$
$t_1(x_1, y_1) = 10 + 2x_1 - 5y_1 + 2x_1y_1$
$h_2(y_1, y_2) = 7 - 6y_1 + 2y_2 - y_1y_2$
$t_2(y_1, y_2) = 1 + y_1 + 2y_2$
X_2
Y_2
$h_3(x_2, y_2) = 7 + x_2 - 3y_2 - 4x_2y_2$
$t_3(x_2, y_2) = 17 - 7x_2 - 4y_2 - x_2y_2$

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 2.2: 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.

conditional defines the probability of a variable given all others. A DN is said to be consistent if there exists a joint distribution $P(\mathbf{x})$ such that each conditional distribution $P_i(x_i \mid \mathbf{x}_{-i})$, where $\mathbf{x}_{-i}$ denotes the projection of $\mathbf{x}$ onto $\mathbf{X} \setminus X_i$, is derived from $P(\mathbf{x})$.

A DN is learned from data by learning a classifier (e.g., logistic regression, multi-layer perceptron, etc.) for each variable, and thus DN learning is embarrassingly parallel. However, because the classifiers are independently learned from data, we often get an inconsistent DN. It has been conjectured (Heckerman et al., 2000) that most DNs learned from data are almost consistent in that only a few parameters need to be changed in order to make them consistent.

The most popular inference method over DNs is *fixed-order* Gibbs sampling (Liu, 2008). If the DN is consistent, then its conditional distributions are derived from a joint distribution $P(\mathbf{x})$, and the stationary distribution (namely the distribution that Gibbs sampling converges to) will be the same as $P(\mathbf{x})$. If the DN is inconsistent, then the stationary distribution of Gibbs sampling will be inconsistent with the conditional distributions.

2.3 Inference Tasks Over Probabilistic Models

2.3.1 Marginal Inference

Marginal inference (`MAR`) is a fundamental task in Probabilistic Models (PMs), where the goal is to compute the probability distribution over a subset of variables by summing out the remaining variables. Formally, given a joint distribution $P(X, Y)$, the marginal distribution over X is defined as:

$$P(X) = \sum_{Y} P(X, Y)$$

Marginal Inference over PCs

Given a Probabilistic Circuits (PCs) $\mathcal{M}$ 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 following recursive *value computations*:

$$v(n, \mathbf{q}) = \begin{cases} l(n, \mathbf{q}) & \text{if } n \in \mathcal{L} \\ \sum_{m \in \mathtt{ch}(n)} \omega(m, n) v(m, \mathbf{q}) & \text{if } n \in \mathcal{S} \\ \prod_{m \in \mathtt{ch}(n)} v(m, \mathbf{q}) & \text{if } n \in \mathcal{P} \end{cases} \tag{2.3}$$

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 3. *Figure 2.1 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 $p_{\mathcal{M}}(X_3 = 1, X_4 = 0)$.*

2.3.2 Most Probable Explanation

The Most Probable Explanation (MPE) task in Probabilistic Models (PMs) involves determining the most likely assignment to unobserved (non-evidence) variables, given observations (evidence). Formally, let $\mathcal{M}$ represent a PGM defined over a set of variables $\mathbf{X}$, with the associated distribution $\mathrm{p}_{\mathcal{M}}(\mathbf{X})$. The variables $\mathbf{X}$ are partitioned into evidence $\mathbf{E} \subseteq \mathbf{X}$ and query $\mathbf{Q} \subseteq \mathbf{X}$ sets, such that

$\mathbf{E} \cap \mathbf{Q} = \emptyset$ and $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. Given an assignment $\mathbf{e}$ to the evidence variables $\mathbf{E}$, the MPE task is formulated as:

$$\texttt{MPE}(\mathbf{Q}, \mathbf{e}) = \arg\max_{\mathbf{q}} \mathrm{p}_{\mathcal{M}}(\mathbf{q}|\mathbf{e}) = \arg\max_{\mathbf{q}} \{\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})\} \tag{2.4}$$

It is known that the MPE task is NP-hard in general and even hard to approximate (Cooper, 1990; Park and Darwiche, 2004; de Campos, 2011; Peharz, 2015; Conaty et al., 2017).

2.3.3 Marginal Maximum-a-Posteriori (MMAP)

Given a Probabilistic Model (PM) $\mathcal{M}$ 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 *Marginal maximum-a-posteriori*s (MMAPs) 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,

$$\texttt{MMAP}(\mathbf{Q}, \mathbf{e}) = \arg\max_{\mathbf{q}} \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}) \tag{2.5}$$

$$= \arg\max_{\mathbf{q}} \sum_{\mathbf{h} \in \{0,1\}^{|\mathbf{H}|}} \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q}, \mathbf{h}) \tag{2.6}$$

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 et al., 2017; Mei et al., 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).

2.3.4 Constrained Most Probable Explanation

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 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,

$$\underset{\mathbf{y}}{\text{maximize}} \ h(\mathbf{x}, \mathbf{y}) \ \ s.t. \ t(\mathbf{x}, \mathbf{y}) \leq q \tag{2.7}$$

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 4. *Given $X_1 = 1$, $X_2 = 1$ and $q = 20$, the CMPE solution of the example problem in Figure 2.2 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 equation 2.7 is equivalent to the following minimization problem:

$$\underset{\mathbf{y}}{\text{minimize}} \;\; f_{\mathbf{x}}(\mathbf{y}) \;\; s.t. \;\; g_{\mathbf{x}}(\mathbf{y}) \leq 0 \tag{2.8}$$

Let $\mathbf{y}^*$ be the optimal solution to the problem given in equation 2.8 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), equation 2.8 can be formulated as an (equivalent) integer linear programming (ILP) problem by introducing auxiliary integer variables for each multilinear term (e.g., $y_{1,2} = y_1 y_2$, $y_{2,3} = y_2 y_3$, 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, (equation 2.8) can be solved optimally using mixed integer linear programming (MILP) solvers such as Gurobi (Gurobi Optimization, LLC, 2023) 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 equation 2.8, 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 $p_{\mathbf{x}}^*$.

Specialized Algorithms for Lower Bounding $p_{\mathbf{x}}^*$

Recently, Rahman et al. (2021) proposed two new approaches for computing upper bounds on the optimal value of the maximization problem given in equation 2.7. 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: 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,

$$\max_{\mu \geq 0} d^*_\mu = \max_{\mu \geq 0} \min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \mu g_{\mathbf{x}}(\mathbf{y}) \leq p^*_{\mathbf{x}} \tag{2.9}$$

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, 2009b; 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 equation 2.9.

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 $p^*_{\mathbf{x}}$.

2.4 Hypergraphs

Hypergraphs extend the concept of traditional graphs by allowing edges, known as hyperedges, to connect more than two vertices. Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$

and a set of hyperedges $\mathcal{E}$, where each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$. Unlike standard graphs, where edges are binary relations between two nodes, hypergraphs capture higher-order interactions among groups of vertices, making them suitable for modeling complex relational data in domains such as social networks, biology, and machine learning. Hypergraphs can represent multi-way dependencies, which are challenging to model using simple pairwise connections.

In data representation learning, hypergraph-based approaches have gained prominence for their ability to capture higher-order correlations. Methods such as HGNN (Feng et al., 2019), HyperGCN (Yadati et al., 2019), hypergraph convolution and attention operators (Bai et al., 2019), HyperSAGE (Arya et al., 2020), HNHN (Dong et al., 2020) and UniGNN (Huang and Yang, 2021) leverage the hypergraph structure to model complex relationships, thereby enhancing the accuracy and robustness of learned representations.

CHAPTER 3

NEURAL NETWORK APPROXIMATORS FOR MARGINAL MAP IN PROBABILISTIC CIRCUITS

3.1 Introduction

Probabilistic circuits (PCs) (Choi et al., 2020a) such as sum-product networks (SPNs) (Poon and Domingos, 2011), arithmetic circuits (Darwiche, 2003), AND/OR graphs (Dechter and Mateescu, 2007), cutset networks (Rahman et al., 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 Probabilistic circuits (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 et al., 2014; Ping et al., 2015).

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

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 et al., 2017). Moreover, it is also NP-hard to approximate MMAP in SPNs to 2^{n^δ} for fixed $0 \leq \delta < 1$, where n is the input size (Conaty et al., 2017; Mei et al., 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 et al., 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 et al., 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 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 et al., 2019). Although this approach can provide fast answers at query time, it often yields MMAP solutions that are far from optimal.

In this chapter, 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, 2017; Fioretto et al., 2020; Donti et al., 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., 2022a). To the best of our knowledge, there is no prior work on solving MMAP in BNs, MNs, or PCs using NNs.

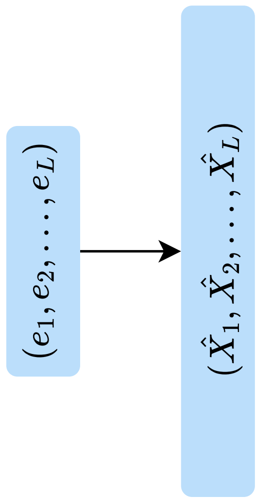
This chapter presents the following contributions. First, we propose to learn a neural network (NN) approximator for solving the MMAP task in PCs. Second, by leveraging the tractability of PCs, we devise a loss function whose gradient can be computed in time that scales linearly in the size of the PC, 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.

3.2 A Neural Optimizer for MMAP in PCs

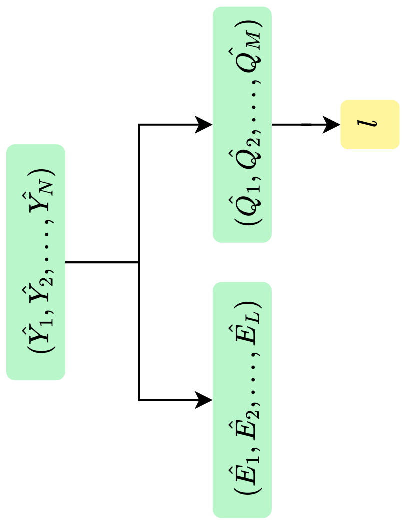
In this section, we introduce a learning-based approach using deep neural networks (NNs) to approximately solve the MMAP problem in PCs. Formally, the NN represents a function $f_\theta(.)$ that is parameterized by θ, and takes an assignment $\mathbf{e}$ over the evidence variables as input and outputs 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 chapter, we focus on answering fixed-partition queries, where the evidence variables $\left(\mathbf{E} = \{E_i\}_{i=1}^{N}\right)$ and query variables $\left(\mathbf{Q} = \{Q_j\}_{j=1}^{M}\right)$ are known *a priori* and remain fixed during both training and testing. 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).


$(e_1, e_2, \ldots, e_L)$
$(\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_L)$

(a) **Input Encoding:** This diagram depicts the encoding process for input features in fixed-partition queries. The input is a vector $(e_1, e_2, \ldots, e_L)$, representing the values of the evidence variables. These inputs serve as the encoded representation $(\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_L)$ for the neural network, facilitating the transformation of evidence variables into a suitable feature space for inference.


$(\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N)$
$(\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_M)$
l
$(\hat{E}_1, \hat{E}_2, \ldots, \hat{E}_L)$

(b) **Output Encoding:** This diagram illustrates the encoding scheme at the output stage for fixed-partition queries. The neural network generates continuous outputs $(\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N)$ for each variable in the probabilistic model after applying a sigmoid activation function. These outputs are then partitioned into two sets: predicted values for the query variables $(\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_M)$ and evidence variables $(\hat{E}_1, \hat{E}_2, \ldots, \hat{E}_L)$. The loss function is applied only to the query variables, ensuring the neural network is optimized specifically for inference tasks.

Figure 3.1: Encoding Schemes for Fixed-Partition Queries: This figure provides the encoding mechanisms used for fixed-partition queries, illustrating the transformation of input evidence values and the processing of neural network outputs into meaningful query variable values.

To represent neural network inputs for the fixed-partition task, we employ the encoding illustrated in Figure 3.1a. Since **E** and **Q** are known *a priori*, only the evidence variable values are provided as inputs to the network. The output encoding, depicted in Figure 3.1b, is designed to extract the query variable values from the neural network and apply the loss function. This encoding ensures that, following the application of the sigmoid activation function, the outputs remain within the continuous range $[0, 1]$.

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 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*.

3.2.1 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

[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 $\texttt{MMAP}(\mathbf{Q}, \mathbf{e})$.

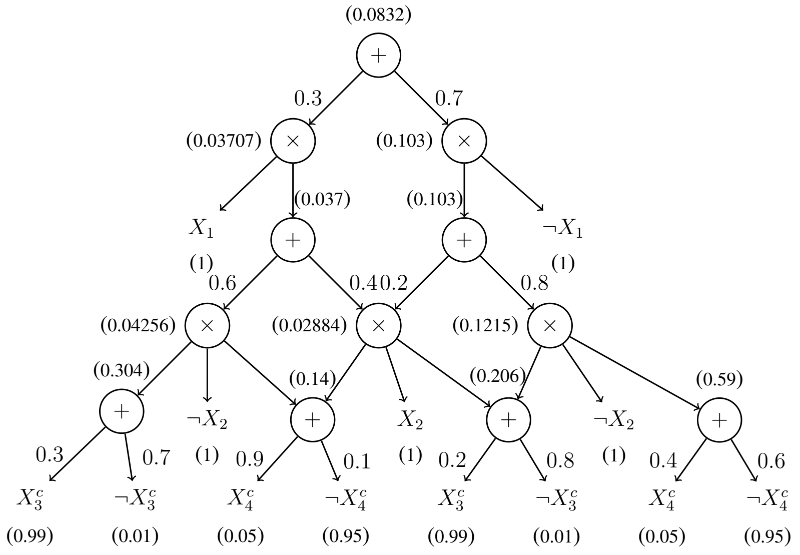


Figure 3.2: QPC obtained from the PC given in 2.1 for query variables $\{X_3, X_4\}$. For simplicity, here, we use an MMAP problem without any evidence. This is because given evidence can be incorporated into the PC by appropriately setting the leaf nodes. We also show value computations for the following 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.

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. equation 2.3 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

$$v'(n, (\mathbf{e}, \mathbf{q}^c)) = \begin{cases} l'(n, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{L} \\ \sum_{m \in \mathtt{ch}(n)} \omega(m, n) v'(m, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{S} \\ \prod_{m \in \mathtt{ch}(n)} v'(n, (\mathbf{e}, \mathbf{q}^c)) & \text{if } n \in \mathcal{P} \end{cases} \tag{3.1}$$

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 5. *Figure 3.2 shows the QPC corresponding to the PC shown in Figure 2.1. We also show value computations for the assignment* $(X_3^c = 0.99, X_4^c = 0.05)$.

3.2.2 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 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 3.2.1. *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* $\mathcal{M}$*.*

Proof. Without loss of generality, assume that the root node of the QPC is a sum node, the QPC has alternating levels of sum and product nodes and assume that we are computing the partial derivative of the QPC w.r.t. node X_j^c such that $X_j \in \mathbf{Q}_j$. Because the QPC is smooth, all child nodes of the root node must contain X_j^c. Therefore, by sum rule of derivative, the derivative at the root node is given by:

$$\frac{\partial v(r, \mathbf{e}, \mathbf{q})}{\partial x_j^c} = \sum_{m \in \mathtt{ch}(r)} \omega(m, r) \frac{\partial v(m, \mathbf{e}, \mathbf{q})}{\partial x_j^c} \tag{3.2}$$

We assume that the QPC is decomposable. As a result, at each product node n of the root, exactly one child node of n will be defined over X_j^c. Therefore, while computing the derivative,

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

we can treat the values of other child nodes as constants (since they do not depend on X_j^c). Let the child node of n that is defind over X_j^c be denoted by m_j. Then, the partial derivative of the product node w.r.t. X_j^c is given by:

$$\frac{\partial v(n, \mathbf{e}, \mathbf{q})}{\partial x_j^c} = \frac{\partial v(m_j, \mathbf{e}, \mathbf{q})}{\partial x_j^c} \prod_{m \in \mathtt{ch}(n) : m \neq m_j} v(m, \mathbf{e}, \mathbf{q}) \tag{3.3}$$

Continuing this analysis further, it is easy to see that the partial derivative of each sum and product node that mentions X_j^c (w.r.t. X_j^c) is given by equations equation 3.2 and equation 3.3 respectively.

The derivative of the leaf node labeled by X_j^c w.r.t. x_j^c equals 1, because it is assigned to x_j^c. Similarly, the derivative of the leaf node labeled by $\neg X_j^c$ w.r.t. x_j^c equals -1, because it is assigned to $1 - x_j^c$.

Thus, we observe that the only leaf assignments that change from the QPC used to compute the loss function are the ones labeled by X_j^c and $\neg X_j^c$. In particular, they go from x_j^c and $1 - x_j^c$ to 1 and -1 respectively. (While all other leaf assignments stay the same.)

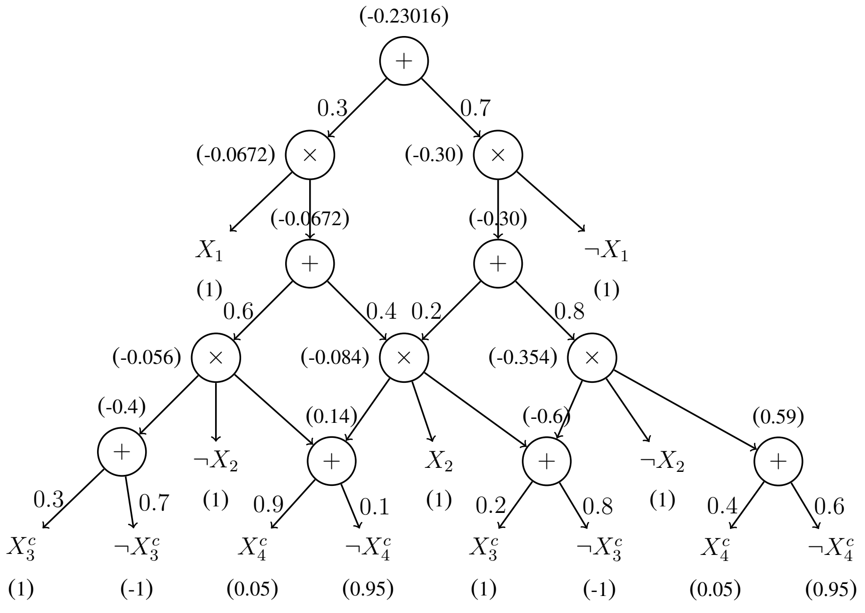
Therefore, the partial derivative of the QPC w.r.t. X_j^c can be computed via value computations using the new leaf assignments described above. Since the value computations require only one pass over the QPC, they run in linear time in the size of the QPC. $\square$

Example 6. *Figures 3.3(a) and 3.3(b) show the value computations for the partial derivative of the QPC w.r.t. X_3^c and X_4^c respectively.*

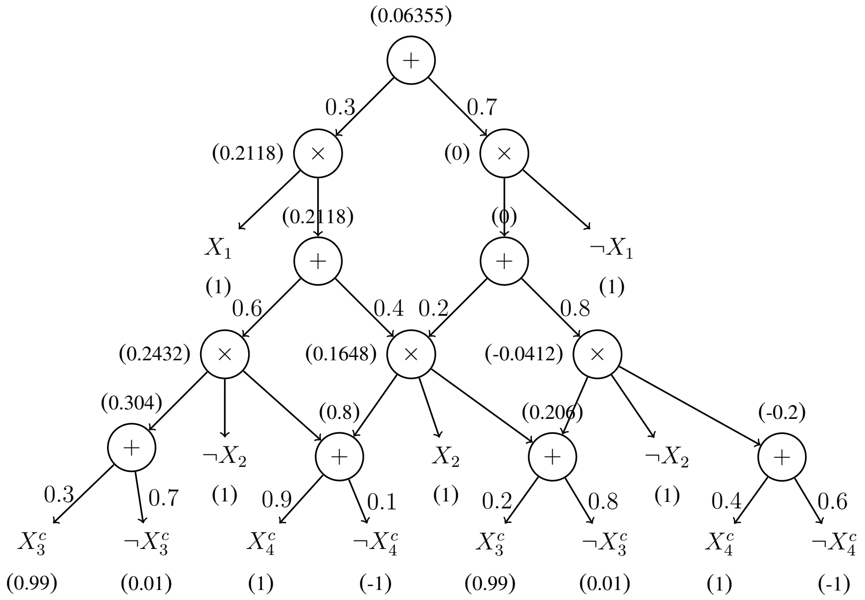
3.2.3 Improving the Loss Function

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:

$$\min_{\mathbf{q}^c} \{-\ln v'(r, (\mathbf{e}, \mathbf{q}^c))\} \leq \min_{\mathbf{q}} \{-\ln v(r, (\mathbf{e}, \mathbf{q}))\} \tag{3.4}$$



(a)



(b)

Figure 3.3: (a) Value computations for partial derivative of the QPC given in Figure 3.2 w.r.t. X_3^c and (b) Value computations for partial derivative of the QPC given in Figure 3.2 w.r.t. X_4^c. The values of the leaf, sum and product nodes are given in brackets on their bottom, top and left respectively. The value of the root node equals the partial derivative.

Since the right-hand side of the inequality given in equation 3.4 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

$$\ell(\mathbf{q}^c) = -\ln v'(r, (\mathbf{e}, \mathbf{q}^c)) - \alpha \sum_{j=1}^{M} q_j^c \log(q_j^c) + (1 - q_j^c) \log(1 - q_j^c) \tag{3.5}$$

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.

3.3 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.

3.3.1 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 $\mathrm{p}_{\mathcal{M}}(Q|\mathrm{e})$ and then set Q to

$$\arg\max_{j \in \{0,1\}} \mathrm{p}_{\mathcal{M}}(Q = j|\mathbf{e})$$

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 $\mathrm{p}_{\mathcal{M}}(q_j|\mathrm{e}, \mathbf{y})$ 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.

3.3.2 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 $\ln \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$.

3.3.3 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, 2012a; 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 et al., 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.

Tables 3.1 and 3.2 provide details of the datasets and the PCs used in our experiments. The selected datasets cover a diverse range of scenarios, serving as representative benchmarks to comprehensively evaluate the performance and scalability of our algorithms. Notably, for the EMNIST dataset, we specifically use classes from the EMNIST Letters subset.

Table 3.1: Overview of Dataset and PCs TPM Datasets

Dataset	**#Variables**	**#Nodes in PC**
nltcs	16	125
msnbc	17	46
kdd	64	274
plants	69	3737
baudio	100	348
bnetflix	100	400
jester	100	274
accidents	111	1178
mushrooms	112	902
connect4	126	2128
tretail	135	359
rcv1	150	519
pumsb star	163	2860
dna	180	1855
kosarek	190	779
tmovie	500	7343
book	500	1628
cwebkb	839	3154
cr52	889	7348
c20ng	910	2467
moviereview	1001	2567
bbc	1058	3399

Table 3.2: CIFAR, MNIST, EMNIST Letters: Dataset and PC Overview

Dataset Name	**Class**	**#Variables**	**#Nodes in PC**
cifar10	0	512	1294
cifar10	1	512	1326
cifar10	2	512	1280
cifar10	3	512	1684
cifar10	4	512	1743
cifar10	5	512	1632
cifar10	6	512	1796
cifar10	7	512	1359
cifar10	8	512	1300
cifar10	9	512	1279
emnist	1	784	2125
emnist	2	784	2124
emnist	3	784	2100
emnist	4	784	2122
emnist	19	784	2684
emnist	20	784	2147
emnist	22	784	2060
emnist	24	784	2734
emnist	26	784	2137
mnist	0	784	3960
mnist	1	784	4324
mnist	2	784	4660
mnist	3	784	4465
mnist	4	784	4510
mnist	5	784	4063
mnist	6	784	3776
mnist	7	784	4408
mnist	8	784	3804
mnist	9	784	3714

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 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.

3.3.4 Neural Network Optimizers

For each PC and query ratio combination, we trained a corresponding neural network (NN) using the loss function described in Section 3.2.3. 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, 2015) 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.

In our experimental setup, we used a consistent minibatch size of 128 instances across all experiments. To facilitate effective training, we applied a learning rate decay strategy, reducing the learning rate by a factor of 0.9 after a fixed number of epochs. For discrete scenarios, a single hyperparameter, α, was required. We selected its value for our loss function (see equation 3.5) via 5-fold cross-validation, exploring the range $\{0.01, 0.1, 1, 10, 100, 1000\}$ to determine the optimal setting.

3.3.5 Results on the TPM Datasets

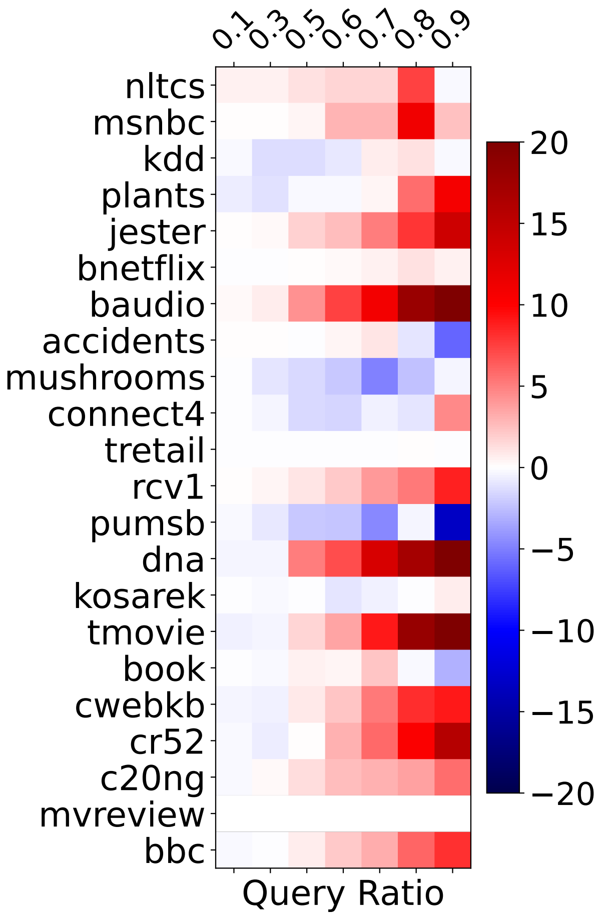
Table 3.3: Contingency tables for competing methods across MPE and MMAP Problems, including initial and Hill Climbing Search comparisons. Highlighted values represent results for `SSMP`.

	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

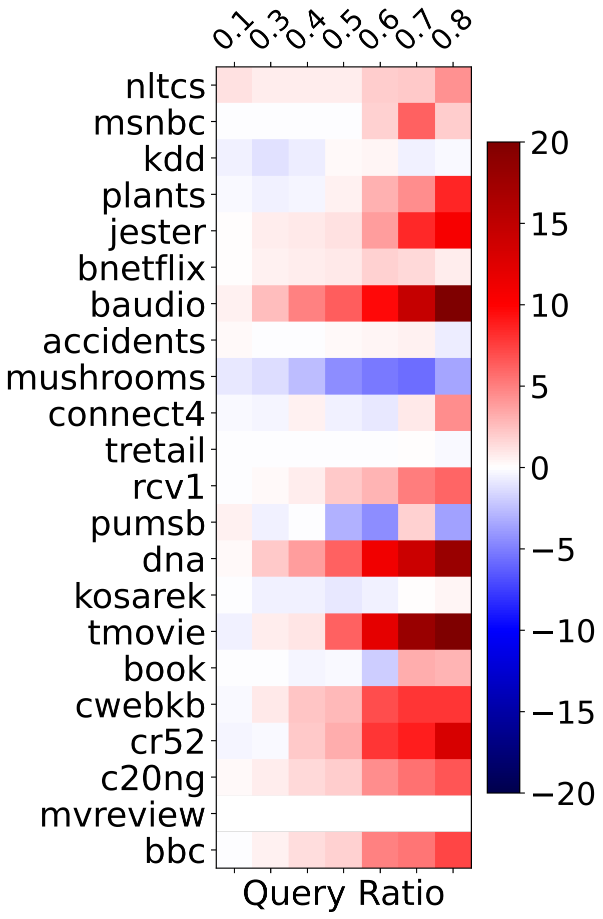
We summarize our results for the competing schemes (3 baselines and `SSMP`) on the 22 TPM datasets using the first two contingency tables given in Table 3.3, one for MPE and one for MMAP. Recall that we generated 154 test datasets each for MPE and MMAP (22 PCs $\times$ 7 qr values). In all contingency 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 3.3, 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 3.3) 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 3.3. 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 3.4: 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 3.4, 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 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

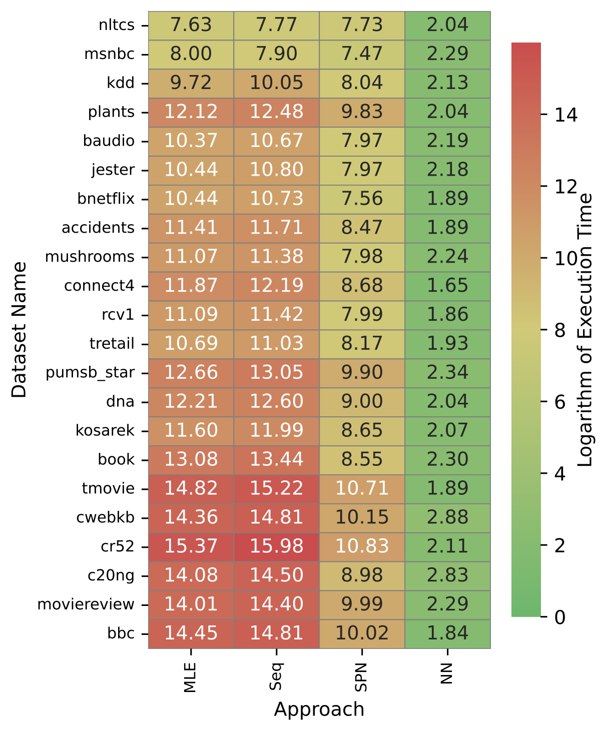
$$\%\text{Diff.} = \frac{ll_{ssmp} - ll_{max}}{|ll_{max}|} \times 100 \tag{3.6}$$

From the heatmap for MPE given in Figure 3.4(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.

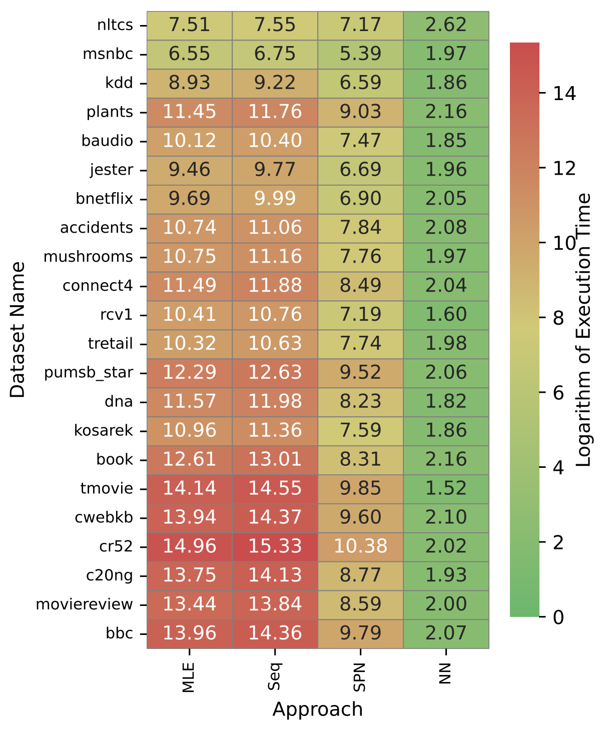
The heatmaps for MMAP are illustrated in Figure 3.4(b). We see a similar trend as the one for MPE; SSMP remains competitive with the Max approximation, particularly when the number of query variables is small. While SSMP outperforms (with some exceptions) the Max approximation when the number of query variables is large.

3.3.6 Inference Time Comparison on TPM Datasets

We present the inference times for the polytime baseline methods and our proposed SSMP method, as illustrated in Figure 3.5. The inference times for MPE queries are depicted in Figure 3.5a, while those for MMAP queries are showcased in Figure 3.5b. The values in each cell correspond to the natural logarithm of the time in microseconds for each method and dataset. Lower values are represented by green, while higher values are denoted by red. Notably, our method exhibits a substantial performance advantage over all other methods. Additionally, it is noteworthy that as the dataset size (and consequently the SPN size) increases, the inference time for the baseline methods (ML, Seq, and Max) increases, whereas our method's inference time remains almost the same. This distinction arises because the baseline methods depend on the SPN size during inference, whereas our approach allows us to regulate inference time by modifying the neural network's size without impacting the SPN.



(a) MPE



(b) MMAP

Figure 3.5: Heatmap illustrating inference time for `ML`, `Seq`, `Max`, and `SSMP` methods on a Logarithmic micro-second scale. The color green indicates shorter (more favorable) time.

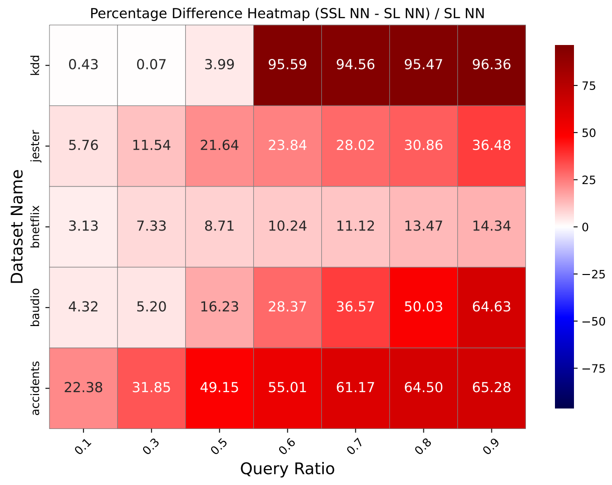
On average `SSMP` requires in the order of 7-10 micro-seconds for MMAP inference on an A40 GPU. The `Max` approximation takes 7 milli-seconds (namely, `SSMP` is almost 1000 times faster). In comparison, as expected, the `Seq` and `ML` approximations are quite slow, requiring roughly 400 to 600 milliseconds to answer MPE and MMAP queries.

3.3.7 A Comparative Analysis: `SSMP` Method Versus Supervised Learning Approaches

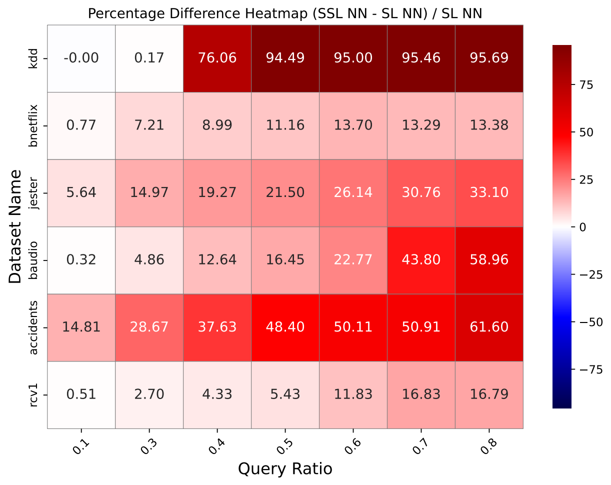
Label Generation for Supervised Learning Benchmarks

To compare the performance of `SSMP` method with a supervised learning approach, we employed Stochastic Hill Climbing Search. This choice was motivated by the challenge posed by the large number of potential query variables in our experimental setup. Given the complexity of obtaining exact solutions in this setting, we opted for a more practical alternative.

Our approach involved initializing the query variables randomly and performing 1000 iterations for each problem instance. The resulting labels were then used to train a supervised neural network. This methodology allowed us to effectively address the challenges associated with a large query space, enabling a meaningful comparative evaluation between SSMP and the supervised learning approach.



(a) MPE



(b) MMAP

Figure 3.6: Heatmap showing the percentage difference in log-likelihood scores between SSMP and Supervised Learning method. Blue color represents the supervised method's superiority (negative values), while red color represents SSMP's superiority (positive values). The datasets are arranged in ascending order of their number of variables.

Comparison with Supervised Neural Network Training

We conduct a comparative analysis of our approach against supervised learning methods to assess its performance in the heatmaps shown in Figure 3.6. The heatmap on the left illustrates the percentage difference in log-likelihood scores for MPE, while the one on the right depicts the corresponding difference for MMAP. Labels for training the supervised method are obtained through the procedure outlined in the previous section. Subsequently, training is executed using the mean squared error (MSE) criterion. We adopt an identical neural network architecture as employed in SSMP method, maintaining consistent training protocols across all aspects except for the training procedure and loss function. This facilitates a direct comparison of these methods, exclusively in terms of their training processes and associated losses.

By observing the heatmaps, we can confirm that our method consistently surpasses the performance of the supervised approach across all cases, except for one instance (specifically, for the kdd dataset and query = 0.1 for MMAP), where both methods exhibit equivalent log-likelihood scores. The supervised method closely aligns with our approach for smaller query sets across most datasets. However, as the query variable count grows, the supervised method's efficacy diminishes. This trend is also pronounced when increasing the variable count within the datasets, leading to a decrease in the supervised method's performance, as depicted in the heatmap. This underscores the need for a self-supervised approach that operates independently of true labels. Notably, the training duration for supervised methods encompasses the time required to obtain the true labels, whereas our method depends solely on a trained probabilistic classifier (PC).

3.3.8 Results on the CIFAR-10 Dataset

Table 3.4: Contingency tables comparing competing methods for MPE on CIFAR, MNIST and EMNIST datasets. Highlighted values represent results for `SSMP`.

	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

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 3.4. 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).

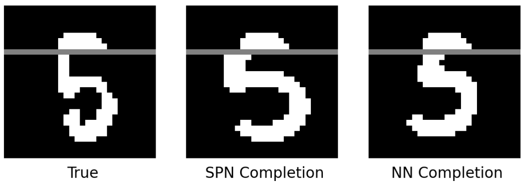
3.3.9 Results on the MNIST and EMNIST Datasets

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 3.4. 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.

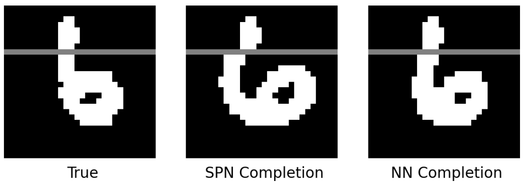
Completing Images in MNIST and EMNIST

Qualitative image completions for MNIST and EMNIST datasets are presented in Figure 3.7 for MNIST Digits {5, 6, 8, 9} and EMNIST characters {a, c, s, v} by `SSMP` and `Max`. Beginning with the original image as the initial reference, followed by the `Max` method's completion attempt, and culminating in our novel `SSMP` method's completion. Notably, `SSMP` demonstrates equal or superior performance compared to the `Max` method. In particular, characters such as 8, 9, A, and S display discrepancies in their `Max`-generated completions, where the lower segments fail to align with their upper segment. In stark contrast, the proposed `SSMP` method offers a remarkable improvement in completion quality. It adeptly addresses the difficulties posed by limited available evidence and generates reconstructions that are not only smoother but also exhibit remarkable coherence. This highlights `SSMP` method's efficacy in real-world scenarios with limited information.

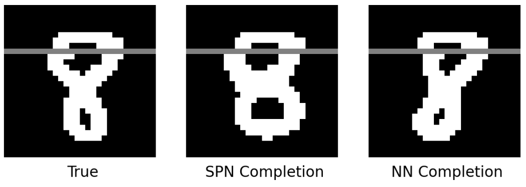
In summary, we find that, on average, our proposed method (`SSMP`) is better than other baseline MPE/MMAP approximations in terms of log-likelihood score. Moreover, it is substantially better than the baseline methods when the number of query variables is large. Also, once learned from data, it is also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



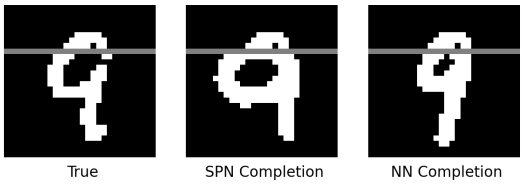
(a) MNIST Digit 5



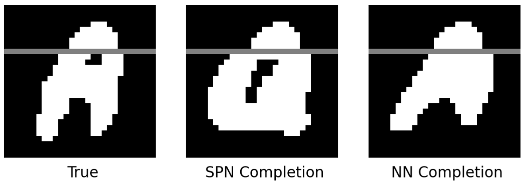
(b) MNIST Digit 6



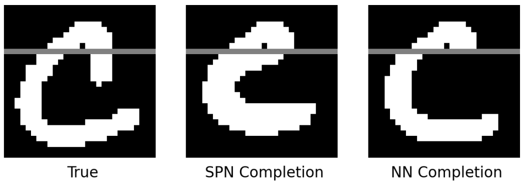
(c) MNIST Digit 8



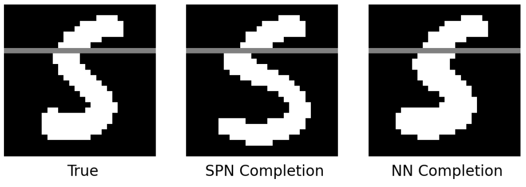
(d) MNIST Digit 9



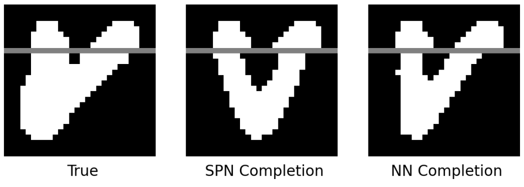
(e) EMNIST Character A



(f) EMNIST Character C



(g) EMNIST Character S



(h) EMNIST Character V

Figure 3.7: Image Completions for MNIST and EMNIST datasets. First image: Original. Second image: `Max` Completion. Third image: `SSMP` Completion. The gray line separates evidence (above) from query (below).

3.4 Chapter Summary

In this chapter, 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.

CHAPTER 4

A NEURAL NETWORK APPROACH FOR EFFICIENTLY ANSWERING MOST PROBABLE EXPLANATION QUERIES IN PROBABILISTIC MODELS

4.1 Introduction

Probabilistic representations such as Probabilistic Circuits (PCs) (Choi et al., 2020a), graphical models (Koller and Friedman, 2009) such as Bayesian Networks (BNs) and Markov Networks (MNs), and Neural Autoregressive Models (NAMs) (Larochelle and Murray, 2011) are widely used to model large, multi-dimensional probability distributions. However, they face a significant challenge: as the complexity of these distributions increases, solving practically relevant NP-hard inference tasks such as finding the Most Probable Explanation (MPE) via exact inference techniques (Otten, 2012; Marinescu and Dechter, 2009a) becomes increasingly difficult and time-consuming. In particular, although various exact and approximate solvers exist for the MPE task in PCs, BNs and MNs, exact solvers are often too slow for practical use, and approximate solvers tend to lack the necessary accuracy, particularly in autoregressive models that currently rely on slow hill-climbing/beam search methods.

In recent work, Arya et al. (Arya et al., 2024b) proposed a method to overcome the limitations of existing approximate methods by using neural networks (NNs) to solve the MPE task in PCs.[1] Their method draws inspiration from the learning to optimize literature (Li and Malik, 2017; Fioretto et al., 2020; Donti et al., 2020; Zamzam and Baker, 2020; Park and Hentenryck, 2023). Given a PC and a *predefined partition of variables into query and evidence sets*, the core idea is to train a NN that takes an assignment to the evidence variables as input and outputs the most likely assignment to the query variables w.r.t. the distribution defined by the PC. Arya et al. suggest

[1] Arya et al. (2024b) developed a NN-based method for solving the *marginal maximum-a-posteriori* (MMAP) task in PCs. We focus on the MPE task, also sometimes referred to as the full MAP task, which is a special case of MMAP. Our method can be easily extended for solving the MMAP problem in PCs and tractable graphical models. For simplicity of exposition, we concentrate on the MPE task in this chapter.

using either supervised or self-supervised learning techniques to train the NN; the former requires access to exact inference schemes, while the latter does not and is therefore more practical.

In this chapter, we address a more general and complex version of the MPE task than the one considered by Arya et al. Specifically, we assume that there is *no predefined partition of the variables into evidence and query sets*, which we refer to as the **any-MPE** task. The complexity of the any-MPE task arises from the exponential increase in the number of input configurations, compounded by the exponential number of possible divisions of variables into evidence and query sets. Furthermore, our method applies to a broad class of probabilistic models, including BNs, MNs and NAMs, whereas Arya et al.'s method is limited to PCs. In addition, Arya et al.'s method does not fully exploit the capabilities of self-supervision, and the benefits of combining supervised and self-supervised loss functions.

In this chapter we presents a novel approach that uses a NN for solving the any-MPE task in a broad class of probabilistic models (PMs) and achieves technical advancements in three key aspects:

1. Efficient MPE Inference via Encoding Scheme and Loss Function: We introduce a new encoding scheme that tailors the NN architecture to the specific structure of the input PM. This scheme not only delineates the input and output nodes for the NN but also establishes a methodology for setting input values and extracting the MPE solution from the NN's outputs. Furthermore, we propose a tractable, and differentiable self-supervised loss function, enabling efficient training.

2. Inference Time Optimization with ITSELF: We introduce a novel inference technique called Inference Time Self Supervised Training (ITSELF). This technique iteratively refines the MPE solution during the inference process itself. It utilizes gradient descent (*back-propagation*) to update the NN's parameters using our proposed self-supervised loss, leading to continual (any-time) improvement towards near-optimal solutions. ITSELF fully utilizes the power of our self-supervised loss, as it does not require labeled data or an external MPE solver.

3. Two-Phase Pre-training with Teacher-Student Architecture: To address challenges associated with self-supervised learning and ITSELF, we propose a two-phase pre-training strategy that leverages a teacher-student architecture. Self-supervised learning can suffer from overfitting and requires careful regularization. Additionally, ITSELF, especially with random initializations, might necessitate a substantial number of gradient updates to converge on optimal solutions. Our approach addresses these issues using the following methodology: (i) The teacher network first overfits the training data using ITSELF and (ii) The student network is then trained using supervised loss functions (e.g., binary cross-entropy) by treating the teacher network's output as pseudo-labels. This supervised training phase improves and regularizes the parameter learning process of the student network. It also provides a robust starting point for ITSELF, significantly reducing the required optimization steps and leading to substantial performance gains.

Finally, we conduct a detailed experimental comparison of our method with existing approaches on several types of PMs such as PCs, PGMs and NAMs. Our results demonstrate that our method surpasses state-of-the-art approximate inference techniques in terms of both accuracy and speed.

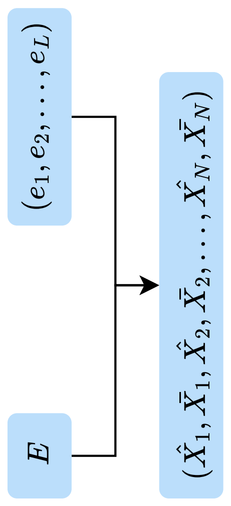
Motivation: The goal of this chapter is to develop a method that trains a NN for a given PM and, at test time, serves as an approximate MPE solver for any-MPE query posed over the PM. By any-MPE, we mean that the NN can take an assignment to an arbitrary subset of variables (evidence) as input and output the most likely assignments to the remaining (query) variables. Recently, Arya et al. (Arya et al., 2024b) proposed a NN-based solution for solving the MPE task in PCs under the constraint that the partition of the variables into evidence and query sets *is known before training the NN*. This constraint is highly restrictive because, for generative models, it is unlikely that such a partition of variables is known in advance. In such cases, one would typically train a discriminative model rather than a generative one. Unlike Arya et al.'s method, our approach yields an any-MPE solver. Additionally, Arya et al.'s approach has several limitations in that it does not fully exploit the benefits of self-supervision during inference time and requires the use of relatively large NNs to achieve good performance in practice. Our proposed approach, described next, addresses these limitations.

4.2 A Self-Supervised Neural Approximator for any-MPE

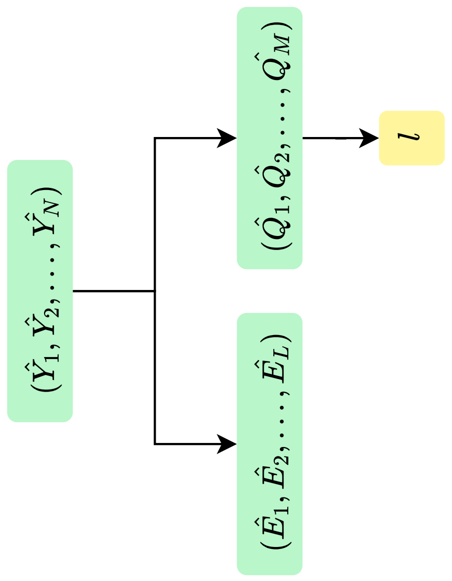
In this section, we develop a neural network (NN) based approach for solving the *any-MPE* task. Specifically, given a PM, we develop an input encoding (see Section 4.2.1) that determines the number of input nodes of the NN and sets their values for the given MPE query. Additionally, we develop an output encoding scheme that specifies the number of NN output nodes required for the given PM and enables the recovery of the MPE solution from the outputs. For training the NN, we introduce a tractable and differentiable self-supervised loss function (see Section 4.2.2), whose global minima aligns with the MPE solutions to efficiently learn the parameters of the NN given *unlabeled data*.

4.2.1 An Encoding For any-MPE Instances

Since NNs require fixed-sized inputs and outputs, we introduce input and output encodings that generate fixed-length input and output vectors for each PM from a given MPE problem instance $\texttt{MPE}(\mathbf{Q}, \mathbf{e})$. To encode the input, for each variable $X_i \in \mathbf{X}$, we associate two input nodes in the NN, denoted by $\hat{X}_i$ and $\bar{X}_i$. Thus for a PM having n (namely, $|\mathbf{X}| = n$) variables, the corresponding NN has $2n$ input nodes. Given a query MPE$(\mathbf{Q}, \mathbf{e})$, we set the values of the input nodes as follows: (1) If $X_i \in \mathbf{E}$ and $X_i = 0$ is in $\mathbf{e}$, then we set $\hat{X}_i = 0$ and $\bar{X}_i = 1$; (2) If $X_i \in \mathbf{E}$ and $X_i = 1$ is in $\mathbf{e}$, then we set $\hat{X}_i = 1$ and $\bar{X}_i = 0$; and (3) If $X_i \in \mathbf{Q}$ then we set $\hat{X}_i = 0$ and $\bar{X}_i = 0$. (The assignment $\hat{X}_i = 1$ and $\bar{X}_i = 1$ is not used.) It is easy to see that the input encoding described above yields an *injective* mapping between the set of all possible MPE queries over the given PM and the set $\{0, 1\}^{2n}$. This means that each unique MPE query $(\mathbf{Q}, \mathbf{e})$ will yield a unique 0-1 input vector of size $2n$. We illustrate the input encoding in Figure 4.1a.


$(e_1, e_2, \ldots, e_L)$
E
$(\hat{X}_1, \bar{X}_1, \hat{X}_2, \bar{X}_2, \ldots, \hat{X}_N, \bar{X}_N)$

(a) **Input Encoding for any-partition Queries:** This diagram illustrates the input encoding scheme for any-partition queries. The input consists of an evidence set E, which includes evidence variables and their corresponding values $(e_1, e_2, \ldots, e_L)$. Each evidence variable is encoded as two neural network input nodes, resulting in an expanded feature set of size $2N$: (1) $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_N$: First node for the evidence variables. (2) $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_N$: Second node for the evidence variables.


$(\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N)$
$(\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_M)$
$(\hat{E}_1, \hat{E}_2, \ldots, \hat{E}_L)$
l

(b) **Output Encoding for Any-Partition Queries:** The output encoding follows the same scheme as in fixed-partition queries. The neural network produces continuous outputs $(\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N)$, which are used to generate two sets of outputs: (1) The predicted continuous values for the query variables $(\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_M)$. (2) The reconstructed values for the evidence variables $(\hat{E}_1, \hat{E}_2, \ldots, \hat{E}_L)$. Finally, the loss function is applied only to the query variables, ensuring the model is trained for inference tasks.

Figure 4.1: Encoding Schemes for Any-Partition Queries. This figure provides the encoding mechanisms used for any-partition queries, highlighting the transformation of input evidence variables into an extended encoding space and the processing of neural network outputs into query variable values.

The output of the neural network comprises of n nodes with sigmoid activation, where each output node is associated with a variable $X_i \in \mathbf{X}$. We ignore the outputs corresponding to the evidence variables and define a loss function over the outputs corresponding to the query variables in the set $\mathbf{Q}$. Figure 4.1b depicts the output encoding for the any-partition task. The MPE solution can be reconstructed from the output nodes of the NN by thresholding the output nodes corresponding to the query variables appropriately (e.g., if the value of the output node is greater than 0.5, then the query variable is assigned the value 1; otherwise it is assigned to 0).

4.2.2 A Self-Supervised Loss Function for any-MPE

Since the output nodes of our proposed NN use sigmoid activation, each output is continuous and lies in the range $[0, 1]$. Given an MPE query $\texttt{MPE}(\mathbf{Q}, \mathbf{e})$, let $\mathbf{q}^c \in [0, 1]^{|\mathbf{Q}|}$ denote the (continuous) MPE assignment predicted by the NN. In MPE inference, given $\mathbf{e}$, we want to find an assignment $\mathbf{q}$ such that $\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$ is maximized, namely, $-\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$ is minimized. Thus, a natural loss function that we can use is $-\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$. Unfortunately, the NN outputs a continuous vector $\mathbf{q}^c$ and as a result $\mathrm{p}_{\mathcal{M}}(\mathbf{q}^c, \mathbf{e})$ is not defined.

Next, we describe how to solve the above problem by leveraging the following property of the class of PMs that we consider in this dissertation—specifically BNs, MNs, PCs and NAMs. In these PMs, the function $\ell(\mathbf{q}, \mathbf{e}) = -\log \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$, which is a function from $\{0, 1\}^n \rightarrow \mathbb{R}$ is either a multi-linear polynomial or a neural network, and can be computed in linear time in the size of the PM. To facilitate the use of continuous outputs, we define a loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) : [0, 1]^n \rightarrow \mathbb{R}$ such that ℓ^c coincides with ℓ on $\{0, 1\}^n$. For PGMs and PCs, ℓ is a multi-linear function and ℓ^c is obtained by substituting each occurrence of a discrete variable $q_i \in \mathbf{q}$ with the corresponding continuous variable $q_i^c \in \mathbf{q}^c$ where $q_i^c \in [0, 1]$. In NAMs, ℓ is a NN and we can perform a similar substitution—we substitute each binary input q_i in the NN with a continuous variable $q_i^c \in [0, 1]$. This substitution transforms the discrete NN into a continuous function while preserving its functional form.

An important property of ℓ^c is that it can be evaluated and differentiated in polynomial time. Moreover, when ℓ is defined by either a neural network (in NAMs) or a multilinear function (in BNs, MNs and PCs), the minimum value of ℓ^c over the domain $[0, 1]^n$ is less than or equal to the minimum value of the original function ℓ over the discrete domain $\{0, 1\}^n$. Formally,

Proposition 4.2.1. *Let* $l(\mathbf{q}, \mathbf{e}) : \{0,1\}^n \rightarrow \mathbb{R}$ *be either a neural network or a multilinear function, and let* $l^c(\mathbf{q}^c, \mathbf{e}) : [0,1]^n \rightarrow \mathbb{R}$ *be its continuous extension obtained by substituting each binary input* q_i *with a continuous variable* $q_i^c \in [0,1]$*. Then,*

$$\min_{\mathbf{q}^c \in [0,1]^n} \ell^c(\mathbf{q}^c, \mathbf{e}) \leq \min_{\mathbf{q} \in \{0,1\}^n} \ell(\mathbf{q}, \mathbf{e})$$

Following Arya et al. (Arya et al., 2024b), we propose to improve the quality of the loss function by tightening the lower bound given in proposition 4.2.1 with an entropy-based penalty (ℓ_E), governed by $\alpha > 0$.

$$\ell_E(\mathbf{q}^c, \alpha) = -\alpha \sum_{j=1}^{|\mathbf{Q}|} \left[q_j^c \log(q_j^c) + (1 - q_j^c) \log(1 - q_j^c) \right] \tag{4.1}$$

This penalty encourages discrete solutions by preferring q_j^c values close to 0 or 1, where α modulates the trade-off. Setting α to 0 yields the continuous approximation; conversely, an α value of ∞ results exclusively in discrete outcomes. From proposition 4.2.1 and by using the theory of Lagrange multipliers, we can show that for any $\alpha > 0$, the use of the entropy penalty yields a tighter lower bound:

Proposition 4.2.2.

$$\min_{\mathbf{q}^c \in [0,1]^n} \ell^c(\mathbf{q}^c, \mathbf{e}) \leq \min_{\mathbf{q}^c \in [0,1]^n} \ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha) \leq \min_{\mathbf{q} \in \{0,1\}^n} \ell(\mathbf{q}, \mathbf{e})$$

How to use the Loss Function: Given a PM defined over n variables, we can use the self-supervised loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$ (treating α as a hyper-parameter) to train any neural network (NN) architecture that has $2n$ input nodes and n output nodes. This trained NN can then be used to answer any arbitrary MPE query posed over the PM. The training data for the neural network consists of assignments (evidence $\mathbf{e}$) to a subset of the variables. Each training example can be generated using the following three-step process. We first sample a full assignment $\mathbf{x}$ to all variables in the PM using techniques like Gibbs sampling or perfect sampling for tractable distributions such as PCs and BNs. Second, we choose an integer k uniformly at random from the range $\{1, \ldots, n\}$ and designate k randomly selected variables as evidence variables $\mathbf{E}$, and the remaining $n-k$ as query variables $\mathbf{Q}$. Finally, we project the full assignment $\mathbf{x}$ on $\mathbf{E}$. The primary advantage of using the self-supervised loss function is that it eliminates the need for access to a dedicated MPE solver to provide supervision during training; gradient-based training of the neural network provides the necessary supervision.

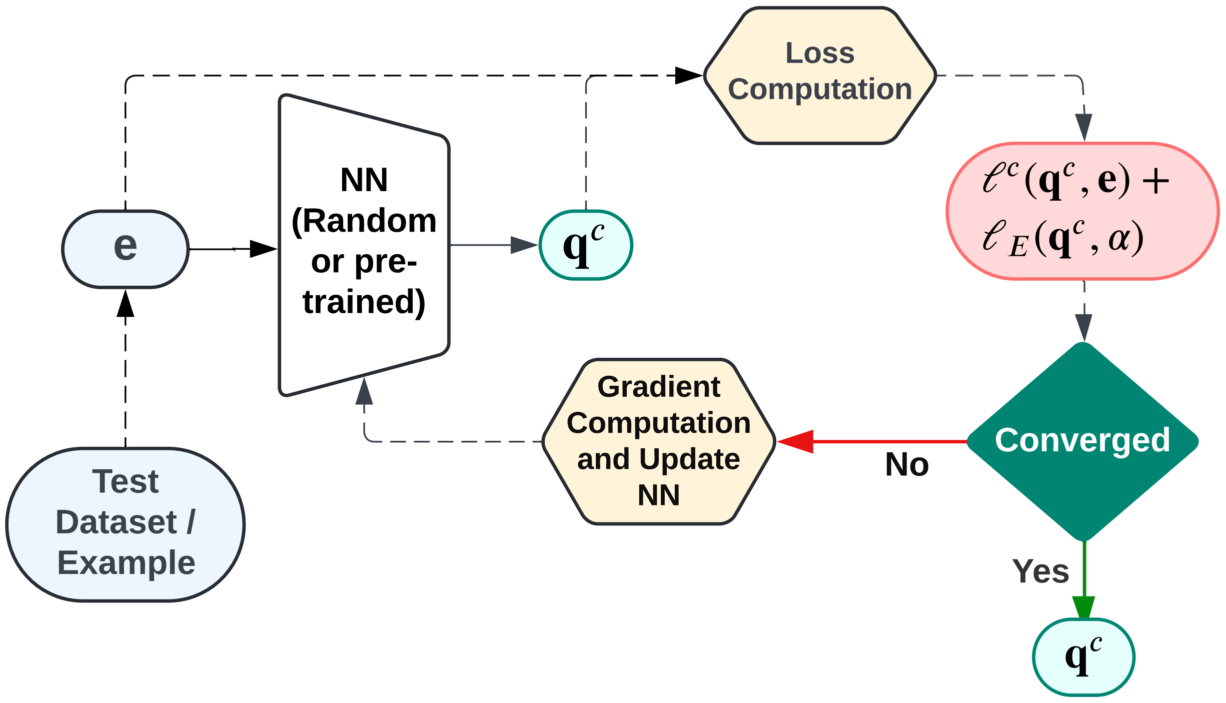

Loss Computation
$\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$
NN (Random or pre-trained)
e
$\mathbf{q}^c$
Gradient Computation and Update NN
No
Converged
Yes
Test Dataset / Example
$\mathbf{q}^c$

Figure 4.2: This figure illustrates the ITSELF procedure, which extends the training-time gradient-based optimization strategy to inference. Given a test dataset or example, the neural network (NN) is initialized either randomly or with a pre-trained model. During inference, the model undergoes iterative gradient updates using the self-supervised loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$. This optimization continues until convergence, ensuring that the inferred Most Probable Explanation (MPE) solution is refined over iterations.

4.2.3 Inference-Time Neural Optimization Using Self-Supervised Loss

At a high level, assuming that the NN is over-parameterized, if we use the self-supervised loss and repeatedly run (stochastic) gradient updates over the NN for a given dataset, theoretical results

(Allen-Zhu et al., 2019; Du et al., 2019) as well as prior experimental work (Zhang et al., 2017; Qin et al., 2024) suggest that the parameters of the NN will converge to a point near the global minimum of the self-supervised loss function. This means that through gradient updates, the network will find a near-optimal MPE assignment for each training example. This strategy of performing gradient updates over the NN can also be used *during inference (test) time to iteratively improve the MPE solution*, thereby maximizing the benefits of self-supervision.

Specifically, at test time, given a test dataset (or example), we initialize the NN either randomly or using a pre-trained model and then run gradient-based updates over the NN iteratively until convergence. The gradient is computed w.r.t. the self-supervised loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$. We call the resulting algorithm ITSELF (<u>I</u>nference <u>T</u>ime Optimization using <u>SELF</u>-Supervised Loss), as detailed in Figure 4.2. The performance of ITSELF typically improves with each iteration until the loss converges.

Our proposed method, ITSELF, is closely related to test-time training approaches which are widely used to solve problems in deep learning (Sun et al., 2020; Liu et al., 2021; Zhu et al., 2021; Wang et al., 2021; Alet et al., 2021; Darestani et al., 2022; Osowiechi et al., 2022; Liu et al., 2022; Li et al., 2023; Hardt and Sun, 2024). Our method differs from these previous approaches in that the global minima of our proposed self-supervised loss correspond to the MPE solutions, provided that the penalty α is sufficiently large.

4.3 Supervised Knowledge Transfer from ITSELF

A drawback of our self-supervised loss function is that, unlike supervised loss functions such as binary cross entropy, it is a non-convex function of the NN outputs[2]. As a result, it has a significantly larger number of local minima compared to the supervised loss function, but also a potentially exponential number of global minima, because an MPE problem can have multiple optimal solutions

[2]Note that we are referring to convexity with respect to the outputs, not the parameters of the NN.

(Marinescu and Dechter, 2019), all of which have the same loss function value. Thus, optimizing and regularizing using the self-supervised loss is difficult compared to a supervised loss, especially when the number of training examples is large.

Moreover, our experiments show that large datasets necessitate large, over-parameterized neural networks (NNs) to achieve near-optimal MPE solutions for all examples. However, when the training data is limited and the NN is sufficiently over-parameterized, our preliminary findings, along with theoretical and empirical results from prior studies (Arora et al., 2019; Chizat and Bach, 2018; Jacot et al., 2018; Lee et al., 2020, 2019), suggest that the NN is more likely to approach the global optima. Specifically, with a reasonably sized NN and a small dataset, the algorithm ITSELF tends to yield near-optimal MPE solutions. A further challenge with ITSELF is that even for small datasets, achieving convergence from a random initialization requires numerous iterations of gradient descent, rendering the training process inefficient and slow.

4.3.1 Teacher-Student Strategy

To address these challenges (using small datasets with ITSELF; designing better initialization for it; and using non-convex loss functions for training), we propose a two-network teacher-student strategy (Hinton et al., 2015; Romero et al., 2015; Zagoruyko and Komodakis, 2017; Yim et al., 2017; Kim et al., 2018; Furlanello et al., 2018; Cho and Hariharan, 2019; Heo et al., 2019; Yang et al., 2019; Mirzadeh et al., 2020), where we have two networks with the same structure that are trained via mini-batch gradient updates. The teacher network is overfitted to the mini-batch using our self-supervised loss via the ITSELF algorithm, and the student network is subsequently trained with a supervised loss function such as binary cross entropy. By overfitting the teacher network via ITSELF on the mini-batch, we ensure that it finds near-optimal MPE assignments for all (unlabeled) examples in the mini-batch and eventually over the whole training dataset.

The student network then learns from the teacher's outputs, using them as soft labels in a supervised learning framework. This transfer of knowledge mitigates the optimization difficulties associated with the non-convex self-supervised loss, allowing the student network to achieve

Algorithm 1 GUided Iterative Dual LEarning with Self-supervised Teacher ($\mathcal{GUIDE}$)

```
1: Input: Training data $\mathcal{D}$, teacher $\mathcal{T}$ and student $\mathcal{S}$ having the same structure
2: Output: Trained student network $\mathcal{S}$
3:         ▷ Database $DB$ stores the best MPE assignment and loss value for each example in $\mathcal{D}$
4: Initialize: Randomly initialize $\mathcal{T}$, $\mathcal{S}$, and $DB$
5: for each epoch do
6:     Sample a mini-batch $\mathcal{D}'$ from $\mathcal{D}$
7:     Update the parameters of $\mathcal{T}$ using the algorithm ITSELF (self-supervised loss) with
       Dataset $\mathcal{D}'$
8:     for each example $\mathbf{e}_i$ in $\mathcal{D}'$ do
9:         Make a forward-pass over $\mathcal{T}$ to get an MPE assignment $\mathbf{q}_i$ for $\mathbf{e}_i$
10:        Update the entry in $DB$ for $\mathbf{e}_i$ with $\mathbf{q}_i$ if it has a lower loss value than the current entry
11:    end for
12:    Update the parameters of $\mathcal{S}$ using the mini-batch $\mathcal{D}'$ and labels from $DB$ and a supervised
       loss
13:    $\mathcal{T} \leftarrow \mathcal{S}$                                ▷ Initialize $\mathcal{T}$ with $\mathcal{S}$ for the next epoch
14: end for
```

faster convergence and better generalization with a more manageable model size. Additionally, this strategy reduces the need for severe over-parameterization and extensive training iterations for the teacher network because it is operating on a smaller dataset. It also helps achieve better initialization for ITSELF.

4.3.2 Training Procedure

Our proposed training procedure, which we call $\mathcal{GUIDE}$, is detailed in Algorithm 1. The algorithm trains a two-network system comprising a teacher network ($\mathcal{T}$) and a student network ($\mathcal{S}$) with the same structure. The goal is to train the student network using a combination of self-supervised and supervised learning strategies. The algorithm takes as input the training data $\mathcal{D}$, along with the teacher and student networks, $\mathcal{T}$ and $\mathcal{S}$, respectively and outputs a trained network $\mathcal{S}$. A database (DB) is utilized to store the best MPE assignment and corresponding loss value for each example in $\mathcal{D}$. The parameters of $\mathcal{T}$ and $\mathcal{S}$, and the entries in DB, are randomly initialized at the start.

In each epoch, a mini-batch $\mathcal{D}'$ is sampled from the training data $\mathcal{D}$. The parameters of the teacher network $\mathcal{T}$ are then updated using the ITSELF algorithm (which uses a self-supervised loss), applied to the mini-batch $\mathcal{D}'$ (the mini-batch helps address large data issues associated with ITSELF). For each example $\mathbf{e}_i$ in $\mathcal{D}'$, we perform a forward-pass over $\mathcal{T}$ to obtain an MPE assignment $\mathbf{q}_i$. The database DB is subsequently updated with $\mathbf{q}_i$ if it has a lower loss value than the current entry for $\mathbf{e}_i$.

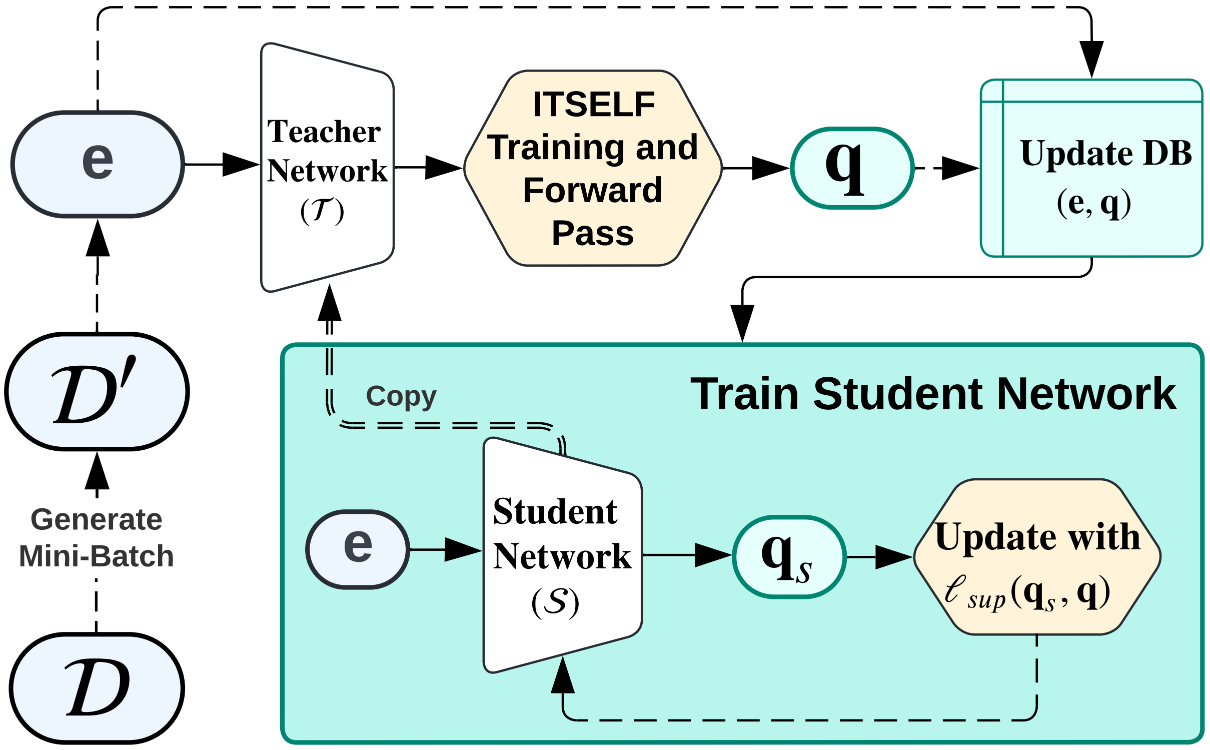

e
Teacher Network ($\mathcal{T}$)
ITSELF Training and Forward Pass
q
Update DB (e, q)
$\mathcal{D}'$
Generate Mini-Batch
$\mathcal{D}$
Copy
Train Student Network
e
Student Network ($\mathcal{S}$)
$\mathbf{q}_s$
Update with $\ell_{sup}(\mathbf{q}_s, \mathbf{q})$

Figure 4.3: This figure illustrates a single training epoch of our proposed method, $\mathcal{GUIDE}$, which utilizes a two-network system comprising a teacher network ($\mathcal{T}$) and a student network ($\mathcal{S}$), both sharing the same architecture. Each epoch begins by sampling a mini-batch $\mathcal{D}'$ from the training data $\mathcal{D}$. The teacher network $\mathcal{T}$ undergoes training using the ITSELF algorithm, optimizing its parameters based on self-supervised loss. For each example $\mathbf{e}_i$ in $\mathcal{D}'$, $\mathcal{T}$ performs a forward pass to generate an MPE assignment $\mathbf{q}_i$. The database DB is updated with $\mathbf{q}_i$ if it achieves a lower loss than the existing entry. The student network $\mathcal{S}$ is then trained using $\mathcal{D}'$ and the stored labels from DB, optimizing its parameters via a supervised loss function $\ell_{sup}(\mathbf{q}_s, \mathbf{q})$. At the end of the epoch, the parameters of the teacher network $\mathcal{T}$ are reinitialized with those of $\mathcal{S}$ to prepare for the next iteration. At inference time, the trained student network $\mathcal{S}$ can be directly used or serve as an initialization for further ITSELF optimization.

Following this, the parameters of the student network $\mathcal{S}$ are updated using the mini-batch $\mathcal{D}'$, the labels from DB, and a supervised loss function (ℓ_{sup}) such as Binary Cross Entropy or $L2$ loss.

Finally, the parameters of the teacher network $\mathcal{T}$ are reinitialized with the updated parameters of the student network $\mathcal{S}$ to prepare for the next epoch (addressing the initialization issue associated with ITSELF). Figure 4.3 illustrates a single training epoch of GUIDE.

Thus, at a high level, Algorithm 1 leverages the strengths of both self-supervised and supervised learning to improve training efficiency and reduce the model complexity, yielding a student network $\mathcal{S}$. Moreover, at test time, the student network can serve as an initialization for ITSELF.

4.4 Extending the Current Approach to Other Data Types and Inference Tasks

The current approach can be extended to support both multi-valued discrete and continuous variables, broadening its utility in diverse scenarios.

For multi-valued discrete variables, the method can be adapted by implementing a multi-class, multi-output classification head. Each query variable is represented by a softmax output node, which provides soft evidence by generating probabilistic distributions across multiple discrete values.

To incorporate continuous variables, we introduce a linear activation function in the output layer. The loss function, specifically the multi-linear representation of the PM, is modified to accommodate continuous neural network outputs. For example, in Probabilistic Circuits that use Gaussian distributions, continuous values can be directly integrated into the loss function, facilitating gradient-based backpropagation.

These extensions primarily involve adjusting the network's output layer and refining the self-supervised loss function represented by the PM. Notably, other elements of our approach, including the ITSELF and GUIDE procedures, remain unchanged.

Our approach further extends to additional inference tasks over probabilistic models, including marginal MAP and constrained most probable explanation (CMPE) tasks. However, the scalability of this approach depends on the computational efficiency of evaluating the loss function for each inference task. When this evaluation becomes computationally infeasible, the proposed method—training a neural network to answer queries over probabilistic models—may itself become infeasible. For example, performing marginal MAP inference over NAMs and PGMs requires repeated evaluations of the loss function associated with the marginal MAP task and its gradient during training. This iterative process, essential for updating the neural network's parameters, can become prohibitively resource-intensive due to the high computational demands of evaluating the marginal MAP loss over these probabilistic models.

4.5 Experiments

This section evaluates the ITSELF method (see Section 4.2.3), the $\mathcal{GUIDE}$ teacher-student training method (see Section 4.3) and the method that uses only self-supervised training, which we call SSMP (see Section 4.2.2 and Chapter 3). We benchmark these against various baselines, including neural network-based and traditional polynomial-time algorithms that directly operate on the probabilistic model. We begin by detailing our experimental framework, including competing methods, evaluation metrics, neural network architectures, and datasets.

4.5.1 Datasets and Graphical Models

We used twenty binary datasets extensively used in tractable probabilistic models literature (Lowd and Davis, 2010; Haaren and Davis, 2012a; Larochelle and Murray, 2011; Bekker et al., 2015)—referred to as TPM datasets—for evaluating PCs and NAMs. For the purpose of evaluating PGMs, we utilized high treewidth models from previous UAI inference competitions (Elidan and Globerson, 2010).

Table 4.1 summarizes the datasets and the probabilistic circuits trained on them. We use the same datasets for both PCs and NAMs. The selection includes both smaller datasets, such as NLTCS and MSNBC, and larger datasets with over 1000 variables.

For Markov networks, we utilize high treewidth grid networks, specifically grid40x40.f2.wrap, grid40x40.f5.wrap, grid40x40.f10.wrap, and grid40x40.f15.wrap. Each model contains 4800 variables and 1600 factors.

Table 4.1: Summary of datasets used with their respective numbers of variables and nodes in probabilistic circuits.

Dataset	Number of Variables	Number of Nodes in PC
NLTCS	16	125
MSNBC	17	46
KDDCup2k	64	274
Plants	69	3737
Audio	100	348
Jester	100	274
Netflix	100	400
Accidents	111	1178
Mushrooms	112	902
Connect 4	126	2128
Retail	135	359
RCV-1	150	519
DNA	180	1855
Book	500	1628
WebKB	839	3154
Reuters-52	889	7348
20 NewsGroup	910	2467
Movie reviews	1001	2567
BBC	1058	3399
Ad	1556	9666

To train Sum Product Networks (SPNs) (Poon and Domingos, 2011), our choice of PCs, we employed the DeeProb-kit library (Loconte and Gala, 2022), with SPN sizes ranging from 46 to 9666 nodes. For NAMs, we trained Masked Autoencoder for Distribution Estimation (MADE) models using PyTorch, following the approach in Germain et al. (2015). For Markov Networks (MNs), a specific type of PGM, we applied Gibbs sampling to generate 8,000, 1,000, and 1,000 samples for the training, testing, and validation sets, respectively. The query ratio (qr), defined as the fraction of variables in the query set, was varied across $\{0.1, 0.3, 0.5, 0.7, 0.8, 0.9\}$ for each probabilistic model (PM).

4.5.2 Baseline Methods and Evaluation Criteria

PCs - We used three polynomial-time baseline methods from the probabilistic circuits and probabilistic graphical models literature as benchmarks (Park and Darwiche, 2004; Poon and Domingos, 2011).

- **MAX Approximation (MAX)** (Poon and Domingos, 2011) transforms sum nodes into max nodes. During the upward pass, max nodes output the highest weighted value from their children. The downward pass, starting from the root, selects the child with the highest value at each max node and includes all children of product nodes.
- **Maximum Likelihood Approximation (ML)** (Park and Darwiche, 2004) computes the marginal distribution $p_{\mathcal{M}}(Q_i|\mathrm{e})$ for each variable $Q_i \in \mathbf{Q}$, setting Q_i to its most likely value.
- **Sequential Approximation (Seq)** (Park and Darwiche, 2004) iteratively assigns query variables according to an order o. At each step j, it selects the j-th query variable Q_j in o and assigns to it a value q_j such that $\mathrm{p}_{\mathcal{M}}(q_j|\mathrm{e}, \mathbf{y})$ is maximized, where $\mathbf{y}$ is an assignment of values to all query variables from 1 to $j-1$.

We further evaluated the impact of initializing stochastic hill climbing searches using solutions from all baseline approaches and our proposed methods for MPE inference, conducting 60-second searches for each MPE problem in our experiments, as detailed in Park and Darwiche (2004).

NAMs - As a baseline, we used the stochastic hill-climbing search (HC) algorithm. Following a procedure similar to that used for PCs, we conducted a 60-second hill-climbing search for each test example, with query variables initialized randomly and setting evidence variables according to the values in the given example.

PGMs - We employed the AOBB method (Otten and Dechter, 2012) as a baseline, using the implementation outlined in Otten (2012). Since AOBB is an anytime algorithm, we set a 60-second time limit for inference per test example.

Neural Baselines - Arya et al. (2024b) introduced Self-Supervised learning based MMAP solver for PCs (SSMP), training a neural network to handle queries on a fixed variable partition within PCs. We extend this approach to address the any-MPE task in PMs (see Section 4.2.2), using a single network to answer any-MPE queries as an additional neural baseline.

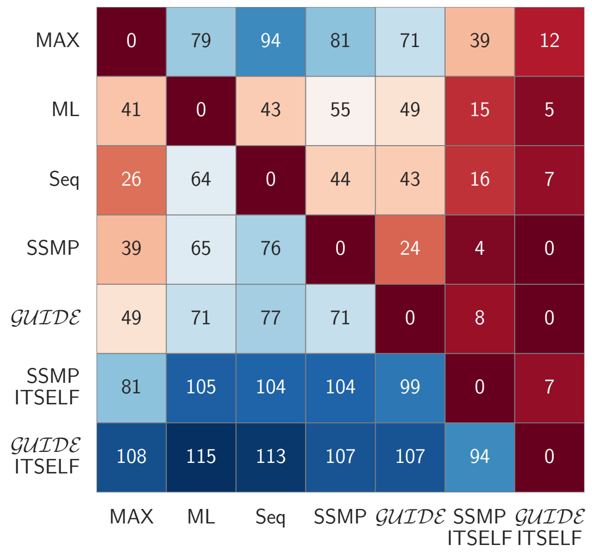
Evaluation Criteria - We evaluated competing approaches based on log-likelihood (LL) scores calculated as $\ln p_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$, and inference times for given evidence $\mathbf{e}$ and query output $\mathbf{q}$. Higher log-likelihood scores indicate better performance, while shorter inference times are preferable.

4.5.3 Neural Network-Based Approaches

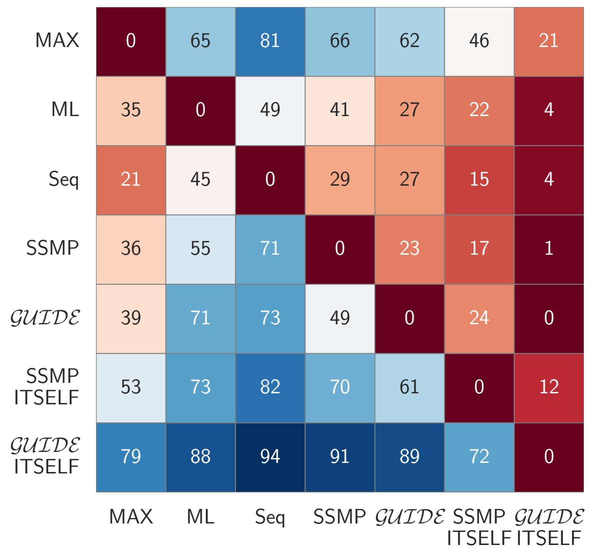
We implemented two neural network training protocols for each PM and query ratio: SSMP and $\mathcal{GUIDE}$. Each model was trained for 20 epochs following the training procedure outlined by Arya et al. (2024b) for SSMP. Both protocols employed two distinct inference strategies, thus forming four neural-based variants. In the first strategy, we performed a single forward pass through the network to estimate the values of query variable, as specified by Arya et al. (2024b). The second strategy utilized our novel test-time optimization-based ITSELF approach for inference. The ITSELF optimization terminates after 100 iterations or upon loss convergence for both PCs and PGMs. For NAMs, we increase the limit to 1,000 iterations while keeping the convergence criterion.

We standardized network architectures for PMs across all experiments. For PCs, we used fully connected Neural Networks (NN) with three hidden layers (128, 256, 512 nodes). For NAMs and PGMs, a single hidden layer of 512 nodes was employed. All hidden layers featured ReLU activation, while the output layers used sigmoid functions with dropout for regularization (Srivastava et al., 2014). We optimized all models using Adam (Kingma and Ba, 2015) and implemented them in PyTorch (Paszke et al., 2019) on an NVIDIA A40 GPU.

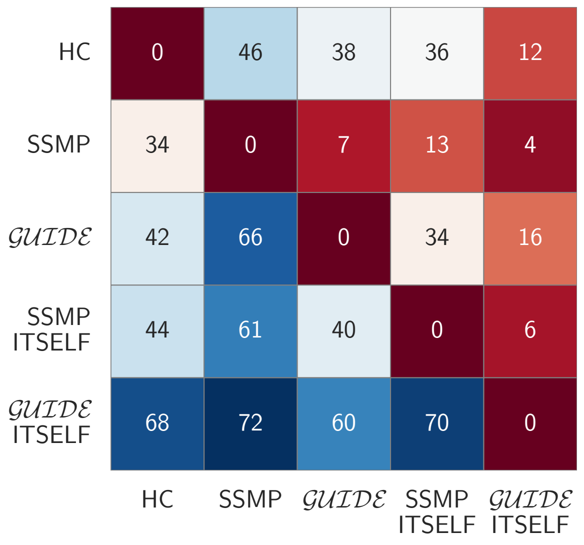
For neural network-based solvers, the mini-batch size was set to 512 samples, and a learning rate decay strategy, reducing the rate by 0.9 upon loss plateauing, was implemented to improve training efficiency. Optimal hyperparameters were identified via extensive 5-fold cross-validation.



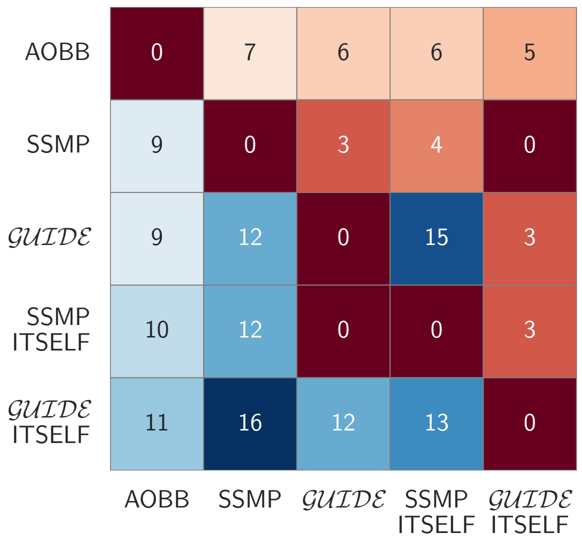
(a) PCs: Initial Solutions



(b) PCs: Hill-Climbing



(c) NAMs



(d) PGMs

Figure 4.4: MPE method comparison across PMs. Blue shows row superiority, red shows column superiority; darker shades indicate larger values.

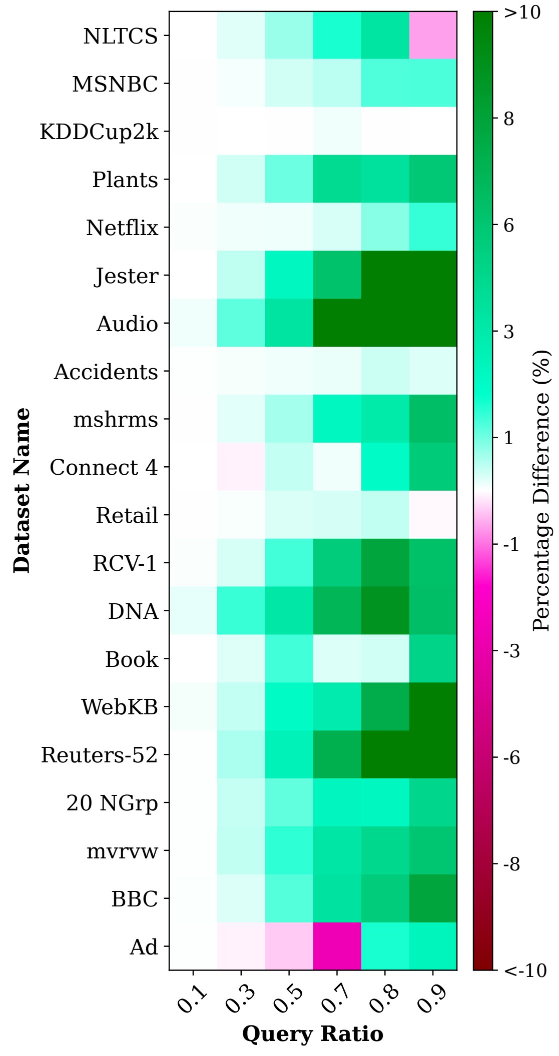
In discrete loss scenarios, the hyperparameter α played a pivotal role. We systematically explored the optimal α value across the range $\{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$ for neural-based models, including ITSELF and *GUIDE*. Notably, higher α values better constrain outputs to binary, thereby facilitating near-optimal results.

Results for PCs: We compare methods—including three polynomial-time baselines, neural network-based SSMP, and our ITSELF and $\mathcal{GUIDE}$ methods—on 20 TPM datasets as shown in the contingency table in Figure 4.4a. We generated 120 test datasets for the MPE task using 20 PCs across 6 query ratios (qr). Each cell (i, j) in the table represents how often (out of 120) the method in row i outperformed the method in column j based on average log-likelihood scores. Any difference between 120 and the combined frequencies of cells (i, j) and (j, i) indicates cases where the compared methods achieved similar scores. We present similar contingency tables for Hill Climbing Search over PCs (Figure 4.4b), NAMs (Figure 4.4c), and PGMs (Figure 4.4d) to benchmark the proposed methods against the baselines.

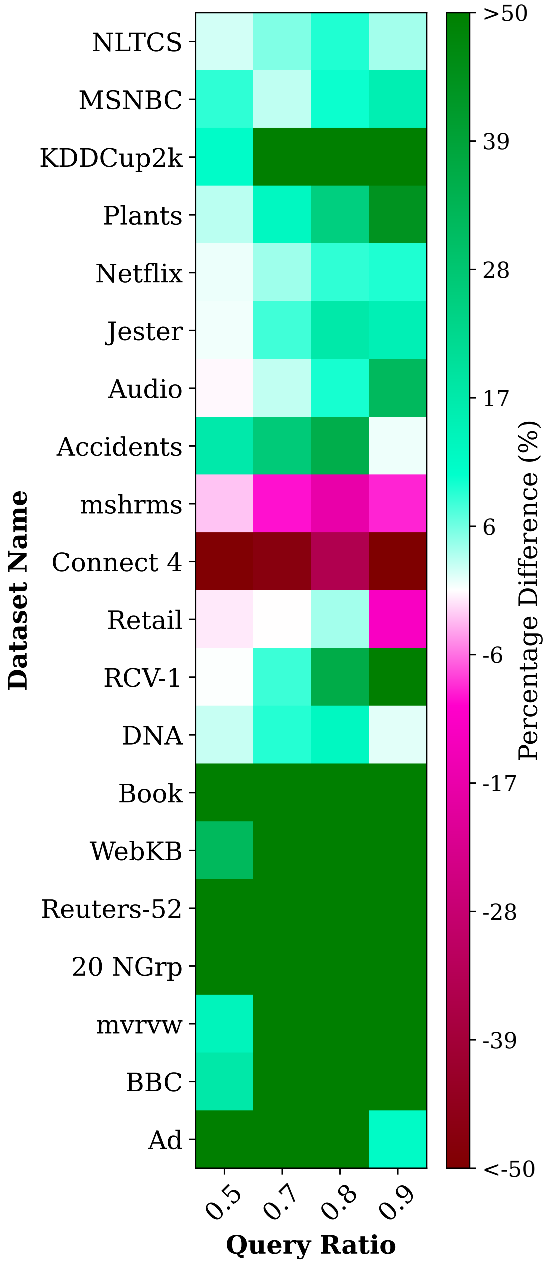
The contingency table for PC (Figure 4.4a) shows that methods incorporating ITSELF consistently outperform both polynomial-time and traditional neural baselines, as indicated by the dark blue cells in the corresponding rows. Notably, $\mathcal{GUIDE}$ + ITSELF is superior to all the other methods in almost two-thirds of the 120 cases, while SSMP + ITSELF is better than both SSMP and $\mathcal{GUIDE}$. In contrast, the polynomial-time baseline MAX is better than both SSMP and $\mathcal{GUIDE}$ (as used in Arya et al. (2024b)), highlighting ITSELF's significant role in boosting model performance for the complex *any-MPE* task.

We compare MAX and $\mathcal{GUIDE}$ + ITSELF using a heatmap in Figure 4.5a. The y-axis presents datasets by variable count and the x-axis represents query ratio. Each cell displays the percentage difference in mean LL scores between the methods, calculated as $\%\text{Diff.} = 100 \times (ll_{nn} - ll_{max})/|ll_{max}|$. The heatmap shows that $\mathcal{GUIDE}$ + ITSELF achieves performance comparable to MAX for small query sets. As the problem complexity increases with an increase in query set size, our method consistently outperforms MAX across all datasets, except for NLTCS and Tretail, as highlighted by the green cells. In the 12 cases where $\mathcal{GUIDE}$ + ITSELF underperforms, the performance gap remains minimal, as indicated by the limited number of red cells in the heatmap.

Figure 4.4b further analyzes the performance of our proposed methods against various baselines as initialization strategies for Hill Climbing Search. This comparison evaluates the effectiveness of ITSELF and $\mathcal{GUIDE}$ in enhancing *anytime methods* compared to conventional heuristic



(a) PC: $\mathcal{GUIDE}$ + ITSELF vs. MAX



(b) NAM: $\mathcal{GUIDE}$ + ITSELF vs. HC

Figure 4.5: Heatmaps showing LL % Differences. Left: PC; Right: NAM. Green cells: our method is better. Darker shades indicate larger values.

initialization approaches. Notably, methods incorporating ITSELF provide superior initialization for local search-based algorithms.

Results for NAMs: The contingency table in Figure 4.4c presents our evaluation of several methods for NAMs, including HC and two neural network approaches, SSMP and $\mathcal{GUIDE}$, each tested with two inference schemes. We evaluated these methods on 20 TPM datasets, creating 80 test sets for the MPE task using 20 MADEs across four query ratios (qr).

The $\mathcal{GUIDE}$ + ITSELF approach demonstrates superior performance compared to both baseline methods and other neural inference schemes, aligning with observations from PC. While HC outperforms SSMP, both $\mathcal{GUIDE}$ and the combination of SSMP-based training with ITSELF-based inference surpass HC, highlighting their advantages over the baseline.

The heatmaps in Figure 4.5b further highlight the superior performance of $\mathcal{GUIDE}$ + ITSELF for NAMs, particularly in larger datasets where it outperforms the HC baseline by over 50% in most cases, as indicated by the dark green cells. The combination of $\mathcal{GUIDE}$-based learning with ITSELF-based inference consistently outperforms the baseline across most datasets, with exceptions only in the Mushrooms, Connect 4, and Retail. Overall, the $\mathcal{GUIDE}$ + ITSELF approach significantly enhances the quality of the MPE solutions in NAM models.

Results for PGMs: The contingency table in 4.4d compares the performance of AOBB and four neural-network-based methods on PGMs across four high-treewidth networks. For this evaluation, we generated 16 test datasets for the MPE task using four PGMs across four query ratios (qr).

Consistent with results from previous PMs, methods using ITSELF for inference consistently outperform the baseline methods AOBB and SSMP across most scenarios. Both $\mathcal{GUIDE}$ and SSMP outperform AOBB in at least 50 percent of the tests. Later, we also presents comparisons against exact solutions, conducted on less complex probabilistic models where ground truth computation remains tractable.

Does the teacher-student-based network outperform a single network trained with the self-supervised loss? ($\mathcal{GUIDE}$ vs. SSMP):

This analysis aims to evaluate the performance of $\mathcal{GUIDE}$ against traditional neural network training methods used in SSMP across different PMs and inference schemes. Using traditional inference scheme (i.e., one forward pass through the network), $\mathcal{GUIDE}$ consistently outperforms SSMP, demonstrating its superiority in 60% of scenarios for PCs, more than 80% for NAM models, and 75% for PGM models. When employing ITSELF-based inference, $\mathcal{GUIDE}$ maintains this advantage, achieving higher quality solutions in more than 75%, 85%, and 80% of cases for PCs, NAMs, and PGMs, respectively. Therefore, models trained using $\mathcal{GUIDE}$ are consistently superior to those trained with SSMP for the *any-MPE* task.

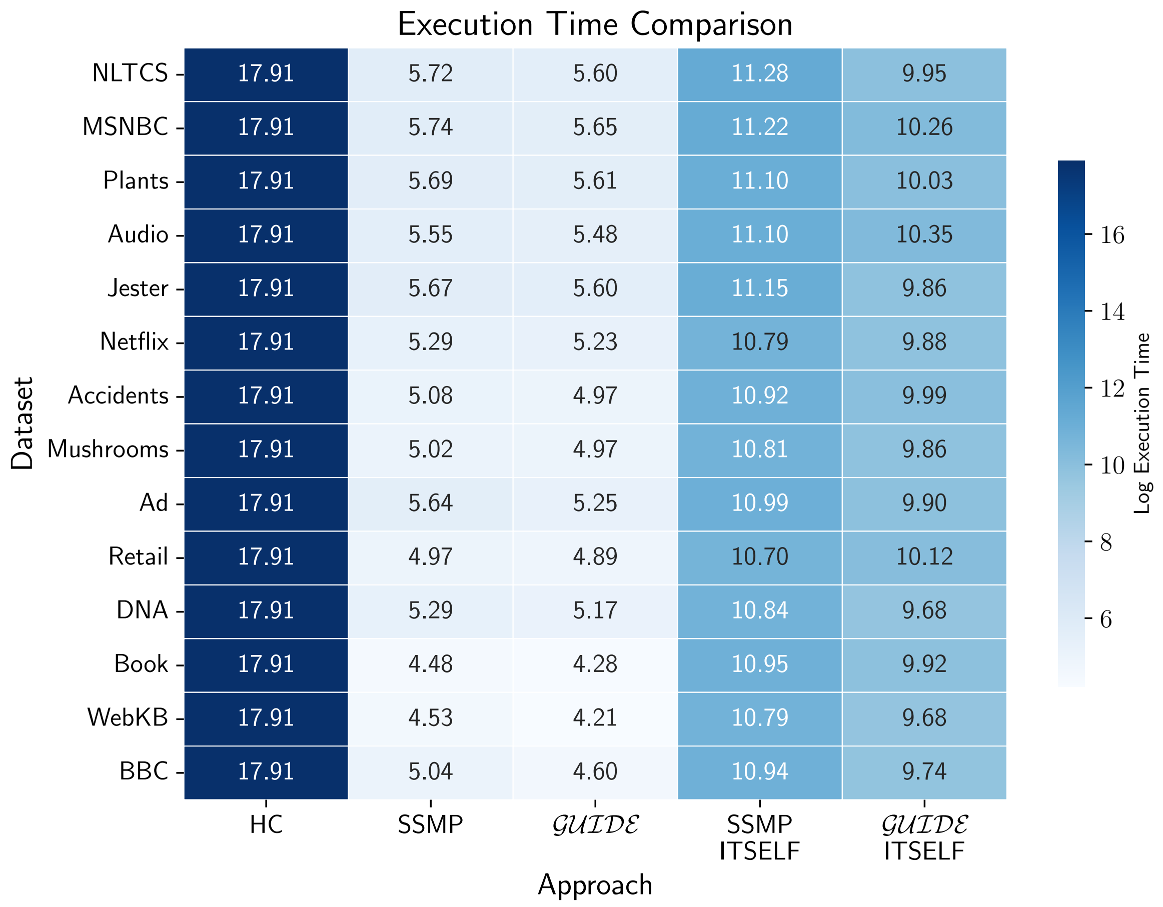
Does inference time optimization improve performance? (One-Pass vs. Multi-Pass):

In this analysis, we compare the performance of the single-pass inference method to that of the proposed multi-pass inference method (ITSELF). ITSELF combined with SSMP training outperforms the other methods in over 85% cases for PC, and more than 75% for NAM and PGM models. When used on models trained with $\mathcal{GUIDE}$, ITSELF demonstrates even better results, achieving superior performance in nearly 90% of PC cases and 75% for both NAMs and PGMs. Overall, $\mathcal{GUIDE}$ with ITSELF inference emerges as the most effective method across all experiments. Empirical evidence consistently demonstrates ITSELF's superiority over single-pass inference across PMs.

4.6 Inference Time Comparison

We present the inference times for all baselines and proposed methods in Figures 4.6 to 4.8. Figure 4.6 details the inference times for MADE, while Figures 4.7 and 4.8 respectively illustrate the times for PCs and PGMs. This comparison facilitates a direct evaluation of the computational efficiency across different models.

Each cell displays the natural logarithm of the time, measured in microseconds, for each method and dataset. Lighter colors indicate lower values. Notably, inferences using SSMP and $\mathcal{GUIDE}$ require the shortest time, as these methods necessitate only a single forward pass through the neural network to obtain the values for the query variables.



Dataset	HC	SSMP	GUIDE	SSMP ITSELF	GUIDE ITSELF
NLTCS	17.91	5.72	5.60	11.28	9.95
MSNBC	17.91	5.74	5.65	11.22	10.26
Plants	17.91	5.69	5.61	11.10	10.03
Audio	17.91	5.55	5.48	11.10	10.35
Jester	17.91	5.67	5.60	11.15	9.86
Netflix	17.91	5.29	5.23	10.79	9.88
Accidents	17.91	5.08	4.97	10.92	9.99
Mushrooms	17.91	5.02	4.97	10.81	9.86
Ad	17.91	5.64	5.25	10.99	9.90
Retail	17.91	4.97	4.89	10.70	10.12
DNA	17.91	5.29	5.17	10.84	9.68
Book	17.91	4.48	4.28	10.95	9.92
WebKB	17.91	4.53	4.21	10.79	9.68
BBC	17.91	5.04	4.60	10.94	9.74

Figure 4.6: Heatmap depicting the inference time for MADE on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

For MADE, the subsequent fastest method employs a model trained with $\mathcal{GUIDE}$ and conducts inference using ITSELF, outperforming the approach that uses SSMP for training. This advantage stems from the reduced number of ITSELF iterations required by $\mathcal{GUIDE}$, benefiting from a more effectively trained model. In PGMs, a similar pattern emerges with $\mathcal{GUIDE}$ + ITSELF as the next fastest method, followed by SSMP + ITSELF. For PCs, MAX ranks as the next fastest, closely followed by the $\mathcal{GUIDE}$ + ITSELF and SSMP + ITSELF methods. Finally, the ML and Seq methods display the highest inference times.

Thus, if you require a highly efficient method capable of performing inference in a fraction of a millisecond, $\mathcal{GUIDE}$ is the optimal choice. It outperforms the baseline for both MADE and PGMs.

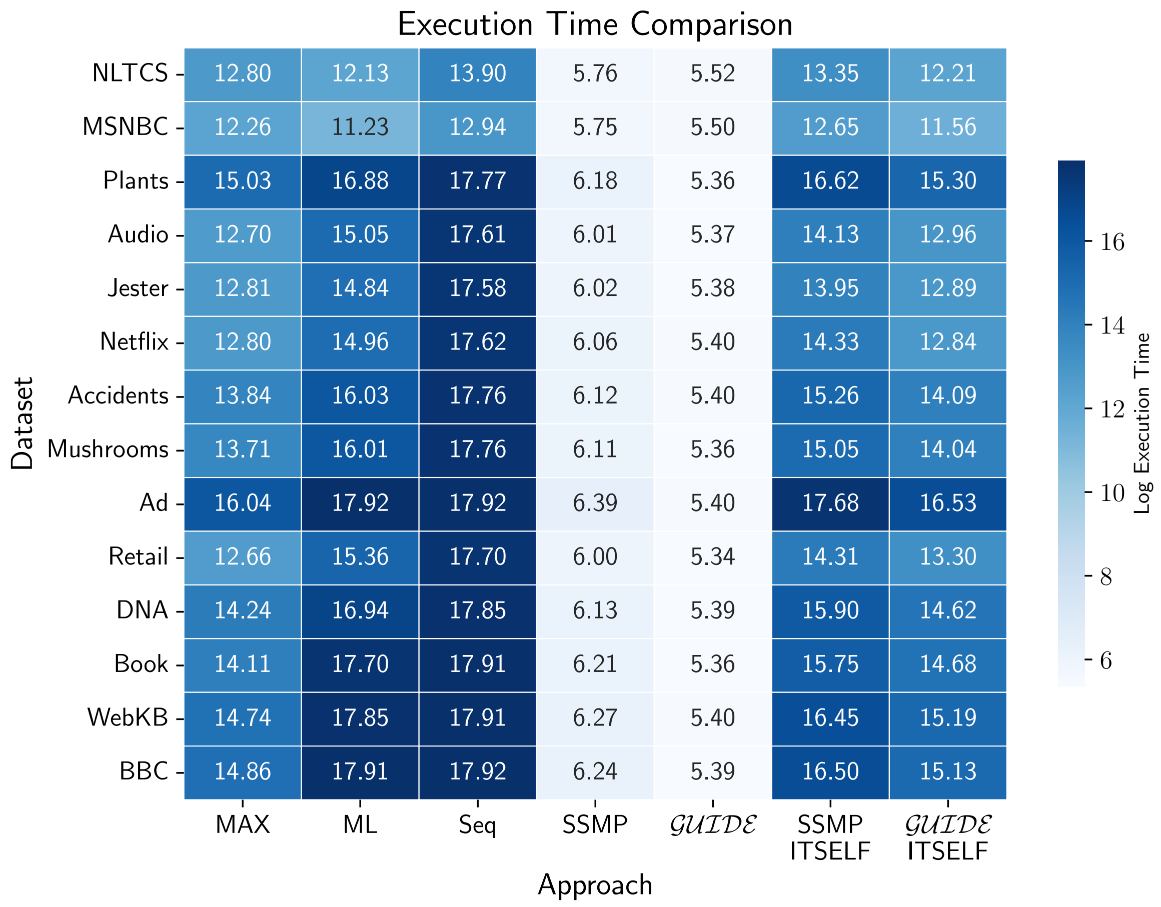


Figure 4.7: Heatmap depicting the inference time for PC on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

However, if higher log-likelihood scores are necessary, $\mathcal{GUIDE}$ + ITSELF would be suitable, as it generally surpasses the baselines in speed and performance across various scenarios.

4.7 Gap Analysis For PGM

Table 4.2 presents the log-likelihood score gap between the neural network methods (SSMP, $\mathcal{GUIDE}$, SSMP + ITSELF, $\mathcal{GUIDE}$ + ITSELF) and exact solutions. These exact solutions are obtained using AOBB, which provides near-optimal results for smaller datasets. For each approach M, the gap is calculated as the relative difference between the score of the near-optimal

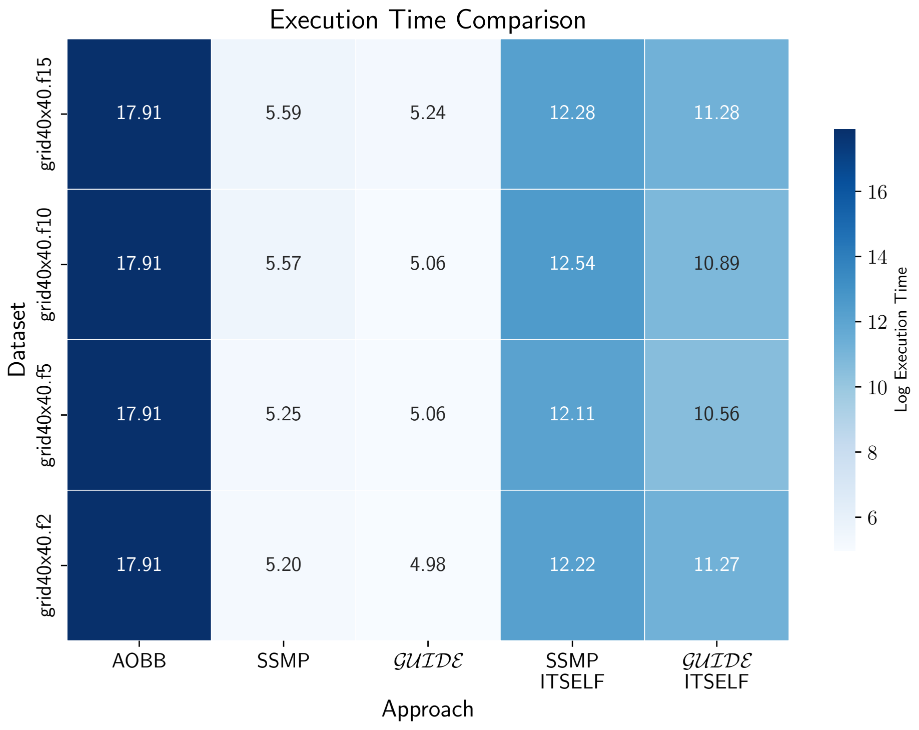


Figure 4.8: Heatmap depicting the inference time for PGM on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

solution (determined by AOBB) and the score achieved by M. This approach is feasible due to the use of small datasets, allowing identification of exact solutions.

The final column highlights the neural-based approach achieving the best performance for each dataset and query ratio combination. Notably, $\mathcal{GUIDE}$ and ITSELF consistently surpass other neural baselines across almost all dataset-query pairs. This analysis provides a comprehensive assessment of the proposed methods relative to exact solutions on small datasets, enabling a direct comparison of their effectiveness.

Summary: Our experiments demonstrate that $\mathcal{GUIDE}$ + ITSELF outperforms both polynomial-time and neural-based baselines across various PMs, as evidenced by higher log-likelihood scores. Notably, ITSELF demonstrates significant advantages over traditional single-pass inference in addressing the complex *any-MPE* query task within probabilistic models, emphasizing the importance of Inference Time Optimization. Furthermore, the superior performance of models

Table 4.2: Gap Between AOBB And Other Methods.

Method	**Query Ratio**	**SSMP**	$\mathcal{GUIDE}$	**SSMP + ITSELF**	$\mathcal{GUIDE}$ **+ ITSELF**	**Best Method**
Grids-17	0.900	0.082	0.060	0.069	0.075	$\mathcal{GUIDE}$
Grids-17	0.800	0.051	0.038	0.040	0.035	$\mathcal{GUIDE}$ + ITSELF
Grids-17	0.700	0.042	0.030	0.034	0.016	$\mathcal{GUIDE}$ + ITSELF
Grids-17	0.500	0.026	0.024	0.024	0.007	$\mathcal{GUIDE}$ + ITSELF
Grids-18	0.900	0.081	0.062	0.071	0.102	$\mathcal{GUIDE}$
Grids-18	0.700	0.033	0.027	0.024	0.015	$\mathcal{GUIDE}$ + ITSELF
Grids-18	0.500	0.020	0.018	0.018	0.006	$\mathcal{GUIDE}$ + ITSELF
Grids-18	0.800	0.054	0.035	0.045	0.037	$\mathcal{GUIDE}$
Segmentation-14	0.500	0.032	0.032	0.032	0.004	$\mathcal{GUIDE}$ + ITSELF
Segmentation-14	0.900	0.045	0.014	0.014	0.005	$\mathcal{GUIDE}$ + ITSELF
Segmentation-14	0.800	0.051	0.024	0.024	0.006	$\mathcal{GUIDE}$ + ITSELF
Segmentation-14	0.700	0.029	0.029	0.029	0.005	$\mathcal{GUIDE}$ + ITSELF
Segmentation-15	0.800	0.046	0.002	0.002	0.002	$\mathcal{GUIDE}$ + ITSELF
Segmentation-15	0.500	0.003	0.003	0.003	0.000	$\mathcal{GUIDE}$ + ITSELF
Segmentation-15	0.900	0.675	0.255	0.433	0.305	$\mathcal{GUIDE}$
Segmentation-15	0.700	0.003	0.003	0.003	0.002	$\mathcal{GUIDE}$ + ITSELF

trained with $\mathcal{GUIDE}$ compared to SSMP highlights the effectiveness of the dual network approach, which improves initial model quality and establishes an optimal starting point for ITSELF.

4.8 Chapter Summary

In this chapter, we introduced novel methods for answering arbitrary Most Probable Explanation (MPE) queries in probabilistic models. Our approach employs self-supervised loss functions to represent MPE objectives, enabling tractable loss and gradient computations during neural network training. We also proposed a new inference time optimization technique, ITSELF, which iteratively improves the solution to the MPE problem via gradient updates. Additionally, we introduced a dual-network-based strategy that combines supervised and unsupervised training which we call $\mathcal{GUIDE}$ to provide better initialization for ITSELF and addressing various challenges associated with self-supervised training. Our method was tested on various benchmarks, including probabilistic circuits, neural autoregressive models, and probabilistic graphical models, using 20 binary datasets and high tree-width networks. It outperformed polytime baselines and other neural methods, substantially in some cases. Additionally, it improved the effectiveness of stochastic hill climbing (local) search strategies.

CHAPTER 5
SINE: SCALABLE MPE INFERENCE FOR PGMS USING ADVANCED NEURAL EMBEDDINGS

5.1 Introduction

Probabilistic graphical models (PGMs) (Koller and Friedman, 2009), including Bayesian Networks (BNs) and Markov Networks (MNs), are widely used to model large, multi-dimensional probability distributions. However, as the complexity of these distributions grows, solving NP-hard inference tasks—such as determining the MPE through exact inference (Otten, 2012; Otten and Dechter, 2012)—becomes computationally infeasible, which limits the scalability of these models in answering complex probabilistic queries. Although several approximate solvers have been developed for the MPE task, they often fail to achieve the accuracy required for real-world applications.

Recent advancements in neural network-based approximate solvers have addressed limitations in existing methods for inference in PGMs. Arya et al. (2024c) introduced a novel technique that combines inference-time optimization with a self-supervised teacher-student framework to answer MPE queries across various probabilistic models. This technique builds on the earlier work of Arya et al. (2024b), which targeted both MPE and MMAP queries in probabilistic circuits (Choi et al., 2020a). By employing a self-supervised loss function, these methods eliminate the need for exact solutions during training. Drawing on the literature of learning to optimize (Li and Malik, 2017; Fioretto et al., 2020; Donti et al., 2020; Zamzam and Baker, 2020; Park and Hentenryck, 2023), they leverage neural models to enable efficient probabilistic reasoning. Furthermore, these neural-based MPE solvers provide two key advantages: they deliver superior solution quality while reducing inference time compared to traditional solvers.

In this chapter, we focus on advancing two key aspects of neural network-based models for answering MPE queries over PGMs: improving the quality of input embeddings for inference tasks and developing improved methods for discretization of the neural network's continuous outputs to obtain MPE solutions. Existing approaches typically rely on encoding schemes that focus solely on the probabilistic query, failing to fully exploit the rich information embedded in the PGM. Moreover, these methods often use thresholding for discretization, which only considers the nearest binary solution and may fail to identify the optimal MPE due to the non-linearity of the self-supervised loss function. As a result, these techniques are limited to solving simpler problems where the model learns to answer a single predefined query (Arya et al., 2024b), or they necessitate test-time optimization to achieve near-optimal solutions for arbitrary queries (Arya et al., 2024c).

In contrast, our approach handles arbitrary queries without requiring test-time optimization by embedding both the structure and parameters of PGMs into a hypergraph representation (Dechter, 2019). We employ neural message-passing algorithms within an attention-based Hypergraph Neural Network (HGNN) (Bai et al., 2019) to efficiently propagate information across the hypergraph. For initializing the HGNN embeddings, we utilize evidence-instantiated factors and initial evidence values, along with partition sets. This process produces more informative representations of the PGM, eliminating the need for additional methods to achieve near-optimal solutions.

To enhance the quality of MPE solutions, we further introduce novel discretization techniques to derive MPE solutions from continuous neural network outputs. The first method utilizes the neural network's outputs to identify variables with uncertain predictions. By augmenting the evidence with confidently predicted variables, the problem is simplified to a query that an oracle can efficiently solve. While neural networks excel at processing large queries quickly, they may exhibit inaccuracies due to the complexity of the loss function. In contrast, oracles—MPE solvers like those in Otten and Dechter (2012)—provide highly accurate solutions for smaller problems, though with slower performance on larger queries. By combining the strengths of both, this approach balances efficiency and accuracy in solving MPE queries.

The second method enhances solution quality by systematically exploring the neighborhood of binary solutions around the neural network's continuous output. Unlike thresholding, which evaluates only a single solution, this approach examines multiple nearby binary configurations.

By leveraging the neural network's predictions as a starting point and examining various nearby configurations, this method effectively navigates the solution space to yield superior solutions compared to the thresholding approach. It also reduces the number of hyper-parameters by removing the need to choose a specific threshold value, which can be arbitrary and may not generalize well across different datasets or problems.

We conducted an extensive empirical evaluation, comparing our proposed method against several non-neural and neural approaches. Our experiments, performed on a diverse set of benchmark models, demonstrate that neural networks trained on our HGNN embeddings and utilizing advanced discretization techniques significantly outperform existing models in both efficiency and accuracy.

Motivation: A key limitation of the approach presented by Arya et al. (2024b) is its reliance on a pre-defined partition of variables into query and evidence subsets. To address this, the authors later introduced the any-MPE task (see Arya et al. (2024c)). In this task, given a probabilistic graphical model (PGM), the neural network takes an assignment to an arbitrary subset of variables (evidence) as input and outputs the most likely assignments for the remaining (query) variables. They also proposed a neural network-based approach for solving the any-MPE task. The goal of this chapter is to address and resolve two main limitations of their approach.

First, Arya et al.'s approach relies exclusively on the query embedding (evidence values and the subsets of query and evidence variables), without leveraging the structure or parameters of the PGM. As a result, their method requires test-time optimization, with much of its performance gains attributed to this process. Second, their method uses a thresholding procedure to derive discrete solutions from the continuous outputs of the neural network. This process does not account for the network's confidence or ensure that the thresholded solution is the optimal discrete solution near the continuous value. Given the non-linearity of the loss function, the best solution might exist near the continuous output.

In the following sections, we introduce a novel technique that generates more *informative* embeddings of the MPE-input by incorporating with the query both the structure and parameters of the given PGM as well as a more reliable path to discrete optimal solutions.

5.2 An Advanced Encoding for any-MPE Queries in PGMs

In this section, we introduce an embedding based on a hypergraph representation to address the *any-MPE* task. Given a PGM and evidence—defined as an assignment of values to an arbitrary subset of variables—our goal is to construct an information-rich input encoding that captures the structure and parameters of the PGM and characterizes the subsets of variables representing both the query and evidence, along with the corresponding assignment to the evidence variables. This enriched encoding enhances generalization, reducing the reliance on expensive test-time optimization. Additionally, we propose an output encoding that aligns the neural network's output layer with the given PGM, enabling efficient recovery of the MPE solution from the network's predictions.

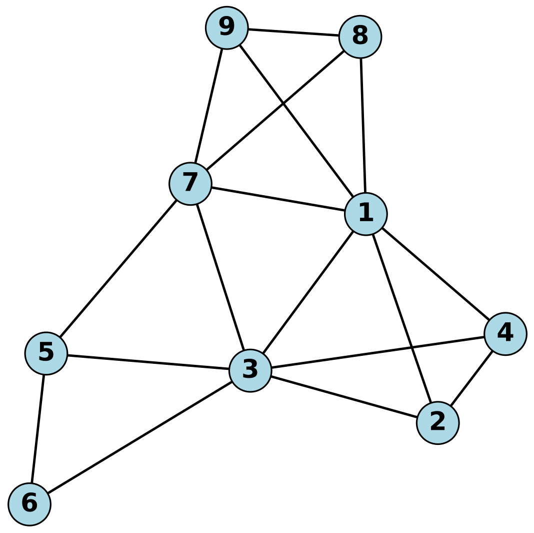
5.2.1 Integrating PGM Parameters Into the Encodings

Given an assignment to the evidence variables, a straightforward approach to utilize the parameters of the PGM into the input encoding is to instantiate the evidence in each factor/clique of the PGM and then concatenate the resulting instantiated factors. More formally, we apply the following transformation to each clique ϕ_k and each observed variable $E = e$:

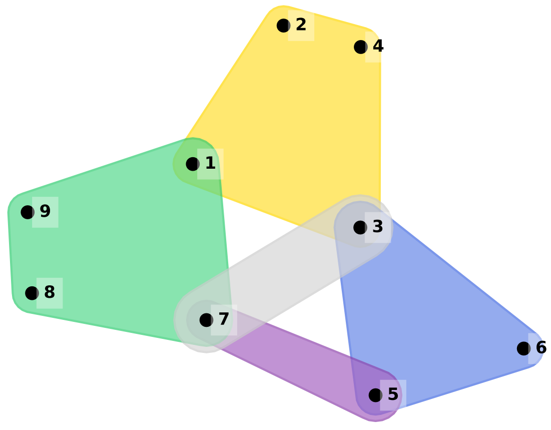
$$\phi_k(X_k) = \begin{cases} 0, & \text{if} \quad \mathbf{x_k}^E \neq e \\ \phi_k(X_k), & \text{if} \quad \mathbf{x_k}^E = e \end{cases}$$

A key property of the resulting encoding, which is an *injective* mapping, is that it maintains a consistent length across all possible evidence assignments, making the length independent of the any-MPE query. This consistency is beneficial because the neural network requires inputs of uniform size. However, this approach has two significant limitations: it fails to incorporate structural information from the PGM, as the concatenation order is arbitrary, and it does not facilitate message passing between cliques. To address these issues, we aim to enhance the expressiveness of these embeddings to achieve a more informative representation.

5.2.2 Hypergraph Based Encoding



(a) Graph-based representation. Nodes represent variables, and edges capture pairwise dependencies.



(b) Hypergraph-based representation. Hyperedges (colored regions) represent cliques in the PGM.

Figure 5.1: Comparison of PGM structure representations: (a) graph, and (b) hypergraph.

PGMs can be effectively represented using both hypergraphs (Dechter, 2019) and graphs, with a one-to-one correspondence between the nodes in these structures. Each factor in the PGM corresponds to a clique in the graph, which maps to a hyperedge in the hypergraph. Figure 5.1 provides an example of these two representations for a PGM with 9 nodes, comprising 2 cliques of size 4, 1 clique of size 3, and 2 pairwise cliques. Each clique represents a factor involving the associated variables in the PGM.

Building on this representation, we explore how a hypergraph framework enables the generation of richer embeddings for any-MPE queries. Utilizing Hypergraph Neural Networks (HGNNs) (Feng et al., 2019), we aggregate embeddings within the hypergraph, enhancing the expressiveness of the resulting representation. Notably, hypergraph-based neural networks exhibit superior performance in capturing complex, higher-order relationships compared to traditional graph-based neural models (Feng et al., 2019; Cai et al., 2022; Chen and Schwaller, 2024).

To utilize HGNNs, three key components are required: (1) A node feature matrix that represents each node's feature vector. (2) A sparse incidence matrix that encodes the connectivity between nodes and hyperedges through the hyperedge indices, (3) Hyperedge feature matrices that allow the model to incorporate edge-specific information.

The incidence matrix is derived directly from the PGM's clique structure. For hyperedge features, we use the embeddings introduced in Section 5.2.1. Instead of concatenating embeddings from all cliques to form a single representation, we keep them separate, enabling each embedding to serve as the hyperedge attribute for its corresponding clique. Node attributes consist of two values: the first represents the node's value (assigned as the observed value for evidence variables or set to -1 for query variables), while the second indicates whether the node belongs to the query subset (true if it does, false otherwise).

To meet the uniform embedding size requirements of HGNNs, we pad the embeddings with zeros. However, when clique sizes vary significantly, this results in a substantial number of zeros in the embeddings. To mitigate this issue, we reparameterize the PGM so that all cliques have a uniform size. This is accomplished by first grouping variables into clusters of a fixed size, initializing each cluster potential to 1, and then multiplying each factor from the PGM with a cluster potential that includes all variables involved in that factor. This reparameterization preserves the joint distribution of the PGM while ensuring uniform clique sizes.

We then apply the hypergraph attention operator (Bai et al., 2019) to aggregate these embeddings. The attention layers process and combine information, producing aggregated node embeddings that serve as the final representation. This approach captures a rich representation incorporating (1) the PGM's structure, (2) its parameters, and (3) the details of the specific query. By accumulating information from neighboring nodes and cliques, it results in more expressive node embeddings. Additionally, this method ensures an *injective* mapping from any-MPE query to its corresponding embedding, preserving the uniqueness of each query's representation.

5.2.3 Neural Network Training

Given a PGM defined over n variables, we use the encodings from either Section 5.2.1 or Section 5.2.2 as inputs to a neural network. For the HGNN-based encoding, we concatenate all node embeddings to form a final representation. The network outputs n nodes using sigmoid activation, where each output node corresponds to a variable $X_i \in \mathbf{X}$. Outputs associated with evidence variables are excluded, and the self-supervised loss function is defined over the outputs for the query variables in $\mathbf{Q}$. The MPE solution is reconstructed by converting the continuous output values in $[0, 1]$ to binary values in $\{0, 1\}$.

We generate the training data following the process described by Arya et al. (2024c). The neural network, along with the encoder parameters (if applicable), is trained using a tractable and differentiable self-supervised loss function (see Section 4.2.2), which allows efficient parameter learning from *unlabeled data*. Once trained, the model can answer arbitrary MPE queries over the PGM.

5.3 Efficient Discretization Techniques for Better MPE Solutions

Both theoretical analyses and empirical evidence (Chizat and Bach, 2018; Allen-Zhu et al., 2019; Du et al., 2019; Zhang et al., 2017; Qin et al., 2024; Arora et al., 2019; Jacot et al., 2018; Lee et al., 2020) suggest that over-parameterized networks with non-convex loss functions (such as the one described in Section 4.2.2) typically converge near the global minimum after repeated gradient updates. Given that the true discrete and continuous losses coincide at $\{0, 1\}^n$, a near-optimal discrete solution is likely to be located near the continuous outputs.

The outputs of the neural networks are continuous, requiring conversion to discrete values to obtain MPE solutions. Recent methods (Arya et al., 2024b,c) use thresholding for this purpose. However, even when the neural network converges near the global minimum, this approach can miss the optimal solution due to the non-linearity of the loss function. Furthermore, it discards valuable information from the continuous outputs by only considering the nearest discrete solution, often leading to suboptimal results, especially for non-linear loss functions.

To address these limitations, we present two novel discretization techniques that provide superior solutions. Unlike thresholding, these methods go beyond selecting a single nearest discrete point by leveraging the output probabilities to assess multiple candidate solutions or employing an oracle for uncertain variables. These approaches effectively leverage the rich information within the continuous output distribution, leading to more accurate MPE solutions.

5.3.1 Oracle-Assisted Uncertainty-Aware Inference

The first proposed method, Oracle-Assisted Uncertainty-Aware Inference (OAUAI), uses an Oracle to reduce the likelihood of generation of incorrect MPE solutions by specifically addressing variables where the neural network exhibits low confidence. We first take the continuous output of the neural network and compute confidence scores for each query variable as $cs_i = |\mathbf{q}_i^c - 0.5|$. These scores allow us to divide the query variables into two subsets: (1) confident variables (high cs_i) and (2) uncertain variables (low cs_i).

While confident variables are directly thresholded to produce binary outputs, uncertain variables are handled differently to avoid relying on potentially inaccurate network predictions. For these variables, an MPE Oracle is queried to determine values that optimize the MPE objective. Similar to the classic cutset conditioning method (Pearl and Dechter, 1990), the Oracle processes a modified query where the confident variables serve as additional evidence, thereby reducing the number of variables involved in the query set. The final MPE solution is then constructed by merging Oracle's output for the uncertain variables with the binary outputs of the confident variables.

This method provides two key advantages: (1) It enhances solution quality by deferring the handling of low-confidence variables to the Oracle. (2) Limiting the Oracle's queries to uncertain variables reduces the query set size, which increases the Oracle's efficiency and decreases the overall computational complexity.

5.3.2 Fast Heuristic Search for Closest Binary Solutions

The second discretization method, referred to as Fast Heuristic Search for Closest Binary Solutions (k-Nearest), aims to identify multiple discrete candidates near the neural network's continuous output and select the optimal solution. Given that exhaustively scoring all $2^{|q|}$ possible discrete solutions is computationally prohibitive, we employ a heuristic search algorithm to efficiently select the k-nearest discrete solutions. This approach balances solution quality and computational feasibility, enabling the identification of high-quality discrete outputs without exhaustive enumeration.

First, we define a distance measure between the continuous output $\mathbf{q}^c$ and a binary vector $\mathbf{b}$ of size N as:

$$D(\mathbf{q}^c, \mathbf{b}) = \sum_{i=1}^{N} |q_i^c - b_i|.$$

The following algorithm leverages this distance measure to guide the heuristic search, enabling the efficient approximation of the k-nearest discrete solutions.

The algorithm employs pruning techniques to substantially narrow the search space at each iteration by retaining only the top-k partial assignments. With a low time complexity ($O(N \times k \log k)$), it allows for larger values of k, thereby yielding more accurate solutions. By exploiting the problem's optimal substructure, the algorithm iteratively builds upon partial solutions to construct the final set of k-nearest discrete solutions.

Consequently, the algorithms from 5.3.1 and 5.3.2 function as advanced discretization methods for deriving near-optimal MPE solutions. We can apply both techniques to the neural network

Algorithm 2 Top-k Closest Binary Assignment Generation

```
1: Input: $\mathbf{q^c}$, $N$, $k$
2: Output: Top $k$ binary assignments $\mathbf{B}^*$ minimizing $D(\mathbf{q^c}, \mathbf{b})$
3: $L \leftarrow [(0, [])]$                                  ▷ (distance, partial assignment)
4: for $i = 1$ to $N$ do
5:     $T \leftarrow []$                                     ▷ Temporary list
6:     for all $(d, a) \in L$ do
7:         $T \leftarrow T \cup (d + q_i^c, a + [0])$                  ▷ For $b_i = 0$
8:         $T \leftarrow T \cup (d + (1 - q_i^c), a + [1])$            ▷ For $b_i = 1$
9:     end for
10:    Sort $T$ by $d$ and keep top $k$
11:    $L \leftarrow T$
12: end for
13: Return top $k$ assignments in $L$
```

outputs, selecting the superior solution for each instance from the two approaches. This combined strategy further improves overall solution quality by leveraging the complementary strengths of both discretization methods.

5.4 Experiments

In this section, we evaluate the HGNN-based embedding (Section 5.2.2) along with the two thresholding methods presented in Sections 5.3.1 and 5.3.2. We benchmark these against several baselines, including both neural network-based and traditional algorithms that directly operate on probabilistic models. We begin by outlining the experimental framework, which includes the competing methods, evaluation metrics, neural network architectures, and probabilistic graphical models used in the study.

5.4.1 Graphical Models

We evaluated PGMs using thirty high-treewidth binary models from UAI inference competitions (Elidan and Globerson, 2010; Dechter et al., 2022). Of these models, twenty-five featured cliques larger than two, with variable counts ranging from 360 to 1444 and a maximum clique size of 12.

The remaining five were pairwise networks with variable counts ranging from 400 to 1600. For all models, we merged smaller cliques to ensure all cliques were of size ≥ 4, using the method described in 5.2.2.

Table 5.1: Dataset Specifications for Higher-Order Probabilistic Graphical Models

Dataset	Number of Variables	Number of Factors	Avg. Clique Size	Max. Clique Size
BN 80	360	360	6.18	12.00
BN 82	360	360	6.18	12.00
BN 84	360	360	6.18	12.00
BN 65	440	440	3.00	5.00
BN 46	499	499	3.40	6.00
BN 59	540	540	3.00	5.00
BN 63	540	540	3.00	5.00
BN 53	561	561	3.00	5.00
BN 55	561	561	3.00	5.00
BN 57	561	561	3.00	5.00
Maxsat aes 64	596	2780	3.00	5.00
BN 47	661	661	3.00	5.00
BN 49	661	661	3.00	5.00
BN 51	661	661	3.00	5.00
BN 61	667	667	3.00	5.00
BN 42	880	880	3.00	5.00
BN 43	880	880	3.00	5.00
BN 44	880	880	3.00	5.00
BN 45	880	880	3.00	5.00
BN 30	1156	1156	2.00	3.00
BN 32	1444	1444	2.00	3.00
BN 34	1444	1444	2.00	3.00
BN 36	1444	1444	2.00	3.00
BN 38	1444	1444	2.00	3.00
BN 40	1444	1444	2.00	3.00

Table 5.2: Dataset Specifications for Pairwise Probabilistic Graphical Models

Dataset	Number of Variables	Number of Factors	Avg. Clique Size	Max. Clique Size
grid20x20.f5.wrap	400	1201	1.00	2.00
grid40x40.f10	1600	4721	1.00	2.00
grid40x40.f10.wrap	1600	4801	1.00	2.00
grid40x40.f15	1600	4721	1.00	2.00
grid40x40.f15.wrap	1600	4801	1.00	2.00

Tables 5.1 and 5.2 summarize the details of the probabilistic graphical models used in the experiments. For higher-order PGMs, the number of variables ranges from 360 to 1444, with an average clique size between 2.00 and 6.18. The maximum clique size varies from 3.00 to 12.00. In contrast, the pairwise PGMs exhibit a more uniform structure across the grid-based datasets, with variables ranging from 400 to 1600 and factors ranging from 1201 to 4801.

We sampled data from the PGMs using Gibbs Sampling, generating 16,000, 2,000, and 2,000 examples for the training, testing, and validation sets, respectively. The query ratio (qr), defined as the fraction of variables in the query set, was varied across $\{0.1, 0.3, 0.5, 0.7, 0.8, 0.9\}$ for each PGM.

5.4.2 Baseline Methods and Evaluation Criteria

Baselines - We evaluated our methods against both traditional and neural-based MPE solvers. As a traditional baseline, we employed the MP algorithm (Pearl, 1988), implemented by Lowd and Rooshenas (2015). The MP algorithm serves as an approximate inference technique designed to identify the Most Probable Explanation (MPE) state.

As a neural-based baseline, we employed SSMP (Arya et al., 2024b), which only encodes the inference query information as input to a neural network and applies thresholding for discretization. This approach enabled comparisons across two dimensions of neural methods: embedding and discretization techniques, which were tested in different combinations during evaluation.

We did not include the inference time optimization scheme (ITSELF) from Arya et al. (2024c) in our comparisons, as it modifies network parameters during inference time to better fit the given query. However, our proposed methods are compatible with ITSELF and could be integrated to benefit from its test time optimization capabilities.

Evaluation Criteria – The competing approaches were assessed based on log-likelihood (LL) scores, calculated as $\ln p_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$, and inference times for a given evidence $\mathbf{e}$ and query output $\mathbf{q}$.

5.4.3 Advanced Embedding and Discretization Methods

For each PGM and query ratio, we trained a neural network using the embeddings described in 5.2.2 and the self-supervised loss from 4.2.2. An attention-based HGNN layer was employed for embedding aggregation, with an input embedding size of 2^k, where k is the number of variables in the largest clique, and an output embedding size of 64. Each model was trained for 20 epochs using the loss function described in 4.2.2. We standardized the network architecture (for the network that processes the embeddings) across all experiments, using a fully connected NN with three hidden layers (128, 256, and 512 nodes). All hidden layers used ReLU activation functions, while the output layer applied sigmoid functions with dropout regularization (Srivastava et al., 2014). The models were trained using the Adam optimizer (Kingma and Ba, 2015) and implemented in PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019). All experiments were conducted on an NVIDIA A40 GPU.

5.4.4 Hyperparameters for Discretization Methods

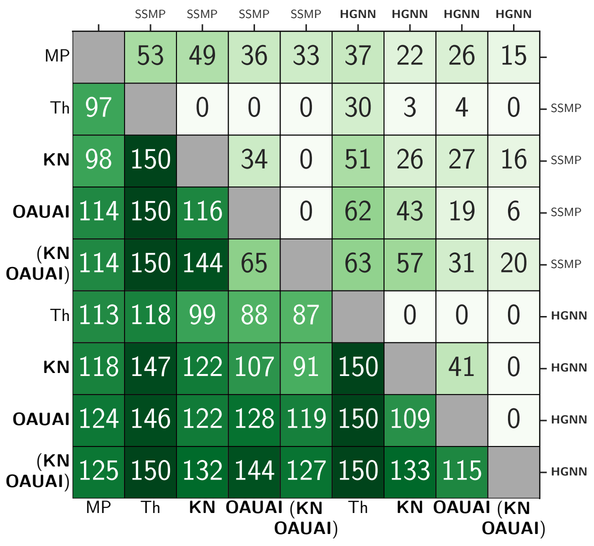
We applied four discretization methods: (1) the thresholding (Thresh) technique, which also serves as the baseline discretization method in our experiments, using a threshold of 0.5, as described in Arya et al. (2024b,c), (2) the OAUAI method from 5.3.1, (3) the k-Nearest method from 5.3.2, and (4) a combination of the two (OAUAI | k-Nearest). For OAUAI, we limited processing to a maximum of 1/4 of the query variables or 200 (whichever was smaller) using AOBB as the chosen oracle, following the method described in Otten and Dechter (2012) method (implementated by Otten (2012)). For k-Nearest, we set k to 1024.

For the neural network models, we used a mini-batch size of 512 and applied a learning rate decay of 0.9 whenever the loss plateaued, improving training efficiency. Parameters were initialized using the Kaiming normal distribution to ensure appropriate weight scaling and support effective training.

We implemented the attention-based HGNN using PyTorch Geometric (Fey and Lenssen, 2019), utilizing a single head for the attention module. Optimal hyperparameters were determined through extensive 5-fold cross-validation, while other hyperparameters followed the configurations in Arya et al. (2024b,c). Additional hyperparameter details are available in Arya et al. (2025).

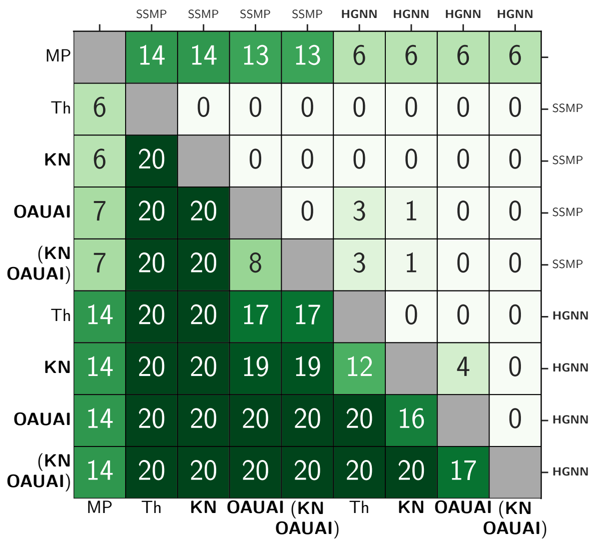
For the baseline MP method, we utilized the implementation and default parameters from Lowd and Rooshenas (2015). For comparison with optimal solutions, we employed AOBB (Otten and Dechter, 2012), using the implementation from Otten (2012), with default parameters. This algorithm also served as the oracle in the OAUAI method.

We evaluated the methods—including the traditional non-neural baseline, embedding schemes (query-based baseline embedding (SSMP) and our proposed hypergraph neural network-based embedding (HGNN)), and discretization schemes (the thresholding-based baseline (Thresh), and our proposed oracle-based (OAUAI), nearest discrete solutions-based (k-Nearest), and the combination of the latter two (HGNN: OAUAI | k-Nearest))—on 30 PGMs.



		SSMP	SSMP	SSMP	SSMP	HGNN	HGNN	HGNN	HGNN
	MP	Th	**KN**	**OAUAI**	**(KN OAUAI)**	Th	**KN**	**OAUAI**	**(KN OAUAI)**	
MP		53	49	36	33	37	22	26	15	
Th	97		0	0	0	30	3	4	0	SSMP
KN	98	150		34	0	51	26	27	16	SSMP
OAUAI	114	150	116		0	62	43	19	6	SSMP
(KN OAUAI)	114	150	144	65		63	57	31	20	SSMP
Th	113	118	99	88	87		0	0	0	HGNN
KN	118	147	122	107	91	150		41	0	HGNN
OAUAI	124	146	122	128	119	150	109		0	HGNN
(KN OAUAI)	125	150	132	144	127	150	133	115		HGNN

(a) CT: Higher-Order Models



		SSMP	SSMP	SSMP	SSMP	HGNN	HGNN	HGNN	HGNN
	MP	Th	**KN**	**OAUAI**	**(KN OAUAI)**	Th	**KN**	**OAUAI**	**(KN OAUAI)**	
MP		14	14	13	13	6	6	6	6	
Th	6		0	0	0	0	0	0	0	SSMP
KN	6	20		0	0	0	0	0	0	SSMP
OAUAI	7	20	20		0	3	1	0	0	SSMP
(KN OAUAI)	7	20	20	8		3	1	0	0	SSMP
Th	14	20	20	17	17		0	0	0	HGNN
KN	14	20	20	19	19	12		4	0	HGNN
OAUAI	14	20	20	20	20	20	16		0	HGNN
(KN OAUAI)	14	20	20	20	20	20	20	17		HGNN

(b) CT: Pairwise Models

Figure 5.2: Contingency tables (a) and (b): baseline methods vs. our approach. Each cell: frequency of the row method outperforming the column method. Method names: embedding and discretization techniques. Proposed method names are in bold.

5.4.5 Empirical Evaluations

Evaluating Traditional and Neural Baselines vs Our Methods:

Table 5.2a presents the contingency tables for PGMs with higher-order cliques. We generated 150 test datasets for the MPE task, utilizing 25 higher-order PGMs across six different query ratios (qr). Each cell (i, j) in the table indicates the number of instances (out of 150) where the method in row i outperformed the method in column j based on average log-likelihood scores. Discrepancies between 150 and the sum of values for cells (i, j) and (j, i) reflect cases where the methods achieved comparable scores.

Similarly, Table 5.2b displays the contingency tables for pairwise models. Here, we generated 20 test datasets using five PGMs across four query ratios (qr), applying the same interpretative framework as for Table 5.2a.

Both Tables 5.2a and 5.2b employ a color scale to convey the comparative strength of the methods: darker green shades suggest that the row method outperformed the column method in many experiments, while lighter shades or white (indicating a cell value of 0) suggest the opposite, with the column method outperforming more frequently. Each cell also displays its corresponding value.

The analysis of the contingency tables for both higher-order and pairwise models reveals a consistent pattern, where the left side of the diagonal shows higher values, represented by darker green shades. This indicates that methods listed in the lower rows generally outperform those in the earlier columns. This trend underscores the superiority of HGNN-based methods, which are positioned in these lower rows, over alternative approaches. Consequently, HGNN configurations consistently achieve higher log-likelihood scores, surpassing both baseline and SSMP methods in effectiveness. To facilitate a deeper analysis, we will now address several key questions.

Q1. How do HGNN-based methods compare to traditional baseline MP?

The values in the first row and first column provide a comparison between the HGNN-based methods and the traditional MP baseline. Specifically, the last four values in the first column indicate how often the HGNN-based embedding, using different discretization methods, outperformed MP. Among these, HGNN: OAUAI | k-Nearest demonstrated the strongest performance, surpassing the baseline in over 80% of cases for higher-order models and more than 70% for pairwise models. Other methods also displayed competitive performance, indicating that models that use our HGNN-based embedding consistently outperform MP in most scenarios.

Q2. Does a more sophisticated hypergraph-based embedding help?

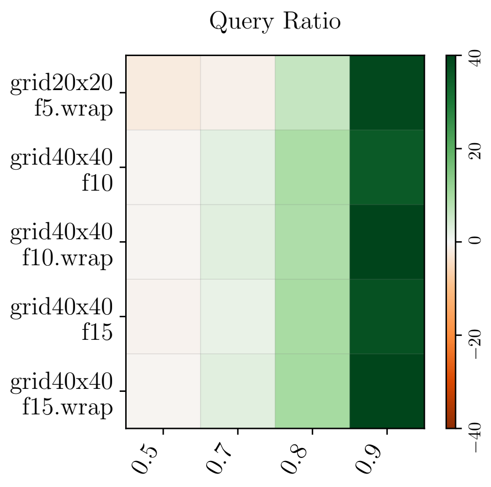
When comparing different methods that use the same discretization technique but differ in their embedding schemes, the HGNN-based embedding consistently outperforms those relying solely on query information. For higher-order networks, the HGNN-based embedding is superior in at least 78% of cases, while for pairwise models, it outperforms in the other method in every instance. This indicates that integrating information from the PGM alongside query data significantly enhances performance compared to embeddings based solely on query information.

Q3. Are advanced discretization techniques more effective than simple thresholding?

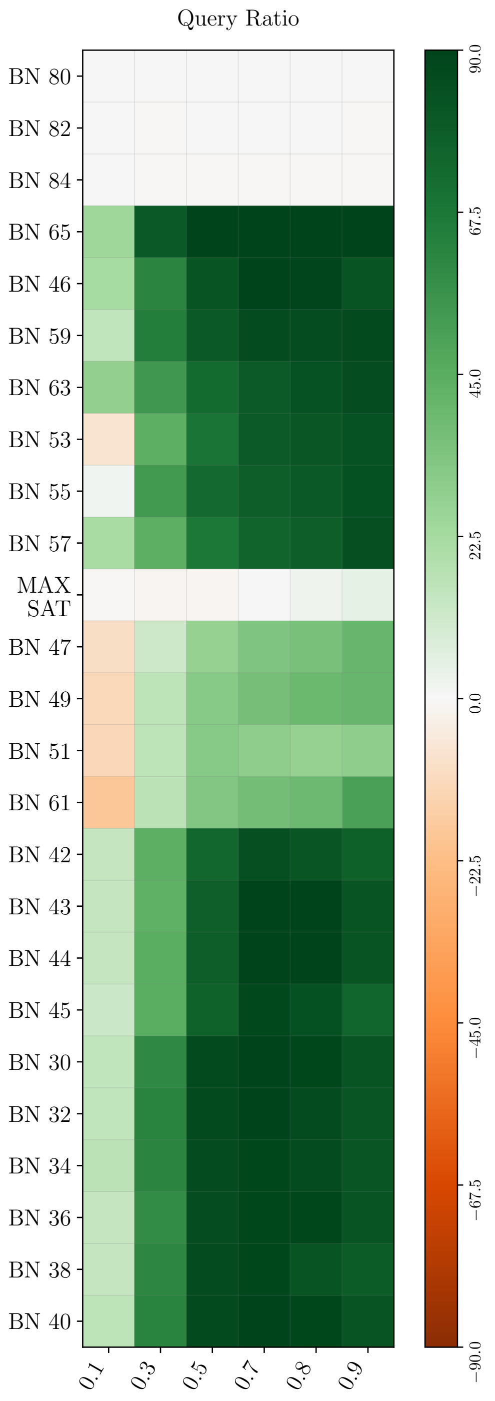
By examining methods that utilize identical encoding schemes but differ in discretization techniques, we observe OAUAI performs the best following by k-Nearest, and Thresh. Notably, OAUAI | k-Nearest outperforms Thresh in nearly all cases for both embeddings and across higher-order and pairwise models. Furthermore, Thresh, which serves as the baseline for our experiments, does not outperform any of the proposed methods, highlighting the substantial improvements offered by advanced discretization techniques.

Detailed Comparison of Our Best Approach Against MP

To offer a more detailed comparison with the traditional baseline (MP), we use heatmaps in Figures 5.3a (Pairwise models) and 5.3b (Higher-order models) to illustrate the performance of HGNN: OAUAI | k-Nearest (our best method) relative to the baseline. Along the y-axis, datasets are arranged by the number of variables, while the x-axis represents different query ratios. Each cell displays the percentage difference in mean log-likelihood (LL) scores between the methods, calculated as $\%$Diff. $= 100 \times (ll_{nn} - ll_{bp})/|ll_{bp}|$. Green cells indicate where our method outperforms



(a) HM: Pairwise Models



(b) HM: Higher-Order Models

Figure 5.3: Heatmaps (a) and (b) display % differences in LL scores (color bar) between MP and HGNN: OAUAI | k-Nearest. X-axis: query ratio; Y-axis: dataset names. Green cells: our method is better; Orange cells: MP performs better. In both figures, darker shades indicate larger values.

MP, while orange cells denote cases where MP performs better. Darker shades correspond to larger differences, with lighter shades indicating smaller differences; white cells denote no difference.

For pairwise models (Figure 5.3a), HGNN: OAUAI | k-Nearest perform comparably to the MP approximation when the query set size is small, as indicated by the light orange cells. However, as the problem complexity increases with larger query set sizes, our method consistently outperforms MP across all datasets, shown by the green color. In these cases, log-likelihood score improvements reach up to 40%.

A similar pattern emerges for higher-order models (Figure 5.3b), where our method matches the baseline's performance for the smallest query ratio. However, as the query ratio increases, the performance improvement becomes more evident (up to 90% improvement), as indicated by the prevalence of dark green cells across most datasets. The heatmap features only a few orange and white cells, indicating that our method consistently outperforms the MP approximation in the majority of scenarios.

One key reason for better performance on higher-order networks is that these models inherently encode a large amount of information within each clique, as each clique is represented by 2^k parameters (where k is the size of the clique). This richness enables the HGNN-based model to significantly enhance the quality of MPE solutions. However, when combining cliques (e.g., $k_1 \dots k_m$) to form larger cliques from smaller ones, we transition from $2^{k_1} + 2^{k_2} + \cdots + 2^{k_m}$ parameters to $2^{k_1+k_2+\cdots+k_m}$ parameters. This increase in parameters can result in embeddings that are less representative. Additionally, the pairwise models used (networks from the grid family) often exhibit similar values in their factors, particularly for the (0,0):(1,1) and (1,0):(0,1) pairs. As a result, when we take the product of cliques, many of the initial embeddings also have similar values, leading to less informative embeddings.

Table 5.3: Gap Between AOBB and All Other Methods, Including Baselines and Proposed Methods, for Higher-Order PGMs: Query Ratio 0.5.

PGM	SSMP Th	SSMP KN	SSMP OAUAI	SSMP (KN|OAUAI)	HGNN Th	HGNN KN	HGNN OAUAI	HGNN (KN|OAUAI)	|Worst-Best|
BN 42	0.10524	0.06363	0.06507	0.05166	0.15445	0.08236	0.04370	**0.03969**	0.11476
BN 43	0.21902	0.15704	0.17266	0.14554	0.25150	0.15481	0.04944	**0.04876**	0.20274
BN 44	0.18344	0.13335	0.12052	0.11149	0.10628	0.04523	0.05316	**0.03503**	0.14841
BN 45	0.17915	0.13698	0.11053	0.10570	0.08086	0.03768	0.03712	**0.02808**	0.15107
BN 46	0.28021	0.16378	0.18497	0.14652	0.38591	0.23951	0.16023	**0.14308**	0.24282
BN 65	0.41790	0.37808	0.22508	0.22489	0.11301	0.04416	0.01785	**0.01775**	0.40015
BN 80	0.27192	0.15047	0.08559	0.07923	0.00109	**0.00000**	0.00088	**0.00000**	0.27191
BN 82	0.15444	0.06093	0.08547	0.04847	0.00131	**0.00002**	0.00095	**0.00002**	0.15445
BN 84	0.20071	0.11452	0.15966	0.10912	0.06721	**0.00049**	0.06718	**0.00049**	0.20022
MAX-SAT	0.03481	0.03248	0.02062	0.02062	0.03021	0.02819	**0.01856**	**0.01856**	0.01625

Table 5.4: Gap Between AOBB and All Other Methods, Including Baselines and Proposed Methods, for Pairwise PGMs: Query Ratio 0.5.

PGM	SSMP Th	SSMP KN	SSMP OAUAI	SSMP (KN|OAUAI)	HGNN Th	HGNN KN	HGNN OAUAI	HGNN (KN|OAUAI)	|Worst-Best|
grid20x20.f5.wrap	0.07852	0.06316	0.05627	0.05356	0.05836	0.04185	0.02456	**0.02406**	0.05446
grid40x40.f10	0.01859	0.01642	0.01586	0.01540	0.01312	0.01075	0.01154	**0.01023**	0.00835
grid40x40.f10.wrap	0.01874	0.01645	0.01344	0.01337	0.01124	0.00909	0.00798	**0.00756**	0.01118
grid40x40.f15	0.02260	0.02004	0.01535	0.01533	0.01715	0.01455	**0.01151**	**0.01151**	0.01109
grid40x40.f15.wrap	0.01858	0.01627	0.01323	0.01315	0.01388	0.01388	0.01031	**0.01006**	0.00853

5.5 Comparing MPE Solutions of Proposed Methods Against Near-Optimal Solutions

Tables 5.3 and 5.4 present the log-likelihood score gaps between AOBB and various neural network techniques, including embedding schemes—query-based (SSMP) and hypergraph neural network-based (HGNN)—as well as discretization schemes: thresholding (Thresh), oracle-based (OAUAI), nearest discrete solutions-based (k-Nearest), and their combination (HGNN: OAUAI | k-Nearest).

For each method M, the gap is calculated as the relative difference between the near-optimal score (determined by AOBB) and the score achieved by M. We evaluate the performance on datasets where AOBB can feasibly identify near-optimal or exact solutions. In this experiment, the query ratio is set to 0.5 to improve the quality of the near-optimal solution. Higher query ratios increase problem difficulty, making it more challenging to achieve near-optimal solutions.

The neural-based approach that achieves the best performance for each dataset is highlighted in **bold**. Smaller values indicate better performance, as a smaller gap implies the method is closer to near-optimal solutions. The last column presents the difference between the best (lowest) gap and the worst (highest) gap.

Notably, HGNN: OAUAI | k-Nearest consistently outperforms other neural baselines across most datasets. This method is followed by OAUAI and k-Nearest, both utilizing HGNN for embedding. This analysis offers a comprehensive comparison of the proposed methods with near-optimal solutions, demonstrating that HGNN-based approaches, particularly OAUAI, k-Nearest, and HGNN: OAUAI | k-Nearest, consistently produce solutions with quality close to the optimal solutions.

5.6 Inference Times

We present the inference times for all baselines and proposed methods in Figures 5.4 and 5.5. Both figures employ a logarithmic microsecond scale, where color intensity represents inference duration—lighter hues denote shorter times.

For all PGMs, the SSMP method with Thresh and the HGNN method with Thresh exhibit the fastest inference, with SSMP being marginally quicker. This marginal difference arises because the HGNN method requires evidence instantiation to generate richer embeddings. Furthermore, the attention module in HGNN slightly increases inference time.

Following these are the k-Nearest, OAUAI, and HGNN: OAUAI | k-Nearest methods, which maintain the same relative ranking for both embeddings. Although these discretization schemes require more time than the thresholding method (Thresh), they produce significantly better solutions. Reducing the value of the hyperparameters—k for k-Nearest and the number of query variables for OAUAI—can decrease inference time, though at the cost of solution quality. Conversely, increasing the value of these hyperparameters can improve solution quality at the cost of longer inference times. The MP baseline exhibits the longest inference times.

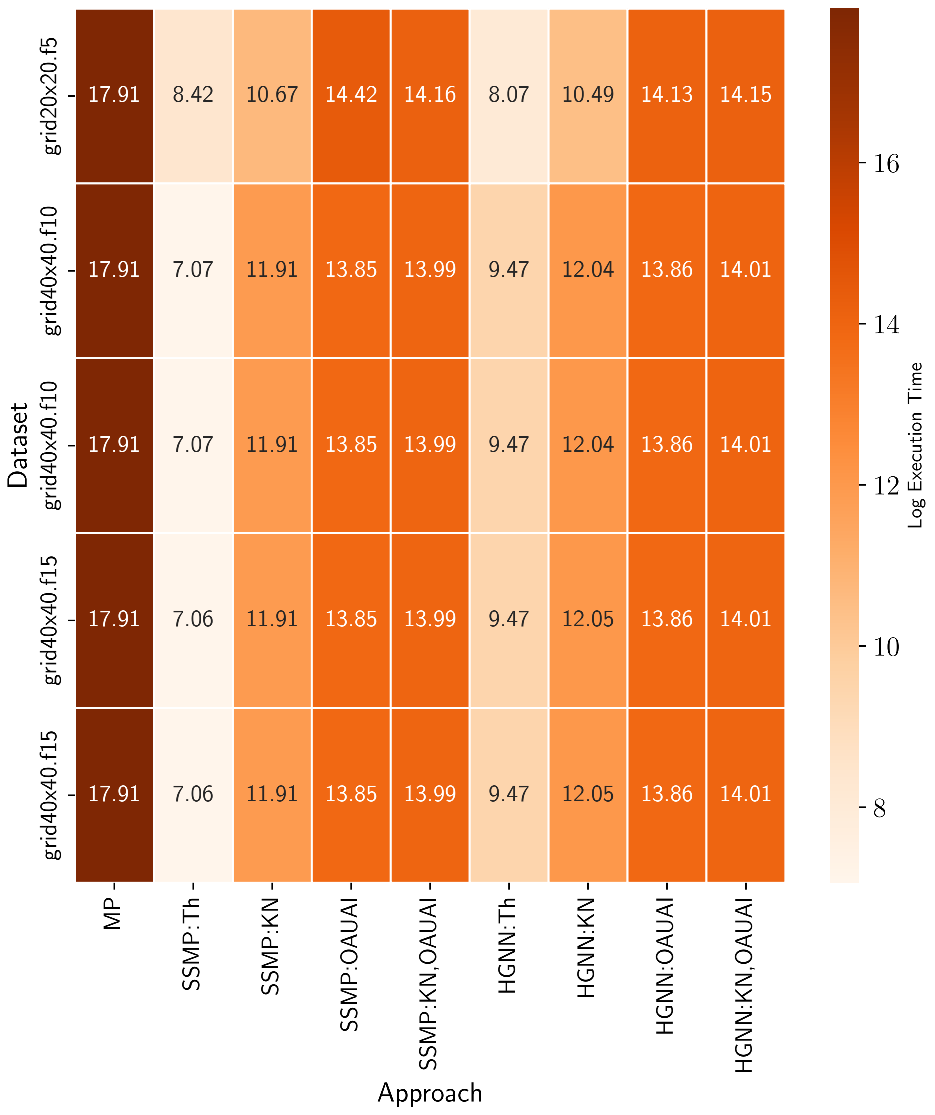
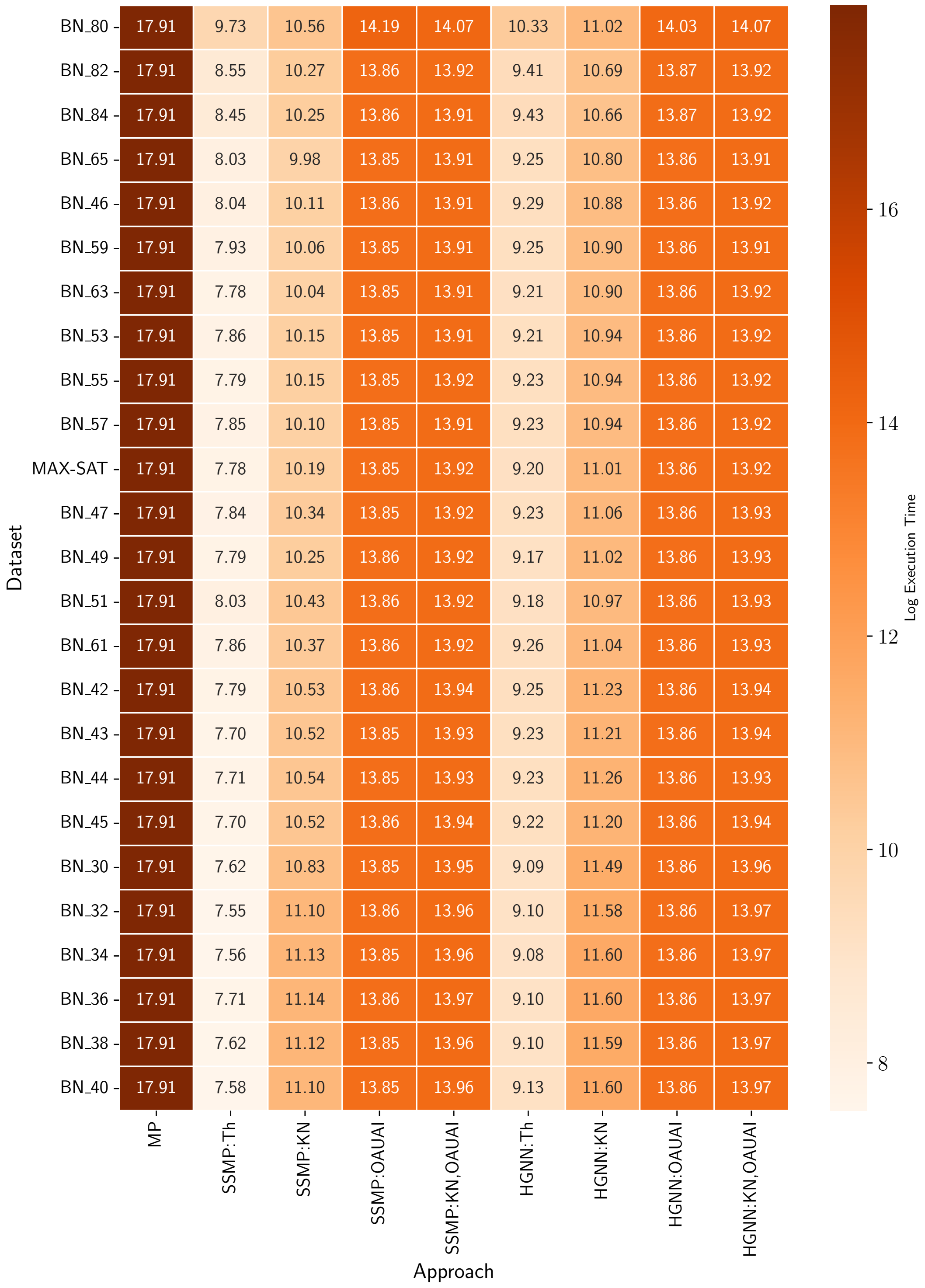


Figure 5.4: Heatmap of inference times for Pairwise PGMs. The logarithmic microsecond scale is represented by color intensity, with lighter hues indicating shorter, more efficient inference durations.



Dataset	MP	SSMP:Th	SSMP:KN	SSMP:OAUAI	SSMP:KN,OAUAI	HGNN:Th	HGNN:KN	HGNN:OAUAI	HGNN:KN,OAUAI
BN_80	17.91	9.73	10.56	14.19	14.07	10.33	11.02	14.03	14.07
BN_82	17.91	8.55	10.27	13.86	13.92	9.41	10.69	13.87	13.92
BN_84	17.91	8.45	10.25	13.86	13.91	9.43	10.66	13.87	13.92
BN_65	17.91	8.03	9.98	13.85	13.91	9.25	10.80	13.86	13.91
BN_46	17.91	8.04	10.11	13.86	13.91	9.29	10.88	13.86	13.92
BN_59	17.91	7.93	10.06	13.85	13.91	9.25	10.90	13.86	13.91
BN_63	17.91	7.78	10.04	13.85	13.91	9.21	10.90	13.86	13.92
BN_53	17.91	7.86	10.15	13.85	13.91	9.21	10.94	13.86	13.92
BN_55	17.91	7.79	10.15	13.85	13.92	9.23	10.94	13.86	13.92
BN_57	17.91	7.85	10.10	13.85	13.91	9.23	10.94	13.86	13.92
MAX-SAT	17.91	7.78	10.19	13.85	13.92	9.20	11.01	13.86	13.92
BN_47	17.91	7.84	10.34	13.85	13.92	9.23	11.06	13.86	13.93
BN_49	17.91	7.79	10.25	13.86	13.92	9.17	11.02	13.86	13.93
BN_51	17.91	8.03	10.43	13.86	13.92	9.18	10.97	13.86	13.93
BN_61	17.91	7.86	10.37	13.86	13.92	9.26	11.04	13.86	13.93
BN_42	17.91	7.79	10.53	13.86	13.94	9.25	11.23	13.86	13.94
BN_43	17.91	7.70	10.52	13.85	13.93	9.23	11.21	13.86	13.94
BN_44	17.91	7.71	10.54	13.85	13.93	9.23	11.26	13.86	13.93
BN_45	17.91	7.70	10.52	13.86	13.94	9.22	11.20	13.86	13.94
BN_30	17.91	7.62	10.83	13.85	13.95	9.09	11.49	13.86	13.96
BN_32	17.91	7.55	11.10	13.86	13.96	9.10	11.58	13.86	13.97
BN_34	17.91	7.56	11.13	13.85	13.96	9.08	11.60	13.86	13.97
BN_36	17.91	7.71	11.14	13.86	13.97	9.10	11.60	13.86	13.97
BN_38	17.91	7.62	11.12	13.85	13.96	9.10	11.59	13.86	13.97
BN_40	17.91	7.58	11.10	13.85	13.96	9.13	11.60	13.86	13.97

Figure 5.5: Heatmap of inference times for Higher-Order PGMs. Color intensity represents a logarithmic microsecond scale, with lighter hues indicating shorter inference durations.

5.7 Chapter Summary

In this chapter, we introduced novel techniques for solving Most Probable Explanation (MPE) queries in probabilistic graphical models, starting with an advanced encoding strategy that employs a HGNN to aggregate information from the query, graphical model structure, and parameters. This approach eliminates the need for test-time optimization to achieve near-optimal solutions. Additionally, we proposed two novel discretization schemes, k-Nearest and OAUAI, for converting the continuous outputs of the neural network into discrete MPE solutions. These schemes consistently outperform traditional thresholding methods, which correspond to a specific form of k-Nearest method when $k = 1$. Our evaluation on 30 binary high tree-width probabilistic graphical model benchmarks demonstrated that our method significantly outperforms traditional baselines and other neural approaches.

CHAPTER 6

LEARNING TO SOLVE THE CONSTRAINED MOST PROBABLE EXPLANATION TASK IN PROBABILISTIC GRAPHICAL MODELS

6.1 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.

The constrained most probable explanation (CMPE) task aims to identify the most probable values $\mathbf{X} = \mathbf{x}$ with respect to f that satisfies the constraint $g(\mathbf{x}) \leq q$. Although both MPE and CMPE are NP-hard in general, CMPE is more challenging to solve in practice than MPE. Notably, CMPE is NP-hard even for PGMs with 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) demonstrated that several probabilistic inference queries are special cases of CMPE, including 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 the robot's predictions and underscores the practical significance of CMPE.

In this chapter, we explore novel machine learning techniques for the CMPE task, inspired by advances in *learning to optimize* (Donti et al., 2020; Fioretto et al., 2020; Park and Hentenryck, 2023; Zamzam and Baker, 2020). These works propose training a neural network that takes the constraints, observations, etc. of an optimization problem *as input* and produces a near-optimal solution to the 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 methods 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. (2022b) 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.

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 Hentenryck, 2023) in the *learning to optimize* literature either relax the constrained objective function using Lagrangian relaxation and then use the Langragian 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 this 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.

Section 6.2 explores machine learning strategies from the learning to optimize literature, and how these strategies can be adapted to tackle the CMPE task. In Section 6.3, the focus will be on our novel self-supervised neural network-based approach that has been designed to train neural networks to provide near-optimal solutions for address CMPE. Finally, Section 6.4 is dedicated to the empirical evaluation and comparison of the proposed method against other supervised and self-supervised approaches.

6.2 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., 2020; Fioretto et al., 2020; Park and Hentenryck, 2023; Zamzam and Baker, 2020) 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.

6.2.1 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 equation 2.8 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 equation 2.8 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 (2020), we propose to train $\mathcal{F}_\Theta$ using the following two loss functions

$$\text{Mean-Squared Error (MSE)} : \frac{1}{N} \sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{6.1}$$

$$\text{Mean-Absolute-Error (MAE)} : \frac{1}{N} \sum_i |\mathbf{y}_i - \hat{\mathbf{y}}_i| \tag{6.2}$$

Experimentally, 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 $\lambda_{\mathbf{x}_i}^k$ and $\hat{\mathbf{y}}_i^k$ 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 $\lambda_{\mathbf{x}_i}^{k+1}$ using

$$\lambda_{\mathbf{x}_i}^{k+1} = \lambda_{\mathbf{x}_i}^{k} + \rho \max\{0, g_{\mathbf{x}_i}(\hat{\mathbf{y}}_i^k)\} \tag{6.3}$$

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.

6.2.2 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 Hentenryck (2023), 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., 2020; 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:

$$\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\left\{0, g_{\mathbf{x}}(\mathbf{y})\right\}^2 \tag{6.4}$$

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:

$$\mathcal{L}_{\mathbf{x}}^{pen}(\hat{\mathbf{y}}) = f_{\mathbf{x}}(\hat{\mathbf{y}}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\left\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\right\}^2 \tag{6.5}$$

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 equation 6.3. More formally, we can update $\lambda_{\mathbf{x}}^k$ using:

$$\lambda_{\mathbf{x}}^{k+1} = \lambda_{\mathbf{x}}^k + \rho \max\left\{0, g_{\mathbf{x}}(\hat{\mathbf{y}}^k)\right\} \tag{6.6}$$

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 equation 6.4):

$$\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \frac{\lambda_{\mathbf{x}}^k}{2} \max\left\{0, g_{\mathbf{x}}(\mathbf{y})\right\}^2 + \mu_{\mathbf{x}}^k g_{\mathbf{x}}(\mathbf{y}) \tag{6.7}$$

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

$$\mu_{\mathbf{x}}^{k+1} = \max\left\{0, \mu_{\mathbf{x}}^k + \lambda_{\mathbf{x}}^k g_{\mathbf{x}}(\mathbf{y}^k)\right\} \tag{6.8}$$

Recently, Park and Hentenryck (2023) 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 $\mu^k_{\mathbf{x}}$. 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:

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

While the dual network uses the following loss function

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

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

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 equation 6.5) may be far away from the optimal solution $\mathbf{y}^*$ of the problem given in equation 2.8, unless the penalty co-efficient $\lambda^k_{\mathbf{x}} \to \infty$. Moreover, when $\lambda^k_{\mathbf{x}}$ 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 $\lambda^k_{\mathbf{x}} \to \infty$ or $\forall \mathbf{x}$ with $g_{\mathbf{x}}(\mathbf{y}) > 0$, $\mu^k_{\mathbf{x}}$ should be such that $\min_{\mathbf{y}} f_{\mathbf{x}}(\mathbf{y}) + \mu^k_{\mathbf{x}} g_{\mathbf{x}}(\mathbf{y}) > p^*_{\mathbf{x}}$. 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.

6.3 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,

$$\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \begin{cases} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0 \\ \alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}})) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \end{cases} \tag{6.9}$$

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 6.3.1. *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 6.9, we have

$$\begin{aligned}\min_{\hat{\mathbf{y}}} \mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \min \Bigg\{ & \min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) \; s.t. \; g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0, \\ & \alpha_{\mathbf{x}} \left(\min_{\hat{\mathbf{y}}} \; f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \; s.t. \; g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \right) \Bigg\} \\ = & \min \left\{ p^*_{\mathbf{x}}, \alpha_{\mathbf{x}} q^*_{\mathbf{x}} \right\} \end{aligned} \tag{6.10}$$

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

$$\min \left\{ p^*_{\mathbf{x}}, \alpha_{\mathbf{x}} q^*_{\mathbf{x}} \right\} = p^*_{\mathbf{x}} \tag{6.11}$$

From equation 6.10 and equation 6.11, 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.

6.3.1 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:

$$\alpha_{\mathbf{x}} > \frac{p^*_{\mathbf{x}}}{q^*_{\mathbf{x}}} \tag{6.12}$$

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 $p^*_{\mathbf{x}}$.

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

$$\min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ s.t. } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0, \tag{6.13}$$

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,

$$\min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}}) \quad \text{s.t.} \quad g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0, \tag{6.14}$$

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. The constrained optimization task is initially transformed into an unconstrained formulation via Lagrange Relaxation. This results in the following optimization problem:

$$\max_{\mu \geq 0} \min_{\hat{\mathbf{y}}} f_{\mathbf{x}}(\hat{\mathbf{y}}) + (1-\mu) \times g_{\mathbf{x}}(\hat{\mathbf{y}}) \tag{6.15}$$

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.

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 $\mathbf{x}$.

6.3.2 Making the Loss Function Smooth and Continuous

The loss function defined in equation 6.9 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

$$\begin{aligned}\widetilde{\mathcal{L}}_{\mathbf{x}}(\hat{\mathbf{y}}) = &\Big((1-\sigma(\beta g_{\mathbf{x}}(\hat{\mathbf{y}}))) \cdot [f_{\mathbf{x}}(\hat{\mathbf{y}})]\Big) + \\ &\Big(\sigma(\beta g_{\mathbf{x}}(\hat{\mathbf{y}})) \cdot [\alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\})]\Big)\end{aligned} \tag{6.16}$$

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.

6.3.3 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

$$\mathcal{L}_{\mathbf{x}}(\hat{\mathbf{y}}) = \begin{cases} f_{\mathbf{x}}(\hat{\mathbf{y}}) \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) \leq 0 \\ \\ \alpha_{\mathbf{x}}(f_{\mathbf{x}}(\hat{\mathbf{y}}) + g_{\mathbf{x}}(\hat{\mathbf{y}})) + \rho \max\{0, g_{\mathbf{x}}(\hat{\mathbf{y}})\}^2 \text{ if } g_{\mathbf{x}}(\hat{\mathbf{y}}) > 0 \end{cases} \tag{6.17}$$

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

6.4 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.

6.4.1 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 6.2). In the main empirical evaluation, we show results on the best performing supervised loss, which is MSE with penalty, denoted by $\texttt{SL}_{pen}$.

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 6.3. 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 Hentenryck (2023) 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.

Note that all methods used the same neural network architecture 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, LLC, 2023). We report the objective values of the ILP solutions in each table (see Tables 6.2, 6.3, and 6.4).

6.4.2 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., 2020b) 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). Table 6.1 provides a comprehensive overview of the characteristics of each binary dataset, including the number of variables and functions present in each dataset. Finally, we evaluated all methods on the task of generating adversarial examples for neural network classifiers.

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 Grids17 and Grids18 networks and Segmentation12, Segmentation14 and Segmentation15 networks.

We learned MPE tractable cutset networks without latent variables using the scheme of Rahman et al. (2014) on five high-dimensional datasets: DNA (Haaren and Davis, 2012b; Ucla-Starai, 2023), NewsGroup (c20ng) (Lowd and Davis, 2010; Ucla-Starai, 2023), WebKB1 (cwebkb) (Lowd and Davis, 2010; Ucla-Starai, 2023), AD (Haaren and Davis, 2012b; Ucla-Starai, 2023), and BBC (Haaren and Davis, 2012b; 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 6.1: 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

6.4.3 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, 2020; Nellikkath and Chatzivasileiadis, 2021) and self-supervised (Park and Hentenryck, 2023; Donti et al., 2020) 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 Hentenryck, 2023), 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, 2015) with a learning rate of 10^{-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.

6.4.4 Hyperparameters

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. 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, referred to as `SS-CMPE` $_{pen}$, is detailed in Section 6.3.3 and further elaborated in Arya et al. (2024a).

Table 6.2: 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
$\mathtt{SL}_{pen}$	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
$\mathtt{SSL}_{pen}$	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**

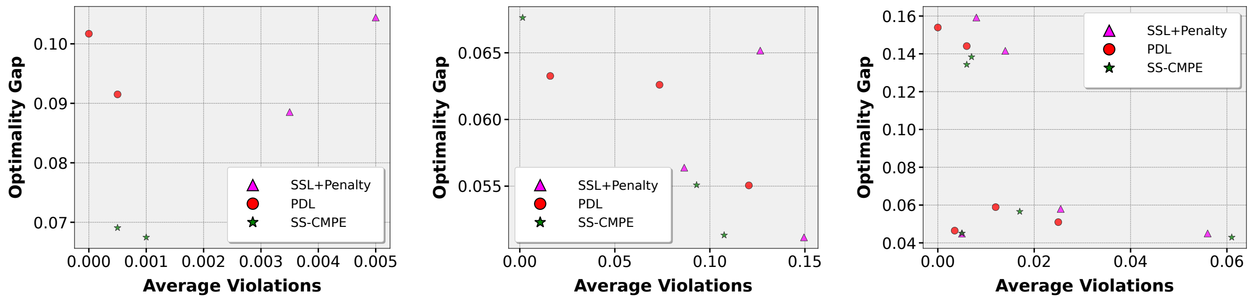
Table 6.3: Average gap and constraint violations over test samples from tractable probabilistic models. ± denotes standard deviation. Bold values indicate the methods with the highest performance. Underlined values denote significant violations, particularly those exceeding a threshold of 0.15. For these 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
$\mathtt{SL}_{pen}$	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
$\mathtt{SSL}_{pen}$	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

6.4.5 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 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. 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). Experiments in Tables 6.2 and 6.3 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. For the supervised methods, we generated the optimal assignment to $\mathbf{Y}$ using an integer linear programming solver called SCIP (Achterberg, 2009). We used 5-fold cross validation for selecting the hyperparameters.

For our proposed scheme, which we call `SS-CMPE`, we used approach described in Section 6.3.1 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.



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

Figure 6.1: Optimality Gap (avg %) and Average Violations for Self-Supervised methods

The results for the UAI datasets are shown in Table 6.2. 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 6.3, 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 6.1, 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.

6.4.6 Adversarial Modification on the MNIST Dataset

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 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 $\texttt{SL}_{pen}$, $\texttt{SSL}_{pen}$, `PDL` and `SS-CMPE`. We used the default train-test split (10K examples for testing and 60K for training).

Table 6.4: 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

Table 6.4 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 ($\texttt{SSL}_{pen}$ and `PDL`) and supervised methods ($\texttt{SL}_{pen}$) in terms of both constraint violations and optimality gap. The second best method in terms of optimality gap is $\texttt{SL}_{pen}$. 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 $\texttt{SSL}_{pen}$.

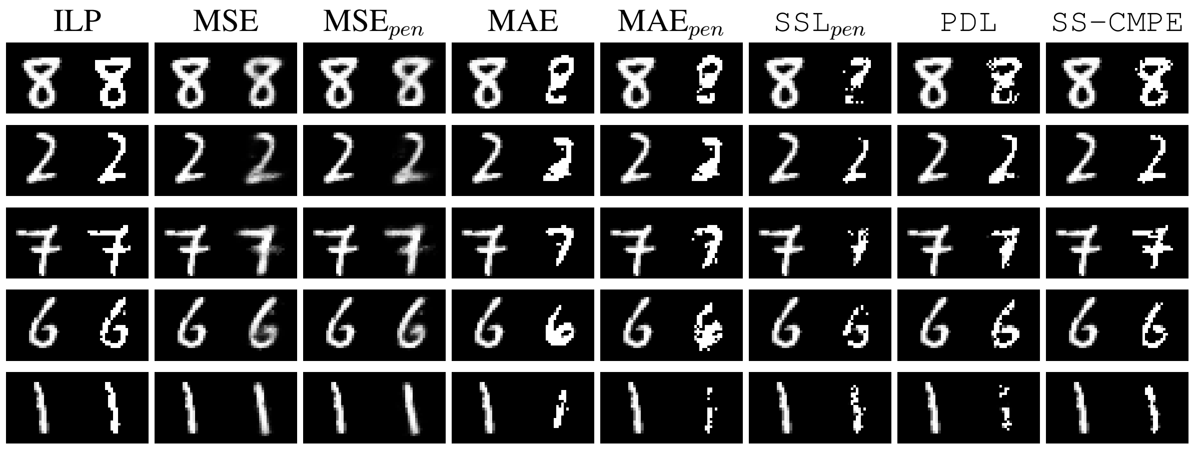


Figure 6.2: Qualitative results on the adversarially generated MNIST digits. Each row represents an original image followed by a corresponding image generated adversarially by 8 different methods: ILP, MSE, SL+Penalty, MAE, MAE+Penalty, $\texttt{SSL}_{pen}$, `PDL`, and `SS-CMPE` .

Figure 6.2 shows qualitative results on adversarial modification to the MNIST digits 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.

6.4.7 The Feasible-Only Optimality Gaps: Comparing Self-Supervised Approaches

We selected a subset of problems from the test set on which all self-supervised methods, namely, $\texttt{SSL}_{pen}$ (Donti et al., 2020), `PDL` (Park and Hentenryck, 2023) 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 6.3. 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.

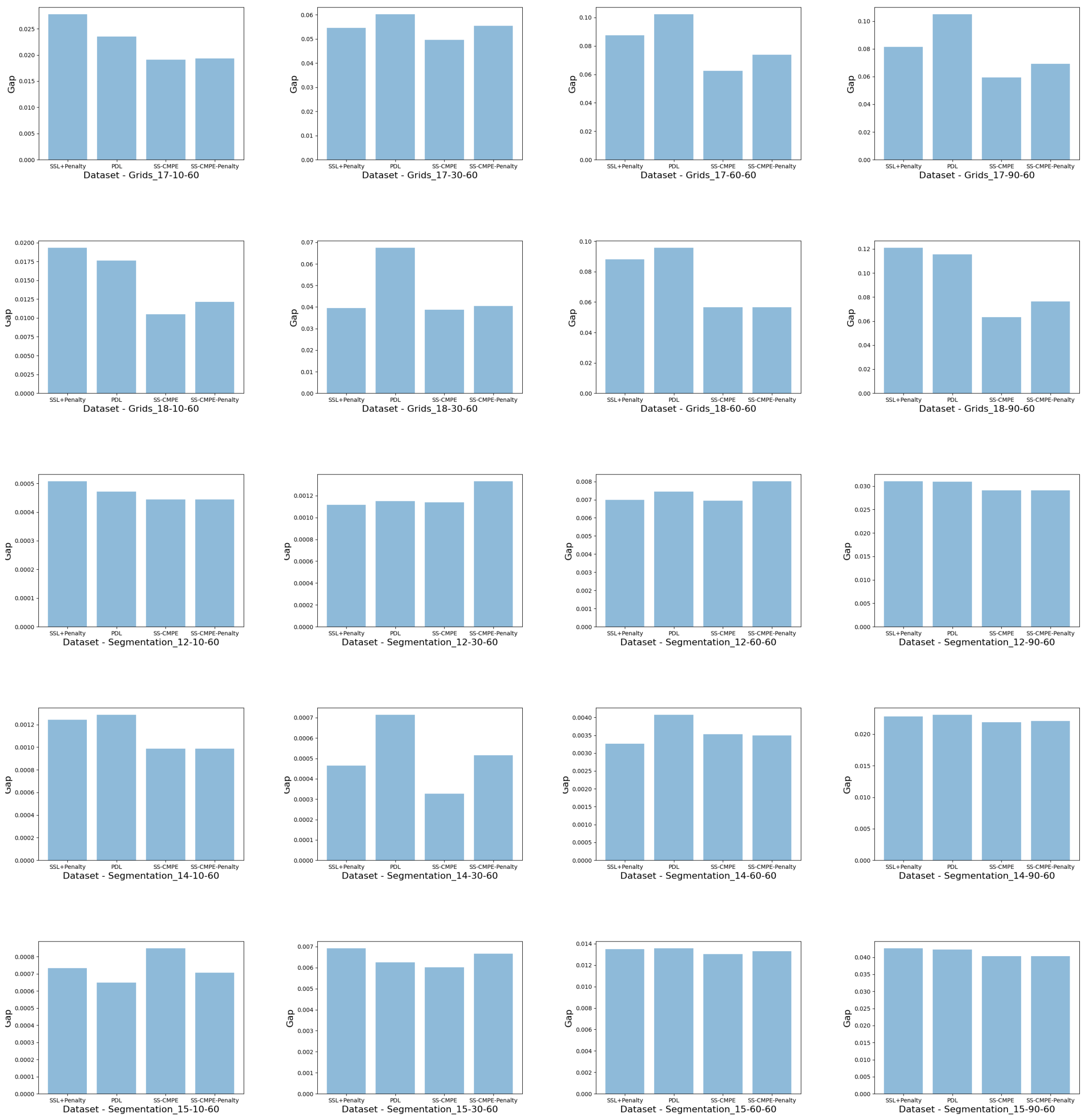
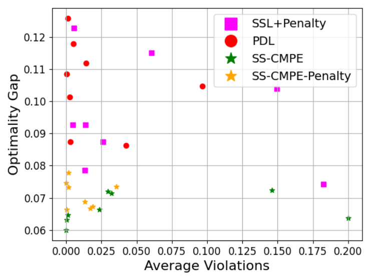
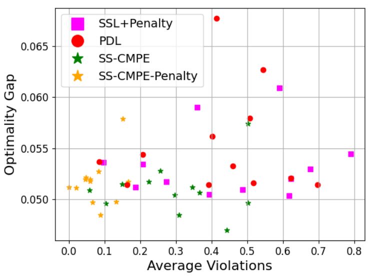


Figure 6.3: Illustration of the optimality gap for self-supervised methods (on feasible examples only) for all approaches. Lower is better.

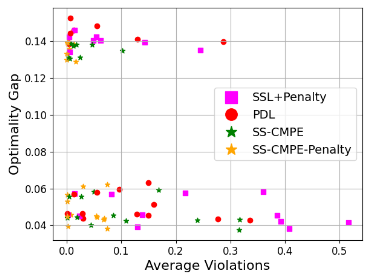
6.4.8 Optimality Gap And Violations in Self-Supervised Methods for Different q Values



(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 6.4: 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 6.4, 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 6.4(a), focused on Grids UAI networks, the 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 SS−CMPE $_{pen}$ method, with occasional high levels of violations observed in two instances.

In Figure 6.4(b), showcasing the Segmentation UAI networks, the SS−CMPE and SS−CMPE $_{pen}$ methods again demonstrate superiority, particularly evident by their prevalence near the origin. Finally, in Figure 6.4(c) related to tractable models, the 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.

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.

6.5 Chapter Summary

In this chapter, 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 $\alpha_{\mathbf{x}}(f_{\mathbf{x}}(\mathbf{y}) + g_{\mathbf{x}}(\mathbf{y}))$.

CHAPTER 7

DEEP DEPENDENCY NETWORKS AND ADVANCED INFERENCE SCHEMES FOR MULTI-LABEL CLASSIFICATION

7.1 Introduction

In this chapter, we focus on the multi-label classification (MLC) task and more specifically, on its two notable instantiations, multi-label action classification (MLAC) for videos and multi-label image classification (MLIC). At a high level, given a pre-defined set of labels (or actions) and a test example (video or image), the goal is to assign each test example to a subset of labels. It is well known that MLC is notoriously difficult because, in practice, the labels are often correlated, and thus, predicting them independently may lead to significant errors. Therefore, most advanced methods explicitly model the relationship or dependencies between the labels, using either probabilistic techniques (Wang et al., 2008; Guo and Xue, 2013; Antonucci et al., 2013; Wang et al., 2014; Tan et al., 2015; Di Mauro et al., 2016) or non-probabilistic/neural methods (Kong et al., 2013; Papagiannopoulou et al., 2015; Chen et al., 2019; Wang et al., 2021; Nguyen et al., 2021; Wang et al., 2021; Liu et al., 2021; Qu et al., 2021; Liu et al., 2022; Zhou et al., 2023; Weng et al., 2023).

To this end, motivated by approaches that combine probabilistic graphical models (PGMs) with neural networks (NNs) (Krishnan et al., 2015; Johnson et al., 2016), we jointly train a hybrid model, termed *deep dependency networks* (DDNs) (Guo and Weng, 2020), as illustrated in Figure 7.1. In a Deep Dependency Network (DDN), a conditional dependency network sits on top of a neural network. The underlying neural network transforms input data (e.g., an image) into a feature set. The dependency network (Heckerman et al., 2000) then utilizes these features to establish a local conditional distribution for each label, considering not only the features but also the other labels. Thus, at a high level, a DDN is a *neurosymbolic model* where the neural network extracts features from data and the dependency network acts as a symbolic counterpart, learning the weighted constraints between the labels.

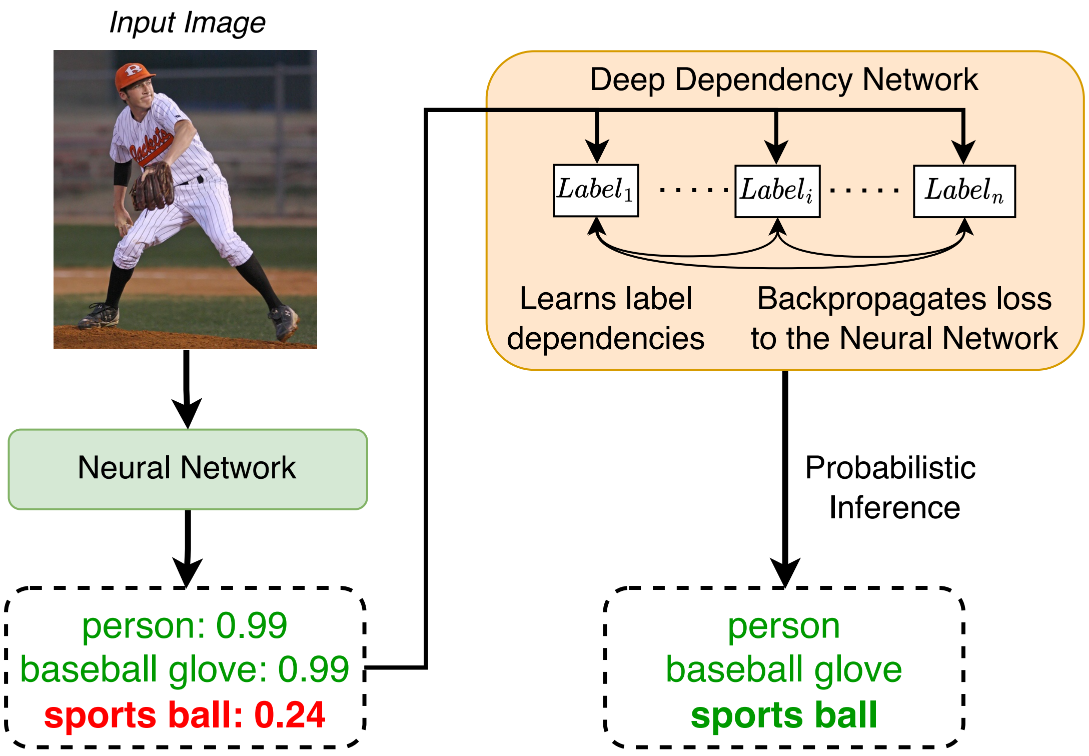


Figure 7.1: Illustrating improvements from our new inference schemes for DDNs. The DDN learns relationships between labels, and the inference schemes reason over them to accurately identify concealed objects, such as *sports ball*.

However, a limitation of DDNs is that they rely on naive techniques such as Gibbs sampling and mean-field inference for probabilistic reasoning and lack advanced probabilistic inference techniques (Lowd and Shamaei, 2011; Lowd, 2012). This chapter addresses these limitations by introducing sophisticated inference schemes tailored for the Most Probable Explanation (MPE) task in DDNs, which involves finding the most likely assignment to the unobserved variables given observations. In essence, a solution to the MPE task, when applied to a probabilistic model defined over labels and observed variables, effectively solves the multi-label classification problem.

More specifically, we propose two new methods for MPE inference in DDNs. Our first method uses a random-walk based local search algorithm. Our second approach uses a piece-wise approximation of the log-sigmoid function to convert the non-linear MPE inference problem in DDNs into an integer linear programming problem. The latter can then be solved using off-the-shelf commercial solvers such as Gurobi (Gurobi Optimization, LLC, 2023).

We evaluate DDNs equipped with our new MPE inference schemes and trained via joint learning on three video datasets: Charades (Sigurdsson et al., 2016), TACoS (Regneri et al., 2013), and Wetlab (Naim et al., 2014), and three image datasets: MS-COCO (Lin et al., 2014), PASCAL VOC 2007 (Everingham et al., 2010), and NUS-WIDE (Chua et al., 2009). We compare their performance to two categories of models: (a) basic neural networks (without dependency networks) and (b) hybrids of Markov random fields (MRFs), an undirected PGM, and neural networks equipped with sophisticated reasoning and learning algorithms. Specifically, we employ three advanced approaches: (1) iterative join graph propagation (IJGP) (Mateescu et al., 2010), a type of generalized Belief propagation method (Yedidia et al., 2000) for marginal inference, (2) integer linear programming (ILP) based techniques for computing most probable explanations (MPE) and (3) a well-known structure learning method based on logistic regression with ℓ_1-regularization (Lee et al., 2006; Wainwright et al., 2006) for pairwise MRFs.

Via a detailed experimental evaluation, we found that, generally speaking, the MRF+NN hybrids outperform NNs, as measured by metrics such as Jaccard index and subset accuracy, especially when advanced inference methods such as IJGP are employed. Additionally, DDNs, when equipped with our novel MILP-based MPE inference approach, often outperform both MRF+NN hybrids and NNs. This enhanced performance of DDNs with advanced MPE solvers is likely attributed to their superior capture of dense label interdependencies, a challenge for MRFs. Notably, MRFs rely on sparsity for efficient inference and learning.

7.2 Training Deep Dependency Networks

In this section, we describe the training procedure for the Deep Dependency Network (Guo and Weng, 2020; Arya et al., 2023), a hybrid framework designed for multi-label action classification in videos and multi-label image classification. Two key components are trained jointly: a neural network and a conditional dependency network. The neural network is responsible for extracting high-quality features from video segments or images, while the dependency network models the relationships between these features and their corresponding labels, supplying the neural network with gradient information regarding these relationships.

7.2.1 Framework

Let $\mathbf{V}$ denote the set of random variables corresponding to the pixels and $\mathbf{v}$ denote the RGB values of the pixels in a frame or a video segment. Let $\mathbf{E}$ denote the (continuous) output nodes of a neural network which represents a function $\mathcal{N} : \mathbf{v} \mapsto \mathbf{e}$, that takes $\mathbf{v}$ as input and outputs an assignment $\mathbf{e}$ to $\mathbf{E}$. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ denote the set of indicator variables representing the labels. For simplicity, we assume that $|\mathbf{E}| = |\mathbf{X}| = n$. Given $(\mathbf{V}, \mathbf{E}, \mathbf{X})$, a deep dependency network (DDN) is a pair $\langle \mathcal{N}, \mathcal{D} \rangle$ where $\mathcal{N}$ is a neural network that maps $\mathbf{V} = \mathbf{v}$ to $\mathbf{E} = \mathbf{e}$ and $\mathcal{D}$ is a conditional dependency network (Guo and Gu, 2011) that models $P(\mathbf{x}|\mathbf{e})$ where $\mathbf{e} = \mathcal{N}(\mathbf{v})$. The conditional dependency network represents the distribution $P(\mathbf{x}|\mathbf{e})$ using a collection of local conditional distributions $P_i(x_i|\mathbf{x}_{-i}, \mathbf{e})$, one for each label X_i, where $\mathbf{x}_{-i} = \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$.

Thus, a DDN is a discriminative model and represents the conditional distribution $P(\mathbf{x}|\mathbf{v})$ using several local conditional distributions $P(x_i|\mathbf{x}_{-i}, \mathbf{e})$ and makes the following conditional independence assumptions $P(x_i|\mathbf{x}_{-i}, \mathbf{v}) = P(x_i|\mathbf{x}_{-i}, \mathbf{e})$ where $\mathbf{e} = \mathcal{N}(\mathbf{v})$. Figure 7.2 demonstrates the DDN architecture.

7.2.2 Learning

We employ the conditional pseudo log-likelihood loss (CPLL) (Besag, 1975) in order to jointly train the two components of DDN, drawing inspiration from the DDN training approach outlined by Guo and Weng (2020); Arya et al. (2023). For each training example $(\mathbf{v}, \mathbf{x})$, we send the video/image through the neural network to obtain a new representation $\mathbf{e}$ of $\mathbf{v}$. In the dependency layer, we learn a classifier for each label X_i to model the conditional distribution $P_i(x_i|\mathbf{x}_{-i}, \mathbf{e})$.

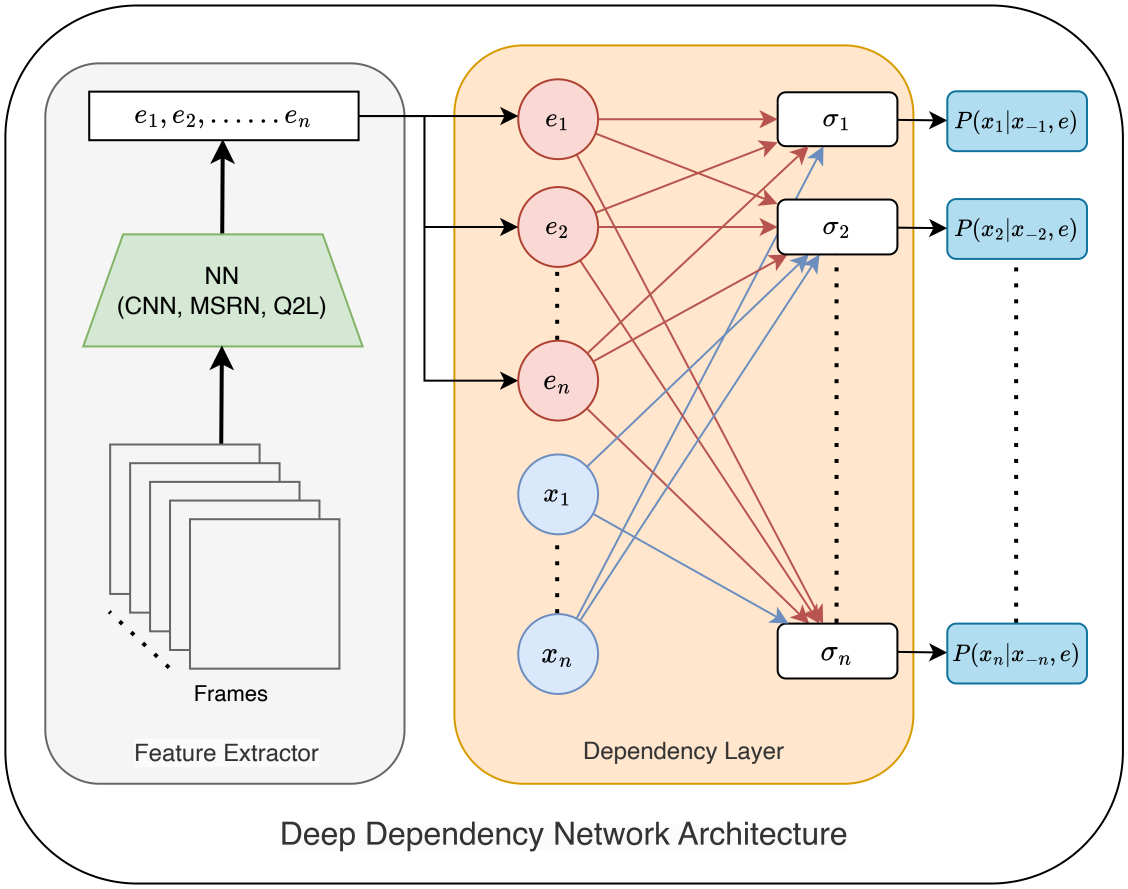


Figure 7.2: Illustration of Dependency Network for multi-label video classification. The NN takes video clips (frames) as input and outputs the features $e_1, e_2, ..., e_n$ (denoted by red colored nodes). These features are then used by the sigmoid output ($\sigma_1, \ldots, \sigma_n$) of the dependency layer to model the local conditional distributions.

More precisely, with the representation of the training instance $(\mathbf{e}, \mathbf{x})$, each sigmoid output of the dependency layer indexed by i and denoted by σ_i (see Figure 7.2) uses X_i as the class variable and $(\mathbf{E} \cup \mathbf{X}_{-i})$ as the attributes. The joint training of the model is achieved through CPLL applied to the outputs of the dependency layer (σ_i's).

Let Θ represent the parameter set of the DDN. We employ gradient-based optimization methods (e.g., backpropagation), to minimize the Conditional Pseudo-Likelihood (CPLL) loss function given below

$$\mathcal{L}(\Theta, \mathbf{v}, \mathbf{x}) = -\sum_{i=1}^{n} \log P_i(x_i|\mathbf{e} = \mathcal{N}(\mathbf{v}), \mathbf{x}_{-i}; \Theta) \tag{7.1}$$

7.3 MPE Inference In DDNs

Unlike a conventional discriminative model, such as a neural network, in a DDN, we cannot predict the output labels by simply making a forward pass over the network. This is because each sigmoid output σ_i (which yields a probability distribution over X_i) of the dependency layer requires an assignment $\mathbf{x}_{-i}$ to all labels except x_i and $\mathbf{x}_{-i}$ is not available at prediction time. Consequently, it is imperative to employ specialized techniques for obtaining output labels in multilabel classification tasks within the context of DDNs.

Given a DDN representing a distribution $P(\mathbf{x}|\mathbf{e})$ where $\mathbf{e} = \mathcal{N}(\mathbf{v})$, the multilabel classification task can be solved by finding the most likely assignment to all the unobserved variables based on the set of observed variables. This task is also called the most probable explanation (MPE) task. Formally, we seek to find $\mathbf{x}^*$ such that:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{e}) \tag{7.2}$$

Next, we present three algorithms for solving the MPE task.

7.3.1 Gibbs Sampling

To perform MPE inference, we first send the video (or frame) through the neural network to yield an assignment $\mathbf{e}$ to all variables in $\mathbf{E}$. Then given $\mathbf{e}$, we generate N samples $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})$ via Gibbs sampling (see for example Koller and Friedman (2009)), a classic MCMC technique. These samples can then be used to estimate the marginal probability distribution of each label X_i using the following mixture estimator (Liu, 2008):

$$\hat{P}_i\left(x_i|\mathbf{v}\right) = \frac{1}{N}\sum_{j=1}^{N} P_i\left(x_i \mid \mathbf{x}_{-i}^{(j)}, \mathbf{e}\right) \tag{7.3}$$

Given an estimate of the marginal distribution $\hat{P}_i\left(x_i|\mathbf{v}\right)$ for each variable X_i, we can estimate the MPE assignment using the standard max-marginal approximation:

$$\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{e}) \approx \prod_{i=1}^{n} \max_{x_i} \hat{P}(x_i|\mathbf{e})$$

In other words, we can construct an approximate MPE assignment by finding the value x_i for each variable that maximizes $\hat{P}_i\left(x_i|\mathbf{e}\right)$.

7.3.2 Local Search Based Methods

Local search algorithms (see for example Selman et al. (1993)) systematically examine the solution space by making localized adjustments, with the objective of either finding an optimal solution or exhausting a pre-established time limit. They offer a viable approach for solving the MPE inference task in DDNs. These algorithms, through their structured exploration and score maximization, are effective in identifying near-optimal label configurations.

In addressing the MPE inference within DDNs, we define the objective function for local search as $\sum_{i=1}^{n} \log\left(P_i(x_i \mid \mathbf{x}_{-i}, \mathbf{e})\right)$, where $\mathbf{e}$ is the evidence provided by $\mathcal{N}$. We propose to use two distinct local search strategies for computing the MPE assignment: random walk and greedy local search.

In *random walk (RW)*, the algorithm begins with a random assignment to the labels and at each iteration, flips a random label (changes the value of the label from a 1 to a 0 or a 0 to a 1) to yield a new assignment. At termination, the algorithm returns the assignment with the highest score explored during the random walk. In *greedy local search*, the algorithm begins with a random assignment to the labels, and at each iteration, with probability p flips a random label to yield a new assignment and with probability $1 - p$ flips a label that yields the maximum improvement in the score. At termination, the algorithm returns the assignment with the highest score explored during its execution.

7.3.3 Multi-Linear Integer Programming

In this section, we present a novel approach for MPE inference in DDNs by formulating the problem as a Second-Order Multi-Linear Integer Programming task. Specifically, we show that the task of maximizing the scoring function $\sum_{i=1}^{n} \log\left(P_i(x_i \mid \mathbf{x}_{-i}, \mathbf{e})\right)$ is equivalent to the optimization problem given in 7.10.

Given that x_i is binary, namely, $x_i \in \{0, 1\}$, we can express the scoring function as follows:

$$\underset{\mathbf{x}}{\text{maximize}} \sum_{i=1}^{n} \left(x_i \log\left(P_i(X_i = 1 \mid \mathbf{x}_{-i}, \mathbf{e})\right) + \ (1 - x_i) \log\left(1 - P_i(X_i = 1 \mid \mathbf{x}_{-i}, \mathbf{e})\right) \right) \tag{7.4}$$

where

$$P_i(X_i = 1|\mathbf{x}_{-i}, \mathbf{e}) = \sigma\left(\sum_{j=1}^{|\mathbf{e}|} w_{ij} e_j + \sum_{\substack{k=1 \\ k \neq i}}^{|\mathbf{x}|} v_{ik} x_k + b_i \right)$$

Let $p_i = P_i(X_i = 1|\mathbf{x}_{-i}, \mathbf{e})$ and $z_i = \sum_{j=1}^{|\mathbf{e}|} w_{ij} e_j + \sum_{\substack{k=1 \\ k \neq i}}^{|\mathbf{x}|} v_{ik} x_k + b_i$. Then, the MPE task, which involves optimizing the scoring function given above, can be expressed as:

$$\underset{\mathbf{x},\mathbf{z}}{\text{maximize}} \sum_{i=1}^{n} \left(x_i \log p_i + (1 - x_i) \log(1 - p_i) \right) \tag{7.5}$$

subject to:

$$p_i = \sigma(z_i), \quad \forall i \in \{1, \dots, n\} \tag{7.6}$$

$$z_i = \sum_{j=1}^{|\mathbf{e}|} w_{ij} e_j + \sum_{\substack{k=1 \\ k \neq i}}^{|\mathbf{x}|} v_{ik} x_k + b_i, \quad \forall i \in \{1, \dots, n\} \tag{7.7}$$

$$x_i \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\} \tag{7.8}$$

Here, w_{ij}'s and v_{ik}'s denote the weights associated with $\mathbf{e}$ and $\mathbf{x}$ for P_i, respectively. The bias term for each P_i is denoted by b_i. In this context, z_i represents the values acquired prior to applying the sigmoid activation function. The first constraint in the formulation, as expressed by 7.6, pertains to the sigmoid function employed in the logistic regression module. The subsequent constraint, defined by 7.7, formalizes the product between the weights and inputs in the conditional DN. Here, e_i represents evidence values supplied to the DN, while $\mathbf{x}$ represents the decision variables that are subject to optimization. Finally, 7.8 specifies that the input variables must be constrained to integer values of 0 or 1. The constraints collectively simulate a forward pass through the network. The objective function is designed to maximize the scoring function pertinent to the Most Probable Explanation (MPE).

The substitution of the constraint $P_i(x_i|\mathbf{x}_{-i}, \mathbf{e}) = \sigma(z_i) = \frac{1}{1+e^{(-z_i)}}$ into the objective function and the subsequent algebraic simplification result in a simplified formulation.

$$\begin{aligned}
&x_i \log p_i + (1 - x_i) \log(1 - p_i) \\
&= x_i \log \left(\frac{e^{z_i}}{1 + e^{z_i}} \right) + (1 - x_i) \log \left(1 - \frac{e^{z_i}}{1 + e^{z_i}} \right) \\
&= x_i \log \left(\frac{e^{z_i}}{1 + e^{z_i}} \right) + (1 - x_i) \log \left(\frac{1}{1 + e^{z_i}} \right) \\
&= x_i \log e^{z_i} - x_i \log(1 + e^{z_i}) - \log(1 + e^{z_i}) + x_i \log(1 + e^{z_i}) \\
&= x_i \log e^{z_i} - \log(1 + e^{z_i}) \\
&= x_i z_i - \log(1 + e^{z_i})
\end{aligned} \tag{7.9}$$

Substituting equation 7.9 in the objective function of equation 7.5, we get

$$\underset{\mathbf{x},\mathbf{z}}{\text{maximize}} \sum_{i=1}^{n} x_i z_i - \log\left(1 + e^{z_i}\right) \tag{7.10}$$

subject to:

$$z_i = \sum_{j=1}^{|\mathbf{e}|} w_{ij} e_j + \sum_{\substack{k=1 \\ k \neq i}}^{|\mathbf{x}|} v_{ik} x_k + b_i, \quad \forall i \in \{1, \ldots, n\} \tag{7.11}$$

$$x_i \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\} \tag{7.12}$$

The optimal value for $\mathbf{x}$ corresponds to the solution of this optimization problem. The second term in the objective specified in equation 7.10 comprises logarithmic and exponential functions, which are non-linear, thereby making it a *non-linear optimization problem*. To address this non-linearity, we propose the use of piece-wise linear approximations(Lin et al., 2013; Geißler et al., 2012; Kumar, 2007; Li et al., 2022; Asghari et al., 2022; Rovatti et al., 2014) for these terms.

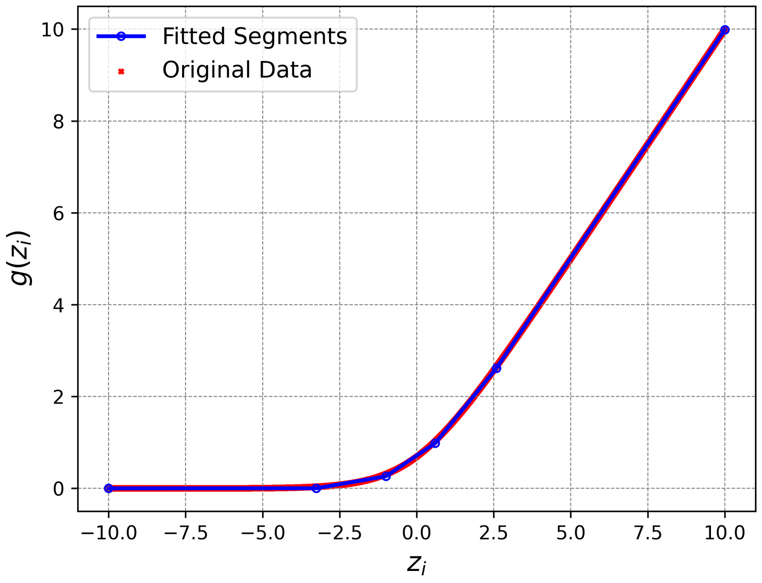


Figure 7.3: Piece-wise linear approximation of $log(1 + e^{z_i})$

Given the objective function's non-linearity, particularly due to the term $\log(1+e^{z_i})$, we utilize a piecewise linear function, $g(z)$, as an approximation for $\log(1+e^{z_i})$. The function $g(z)$ is defined as follows:

$$g(z) = \begin{cases} z & z \gg 1 \\ e^z & z \ll 1 \\ \log(1+e^z) & \text{otherwise} \end{cases} \tag{7.13}$$

To obtain a piecewise linear approximation of $\log(1 + e^{z_i})$, a single linear function suffices for $z \gg 1$, while the majority of the linear pieces are utilized for approximating the function near 1. A piecewise linear approximation of the function is detailed in the Figure 7.3. We employ five segments for the approximation. These piecewise functions can be integrated as linear constraints, facilitating the conversion of the non-linear objective into a linear objective with ease. We also present the piecewise equations for the approximation as follows -

Table 7.1: Piecewise Linear Approximation for $g(z) \approx \log(1 + e^{z_i})$

z	$g(z)$
$]-\infty, -3.257)$	0
$[-3.257, -0.998)$	$(2^{-4} + 2^{-5} + 2^{-6} + 2^{-8})z + 0.379$
$[-0.998, 0.602)$	$(2^{-2} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8})z + 0.715$
$[0.602, 2.584)$	$(2^{-1} + 2^{-2} + 2^{-4} + 2^{-7})z + 0.492$
$[2.584, +\infty[$	z

The optimization problem can be restated by incorporating $g(z_i)$ as a piece-wise linear approximation of $\log(1 + e^{z_i})$. This modification allows for the representation as follows:

$$\begin{aligned} &\underset{\mathbf{x},\mathbf{z}}{\text{maximize}} \sum_{i=1}^{n} x_i z_i - g(z_i) \\ &\text{subject to:} \\ &z_i = \sum_{j=1}^{|\mathbf{e}|} w_{ij} e_j + \sum_{\substack{k=1 \\ k \neq i}}^{|\mathbf{x}|} v_{ik} x_k + b_i, \quad \forall i \in \{1, \ldots, n\} \\ &x_i \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\} \end{aligned} \tag{7.14}$$

The optimization problem given in equation 7.14 is an integer multilinear program of order 2 because it includes terms of the form $x_i x_j$ where both x_i and x_j take values from the set $\{0, 1\}$. Since $x_i x_j$ corresponds to a "logical and" between two Boolean variables, all such expressions can be easily encoded as linear constraints yielding an integer linear program (ILP).

To illustrate the encoding of binary variable products into linear constraints within an integer linear program (ILP), consider an optimization problem involving three binary variables X_1, X_2, and X_3, with x_1, x_2, and x_3 representing their respective assignments. We formulate an objective function that includes the products of these binary variables:

$$\max_{x_1,x_2,x_3} \; x_1x_2 - x_2x_3 + x_1x_3. \tag{7.15}$$

While this example does not include constraints, the methodology can be adapted to constrained optimization problems. To linearize the products of binary variables, we introduce auxiliary variables z_1, z_2, and z_3 to represent each binary product. The revised optimization problem is thus formulated as follows:

$$\begin{aligned} \max_{x_1,x_2,x_3} \quad & z_1 - z_2 + z_3 \\ \text{s.t.} \quad & x_1 \wedge x_2 = z_1, \\ & x_2 \wedge x_3 = z_2, \\ & x_1 \wedge x_3 = z_3. \end{aligned} \tag{7.16}$$

The optimization problem may be expressed by incorporating additional linear constraints to represent each boolean product. The resulting problem is presented as follows:

$$\begin{aligned} \max_{x_1,x_2,x_3} \quad & z_1 - z_2 + z_3 \\ \text{s.t.} \quad & x_1 + x_2 - 1 \leq z_1, \\ & z_1 \leq x_1, \\ & z_1 \leq x_2, \\ & x_2 + x_3 - 1 \leq z_2, \\ & z_2 \leq x_2, \\ & z_2 \leq x_3, \\ & x_1 + x_3 - 1 \leq z_3, \\ & z_3 \leq x_1, \\ & z_3 \leq x_3. \end{aligned} \tag{7.17}$$

The outlined formulation adeptly captures logical AND operations between binary variables through linear constraints, thus facilitating the application of Mixed-Integer Linear Programming (MILP) solvers to identify the optimal solution. In our experiments, we solved the ILP using Gurobi (Gurobi Optimization, LLC, 2023), which is an anytime ILP solver. Another useful feature of the ILP formulation is that we can easily incorporate prior knowledge (e.g., an image may not have more than ten objects/labels) into the ILP under the assumption that the knowledge can be reliably modeled using linear constraints.

7.4 Experimental Evaluation

In this section, we evaluate the proposed methods on two multi-label classification tasks: (1) multi-label activity classification using three video datasets; and (2) multi-label image classification using three image datasets. We begin by describing the datasets and metrics, followed by the experimental setup, and conclude with the results. All models were implemented utilizing PyTorch and were trained and evaluated on a machine with an NVIDIA A40 GPU and an Intel(R) Xeon(R) Silver 4314 CPU.

7.4.1 Datasets and Metrics

We evaluated our algorithms on three video datasets: (1) Charades (Sigurdsson et al., 2016); (2) TACoS (Regneri et al., 2013); and (3) Wetlab (Naim et al., 2015). Charades dataset comprises videos of people performing daily indoor activities while interacting with various objects. We adopted the train-test split instructions given in PySlowFast (Fan et al., 2020) with 7,986 training and 1,863 validation videos. TACoS consists of third-person videos of a person cooking in a kitchen. The dataset comes with hand-annotated labels of actions, objects, and locations for each video frame. From the complete set of these labels, we selected 28 labels resulting in 60,313 training frames and 9,355 test frames across 17 videos. Wetlab features experimental activities in labs, consisting of five training videos (100,054 frames) and one test video (11,743 frames) with 57 distinct labels.

For multi-label image classification (MLIC), we examined: (1) MS-COCO (Lin et al., 2014); (2) PASCAL VOC 2007 (Everingham et al., 2010); and (3) NUS-WIDE (Chua et al., 2009). MS-COCO, a well-known dataset for detection and segmentation, comprises 122,218 labeled images with an average of 2.9 labels per image. We used its 2014 version. NUS-WIDE is a real-world web image dataset that contains 269,648 images from Flickr. Each image has been manually annotated with a subset of 81 visual classes that include objects and scenes. PASCAL VOC 2007 contains 5,011 train-validation and 4,952 test images, with each image labeled with one or more of the 20 available object classes.

We follow the instructions provided in Qu et al. (2021) to do the train-test split for NUS-WIDE and PASCAL VOC. We evaluated performance on the TACoS, Wetlab, MS-COCO, NUS-WIDE, and VOC datasets using Subset Accuracy (SA), Jaccard Index (JI), Hamming Loss (HL), Macro F1 Score (Macro F1), Micro F1 Score (Micro F1) and F1 Score (F1). With the exception of Hamming Loss, superior performance is indicated by higher scores in all considered metrics. We omit SA for the Charades dataset due to the infeasibility of achieving reasonable scores given its large label space. Additionally, Hamming Loss (HL) has been excluded for the MS-COCO dataset as the performance of all methods is indistinguishable.

Given that the focus in MPE inference is on identifying the most probable label configurations rather than individual label probabilities, the use of Mean Average Precision (mAP) as a performance metric is not applicable to our study. Therefore, our primary analysis relies on SA, JI, HL, Macro F1, Micro F1, and F1.

7.4.2 Experimental Setup and Methods

We used three types of architectures in our experiments: (1) Baseline neural networks, which are specific to each dataset; (2) neural networks augmented with MRFs, which we will refer to as deep random fields or DRFs in short; and (3) a dependency network on top of the neural networks called deep dependency networks (DDNs).

Neural Networks. We choose four different types of neural networks, and they act as a baseline for the experiments and as a feature extractor for DRFs and DDNs. Specifically, we experimented with: (1) 2D CNNs, (2) 3D CNNs, (3) transformers, and (4) CNNs with attention module and graph attention networks (GAT) (Velickovic et al., 2018). This helps us show that our proposed method can improve the performance of a wide variety of neural architectures, even those which model label relationships, because unlike the latter, it performs probabilistic inference.

For Charades dataset, we use the PySlowFast (Fan et al., 2020) implementation of the SlowFast Network (Feichtenhofer et al., 2019) (a state-of-the-art 3D CNN for video classification). For TACoS and Wetlab datasets, we use InceptionV3 (Szegedy et al., 2016), a state-of-the-art 2D CNN model for image classification. For the MS-COCO dataset, we used Query2Label (Q2L) (Liu et al., 2021), which uses transformers to pool class-related features. Q2L also learns label embeddings from data to capture the relationships between the labels. Finally, we used the multi-layered semantic representation network (MSRN) (Qu et al., 2021) for NUS-WIDE and PASCAL VOC. MSRN also models label correlations and learns semantic representations at multiple convolutional layers. We adopt pre-trained models and hyper-parameters from existing repositories for Charades, MS-COCO, NUS-WIDE, and PASCAL VOC. For TaCOS and Wetlab datasets, we fine-tuned an InceptionV3 model that was pre-trained on the ImageNet dataset.

Deep Random Fields (DRFs). As a baseline, we used a model that combines MRFs with neural networks. This DRF model is similar to DDN except that we use an MRF instead of a DN to compute $P(\mathbf{x}|\mathbf{e})$. We trained the MRFs generatively; namely, we learned a joint distribution $P(\mathbf{x}, \mathbf{e})$, which can be used to compute $P(\mathbf{x}|\mathbf{e})$ by instantiating evidence. We chose generative learning because we learned the structure of the MRFs from data, and discriminative structure learning is slow in practice (Koller and Friedman, 2009). Specifically, we used the logistic regression with ℓ_1 regularization method of (Wainwright et al., 2006) to learn a pairwise MRF. The training data for this method is obtained by sending each annotated video clip (or frame) $(\mathbf{v}, \mathbf{x})$ through the neural network and transforming it to $(\mathbf{e}, \mathbf{x})$ where $\mathbf{e} = \mathbb{N}(\mathbf{v})$. At termination, this method yields a graph $\mathcal{G}$ defined over $\mathbf{X} \cup \mathbf{E}$.

For parameter/weight learning, we converted each edge over $\mathbf{X} \cup \mathbf{E}$ to a conjunctive feature. For example, if the method learns an edge between X_i and E_j, we use a conjunctive feature $X_i \wedge E_j$ which is true if both X_i and E_j are assigned the value 1. Then we learned the weights for each feature by maximizing the pseudo log-likelihood of the data.

For inference over MRFs, we used Gibbs sampling (GS), Iterative Join Graph Propagation (IJGP) (Mateescu et al., 2010), and Integer Linear Programming (ILP) methods. Thus, three versions of DRFs corresponding to the inference scheme were used. We refer to these schemes as DRF-GS, DRF-ILP, and DRF-IJGP, respectively. Note that IJGP and ILP are advanced schemes, and we are unaware of their use for multi-label classification. Our goal is to test whether advanced inference schemes help improve the performance of deep random fields.

Deep Dependency Networks (DDNs). We trained the Deep Dependency Networks (DDNs) using the joint learning loss described in equation 7.1. We examined four unique inference methods for DDNs: (1) DDN-GS, employing Gibbs Sampling; (2) DDN-RW, leveraging a random walk local search; (3) DDN-Greedy, implementing a greedy local search; and (4) DDN-ILP, utilizing Integer Linear Programming to optimize the objective given in equation 7.14. It is noteworthy that, until now, only DDN-GS has been used for inference in Dependency Networks. DDN-RW and DDN-Greedy, which are general-purpose local search techniques, are our proposals for MPE inference within Dependency Networks. Lastly, DDN-ILP introduces a novel approach using optimization techniques with the objective of enhancing MPE inference on dependency networks.

Hyperparameters. For DRFs, in order to learn a sparse structure (using the logistic regression with ℓ_1 regularization method of Wainwright et al. (2006)), we increased the regularization constant associated with the ℓ_1 regularization term until the number of neighbors of each node in $\mathcal{G}$ is bounded between 2 and 10. We enforced this sparsity constraint in order to ensure that the inference schemes, specifically IJGP and ILP, are accurate and the model does not overfit the training

Table 7.2: Comparison of our methods with the feature extractor for MLAC. The best/second best values are bold/underlined.

Dataset	Metric	SlowFast	InceptionV3	DRF			DDN			
				GS	ILP	IJGP	GS	RW	Greedy	ILP
Charades	JI	0.29	-	0.22	0.31	0.32	0.31	0.30	0.31	**0.33**
	HL $\downarrow$	**0.052**	-	0.194	0.067	0.054	0.054	0.056	0.056	**0.052**
	Macro F1	0.32	-	0.16	0.18	0.19	0.30	0.13	0.33	**0.36**
	Micro F1	0.45	-	0.31	0.21	0.29	0.43	0.24	0.44	**0.47**
	F1	0.42	-	0.28	0.22	0.26	0.41	0.22	0.41	**0.44**
TACoS	SA	-	0.40	0.47	0.51	0.44	0.54	0.46	0.56	**0.63**
	JI	-	0.61	0.65	0.65	0.70	0.69	0.64	0.69	**0.72**
	HL $\downarrow$	-	0.082	0.044	**0.030**	0.043	0.041	0.042	0.040	0.040
	Macro F1	-	0.26	0.54	0.42	0.54	0.52	0.59	0.61	**0.62**
	Micro F1	-	0.48	0.75	0.73	**0.83**	0.74	0.77	0.80	0.79
	F1	-	0.44	0.70	0.66	**0.78**	0.68	0.75	0.75	0.76
Wetlab	SA	-	0.35	0.35	0.60	0.58	0.63	0.46	0.55	**0.65**
	JI	-	0.64	0.52	0.73	0.74	**0.78**	0.63	0.68	0.76
	HL $\downarrow$	-	0.017	0.058	0.014	0.014	**0.011**	0.025	0.014	0.014
	Macro F1	-	0.24	0.22	0.27	0.24	**0.28**	0.22	**0.28**	**0.28**
	Micro F1	-	0.76	0.69	0.81	**0.82**	0.80	0.66	0.80	0.81
	F1	-	0.72	0.68	0.77	0.77	0.79	0.68	0.79	**0.80**

data. IJGP, ILP, and GS are anytime methods; for them, we used a time-bound of one minute per example.

For DDNs, we used ℓ_1 regularization for the dependency layer. Through cross-validation, we selected the regularization constant from the set $\{0.1, 0.01, 0.001\}$. In the context of joint learning, the learning rates of the DDN model were adjusted using an extended version of the learning rate scheduler from PySlowFast (Fan et al., 2020). We impose a strict time constraint of 60 seconds per example for DDN-GS, DDN-RW, DDN-Greedy, and DDN-ILP. The DDN-ILP method is solved using Gurobi Optimization, LLC (2023) and utilizes accurate piece-wise linear approximations for non-linear functions, subject to a pre-defined error tolerance of 0.001.

Table 7.3: Comparison of our methods with the feature extractor for MLIC. The best/second best values are bold/underlined.

Dataset	**Metric**	**Q2L**	**MSRN**	**DRF**			**DDN**			
				GS	**ILP**	**IJGP**	**GS**	**RW**	**Greedy**	**ILP**
MS-C	SA	0.51	-	0.35	$\underline{0.54}$	**0.55**	**0.55**	0.53	**0.55**	**0.55**
	JI	0.80	-	0.69	$\underline{0.82}$	$\underline{0.82}$	$\underline{0.82}$	0.81	$\underline{0.82}$	**0.83**
	Macro F1	**0.86**	-	0.81	0.84	$\underline{0.85}$	**0.86**	0.82	$\underline{0.85}$	$\underline{0.85}$
	Micro F1	$\underline{0.86}$	-	0.82	0.83	0.85	**0.87**	0.85	$\underline{0.86}$	$\underline{0.86}$
	F1	**0.88**	-	0.79	0.82	0.85	**0.88**	0.86	$\underline{0.87}$	**0.88**
N-W	SA	-	0.31	0.28	$\underline{0.32}$	$\underline{0.32}$	**0.33**	0.25	0.30	**0.33**
	JI	-	$\underline{0.64}$	0.55	0.59	0.63	0.62	0.56	0.61	**0.65**
	HL $\downarrow$	-	**0.015**	0.020	$\underline{0.016}$	0.017	$\underline{0.016}$	$\underline{0.016}$	$\underline{0.016}$	**0.015**
	Macro F1	-	**0.56**	0.23	0.28	0.32	$\underline{0.52}$	0.26	0.29	0.51
	Micro F1	-	$\underline{0.73}$	0.63	0.70	0.71	0.71	0.70	0.72	**0.74**
	F1	-	**0.71**	0.62	0.67	$\underline{0.69}$	$\underline{0.69}$	0.68	$\underline{0.69}$	**0.71**
P-V	SA	-	0.71	0.73	0.76	0.76	0.76	0.82	$\underline{0.86}$	**0.89**
	JI	-	0.85	0.83	0.88	0.87	0.87	0.89	$\underline{0.91}$	**0.95**
	HL $\downarrow$	-	0.015	0.021	0.019	0.019	0.008	**0.006**	$\underline{0.007}$	**0.006**
	Macro F1	-	0.89	0.75	0.81	0.77	0.94	0.94	$\underline{0.95}$	**0.96**
	Micro F1	-	0.90	0.85	0.87	0.86	0.94	0.94	$\underline{0.95}$	**0.96**
	F1	-	0.91	0.83	0.87	0.86	$\underline{0.96}$	0.93	$\underline{0.96}$	**0.97**

7.4.3 Results

We compare the baseline neural networks with three versions of DRFs and four versions of DDNs using the six metrics and six datasets given in Section 7.4.1. The results are presented in Tables 7.2 and 7.3.

Comparison between baseline neural network and DRFs. We observe that IJGP and ILP outperform the baseline neural networks (which include transformers for some datasets) in terms of JI, SA, and HL on four out of the six datasets. IJGP typically outperforms GS and ILP on JI. In terms of F1 metrics, IJGP and ILP methods tend to perform better than Gibbs Sampling (GS) approaches, but they are less effective than baseline NNs. ILP's superiority in SA—a metric that scores 1 for an exact label match and 0 otherwise—can be attributed to its precise most probable explanation (MPE) inference. An accurate MPE inference, when paired with a precise model, is likely to achieve high SA scores. Note that getting a higher SA is much harder in datasets having large number of labels. Specifically, SA does not distinguish between models that predict *almost* correct labels and completely incorrect outputs. We observe that advanced inference schemes, particularly IJGP and ILP, are superior on average to GS.

Comparison between baseline neural networks and DDNs. Our study has shown that the DDN model with the proposed MILP based inference method is superior to the baseline neural networks in four out of six datasets, with notable enhancements in SA and JI metrics, such as an 18%, 23%, and 30% improvement in SA for the PASCAL-VOC, TACoS, and Wetlab datasets, respectively. Moreover, the MILP-based method either significantly surpasses or matches all other inference strategies for DDNs. While Gibbs Sampling-based inference is typically better than the Random Walk based approach in DDNs, its performance against the greedy sampling method varies.

We observe that on the MLIC task, the DDN with advanced MPE inference outperforms Q2L and MSRN, even though both Q2L and MSRN model label correlations. This suggests that DDNs are either able to uncover additional relationships between labels during the learning phase or better reason about them during the inference phase or both. In particular, both Q2L and MSRN do not use MPE inference to predict the labels because they do not explicitly model the joint probability distribution over the labels.

Figure 7.4 displays the images and their predicted labels by both Q2L and DDN-ILP on the MS-COCO dataset. Our method not only adds the labels omitted by Q2L but also eliminates several incorrect predictions. In the first two images, our approach rectifies label omissions by Q2L, conforming to the ground truth. In the third image, our method removes erroneous predictions. The last image illustrates a case where DDN underperforms compared to Q2L by failing to identify a ground-truth label.

Comparison between DRFs and DDNs. Based on our observations, we found that jointly trained DDNs in conjunction with the proposed inference method consistently lead to superior performance compared to the top-performing DRFs across all datasets. Nonetheless, in certain

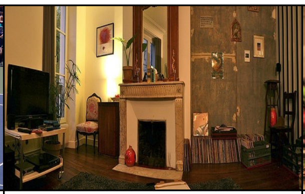
Image				
Ground Truth	person, sports ball, tennis racket, chair, clock	chair, potted plant, tv, remote, book, vase	person, bottle, pizza, clock	bird, skateboard, couch
Q2L	person (1.00), tennis racket (1.00), chair (0.96), clock (0.95), **[sports ball (0.33)]**	chair (0.99), potted plant (0.99), tv (1.00), book (1.00), vase (0.96), [**remote (0.25)**]	person (1.00), bottle (0.99), pizza (1.00), **potted plant (0.74)**, clock (0.77), **vase (0.80),**	bird (1.00), skateboard (1.00), couch (0.73)
DDN	person, sports ball, tennis racket, chair, clock	chair, potted plant, tv, remote, book, vase	person, bottle, pizza, clock	bird, skateboard

Figure 7.4: Comparison of labels predicted by Q2L (Liu et al., 2021) and our DDN-ILP scheme on the MS-COCO dataset. Labels in bold represent the difference between the predictions of the two methods, assuming that a threshold of 0.5 is used (i.e., every label whose probability > 0.5 is considered a predicted label). Due to the MPE focus in DDN-ILP, only label configurations are generated, omitting corresponding probabilities. The first three column shows examples where DDN improves over Q2L, while the last column (outlined in red) shows an example where DDN is worse than Q2L.

situations, DRFs that employ advanced inference strategies produce results that closely match those of DDNs for both JI and SA, making DRFs a viable option, particularly when limited GPU resources are available for training and optimization of both JI and SA is prioritized.

In summary, the empirical results indicate that Deep Dependency Networks (DDNs) equipped with advanced inference strategies consistently outperform conventional neural networks and MRF + NN hybrids in multi-label classification tasks. Furthermore, these advanced inference methods surpass traditional sampling-based techniques for DDNs. The superior performance of both DDNs and DRFs utilizing advanced inference techniques supports the value of such mechanisms and suggests that further advancement in this area has the potential to unlock additional capabilities within these models.

7.5 Related Work

A large number of methods have been proposed that train PGMs and NNs jointly. For example, Zheng et al. (2015) proposed to combine conditional random fields (CRFs) and recurrent neural networks (RNNs), Schwing and Urtasun (2015); Larsson et al. (2017, 2018); Arnab et al. (2016) showed how to combine CNNs and CRFs, Chen et al. (2015) proposed to use densely connected graphical models with CNNs, and Johnson et al. (2016) combined latent graphical models with neural networks. The combination of PGMs and NNs has also been applied to improve performance on a wide variety of real-world tasks. Notable examples include human pose estimation (Tompson et al., 2014; Liang et al., 2018; Song et al., 2017; Yang et al., 2016), semantic labeling of body parts (Kirillov et al., 2016), stereo estimation (Knöbelreiter et al., 2017), language understanding (Yao et al., 2014), face sketch synthesis (Zhu et al., 2021) and crowd-sourcing aggregation (Li et al., 2021)). These hybrid models have also been used for solving a range of computer vision tasks such as semantic segmentation (Arnab et al., 2018; Guo and Dou, 2021), image crowd counting (Han et al., 2017), visual relationship detection (Yu et al., 2022), modeling for epileptic seizure detection (Craley et al., 2019), face sketch synthesis (Zhang et al., 2020), semantic image segmentation (Chen et al., 2018; Lin et al., 2016), 2D Hand-pose Estimation (Kong et al., 2019), depth estimation from a single monocular image (Liu et al., 2015), animal pose tracking (Wu et al., 2020) and pose estimation (Chen and Yuille, 2014).

To date, dependency networks have been used to solve various tasks such as collective classification (Neville and Jensen, 2003), binary classification (Gámez et al., 2006, 2008), multi-label classification (Guo and Gu, 2011), part-of-speech tagging (Tarantola and Blanc, 2002), relation prediction (Figueiredo et al., 2021), text classification (Guo and Weng, 2020) and collaborative filtering (Heckerman et al., 2000). However, DDNs have traditionally been restricted to Gibbs sampling (Heckerman et al., 2000) and mean-field inference (Lowd and Shamaei, 2011), showing limited compatibility with advanced probabilistic inference methods (Lowd, 2012). This study marks the inaugural attempt to incorporate advanced inference methods for the MPE task in DDNs, utilizing jointly trained networks for MLC scenarios.

7.6 Chapter Summary

More and more state-of-the-art methods for challenging applications of computer vision tasks usually use deep neural networks. Deep neural networks are good at extracting features in vision tasks like image classification, video classification, object detection, image segmentation, and others. Nevertheless, for more complex tasks involving multi-label classification, these methods cannot model crucial information like inter-label dependencies and infer about them. In this work, we present novel inference algorithms for Deep Dependency Networks (DDNs), a powerful neurosymbolic approach, that exhibit consistent superiority compared to traditional neural network baselines, sometimes by a substantial margin. These algorithms offer an improvement over existing Gibbs Sampling-based inference schemes used for DDNs, without incurring significant computational burden. Importantly, they have the capacity to infer inter-label dependencies that are commonly overlooked by baseline techniques utilizing transformers, attention modules, and Graph Attention Networks (GAT). By formulating the inference procedure as an optimization problem, our approach permits the integration of domain-specific constraints, resulting in a more knowledgeable and focused inference process. In particular, our optimization-based approach furnishes a robust and computationally efficient mechanism for inference, well-suited for handling intricate multi-label classification tasks.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this chapter, we conclude the dissertation by summarizing our key contributions and describing the directions for future research.

This dissertation addresses the challenge of efficient and accurate inference in Probabilistic Models (PMs), focusing on three core tasks: *Most Probable Explanation* (MPE), *Constrained Most Probable Explanation* (CMPE), and *Marginal maximum-a-posteriori* (MMAP). We perform these tasks over various PMs, including Probabilistic Graphical Models (PGMs), Probabilistic Circuits (PCs), and Neural Autoregressive Models (NAMs). These inference tasks are NP-hard in general, and existing solvers either trade accuracy for speed or face scalability limitations.

To address these challenges, we proposed a suite of neural network-based solvers leveraging self-supervised learning. Our solvers enable real-time inference over large, complex probabilistic models, even for arbitrary evidence-query partitions and subject to additional domain constraints. In addition to neural solvers, we introduced optimization-based schemes for inference within neurosymbolic models, such as deep dependency networks. Our comprehensive empirical evaluation demonstrated the effectiveness of these methods across diverse datasets, tasks, and probabilistic representations.

Limitations in State-of-the-Art Methods:

- Exact solvers for MPE, CMPE, and MMAP become computationally infeasible as model size and complexity grow.
- Existing approximate solvers struggle to achieve sufficient accuracy, particularly in autoregressive models, where inference relies on sampling strategies such as beam search and hill climbing.

- The CMPE problem is particularly challenging due to its dense global constraints, making it especially difficult for off-the-shelf solvers such as Gurobi and SCIP to handle efficiently, particularly for large-scale instances.

- Deep dependency networks (DDNs) lack advanced inference techniques, depending on Gibbs sampling or mean-field approximations, which limits their reasoning power.

8.1 Contributions of this Dissertation

Efficient Neural Solvers for Unconstrained Inference Tasks over PMs

We proposed a self-supervised neural approach to answer MPE and MMAP queries over a broad class of PMs, including PCs, PGMs, and NAMs. First, we introduced a novel encoding scheme to represent fixed evidence-query configurations, paired with a tractable self-supervised loss directly derived from the underlying probabilistic model. Next, to further improve accuracy in the any-partition setting, where evidence and query variables change across queries, we developed Inference Time Self Supervised Training (ITSELF), an inference-time optimization procedure that iteratively refines predicted solutions, and GUided Iterative Dual LEarning with Self-supervised Teacher ($\mathcal{GUIDE}$), a stabilized training method based on a teacher-student framework. Finally, we proposed enhanced encoding schemes for PGMs and more efficient discretization strategies, effectively removing the need for inference-time optimization when answering arbitrary queries over these models. Our experimental results demonstrated state-of-the-art performance in both accuracy and speed, significantly outperforming polynomial-time baselines.

Constrained Inference over Log-Linear Models using Self-Supervised Neural Solvers

We extended our neural framework to answer *Constrained Most Probable Explanation* (CMPE) queries, which requires finding the most probable explanation subject to user-specified constraints.

We proposed a novel self-supervised loss function that ensures feasible solutions satisfy the constraints while maximizing the objective. Unlike prior approaches relying on Lagrangian relaxation, our loss function guarantees optimal solutions for the original constrained problem. Our empirical evaluation across diverse benchmarks demonstrated that the proposed self-supervised solver, `SS-CMPE` , consistently outperformed learning-to-optimize baselines in both solution quality and computational efficiency.

Advanced Inference Schemes for Neurosymbolic Models

We introduced novel inference methods for deep dependency networks (DDNs), which integrate neural networks for feature extraction with probabilistic models for capturing label dependencies. To address the limitations of existing sampling-based inference in DDNs, we proposed two techniques: (a) a random-walk based local search algorithm, and (b) a novel integer linear programming formulation that directly solves the MPE task. Our experiments on six multi-label classification datasets demonstrated that these techniques significantly improved the outputs of DDNs, outperforming both standalone neural classifiers and neural-graphical hybrids.

8.2 Directions for Future Research

Future research will focus on advancing neural solvers for real-world inference over discrete and continuous variables, designing self-supervised loss functions for broader classes of generative models, establishing theoretical approximation guarantees for neural inference, and developing interactive inference systems that support dynamic user interaction and privacy-preserving updates.

Advancing Neural Solvers for Inference

A crucial direction for future research involves enhancing neural solvers for inference in real-world domains such as healthcare, industrial automation, and other safety-critical applications. In healthcare, neural models can improve diagnostic accuracy by reasoning over diverse variables, including disease types (discrete), patient temperature fluctuations (continuous), and other relevant patient attributes. Similarly, in manufacturing, neural inference can enhance defect detection by incorporating machine condition data, such as defect categories (discrete), machine temperature variations (continuous), and other relevant factors.

To enhance the applicability of neural models, future work should focus on designing neural architectures capable of handling both discrete and continuous variables. One potential solution is to develop *multi-class, multi-output heads* for discrete variables and *linear output heads* for continuous variables. Another promising avenue is the design of *self-supervised loss functions* tailored to broader classes of generative models and inference tasks, enabling more robust and generalizable solutions.

Theoretical Approximation Guarantees

An important direction for future research is establishing theoretical approximation guarantees for neural inference approaches. Unlike traditional probabilistic inference algorithms, most neural approximators lack formal bounds on solution quality relative to exact inference. The absence of such guarantees limits the reliability of neural methods in high-stakes applications where approximate answers must still meet rigorous quality thresholds. Providing theoretical assurances would bridge this gap and make neural inference solutions more trustworthy.

Future work should focus on deriving formal bounds that relate the neural model's solution quality to the exact solution under different model classes and query types. Formalizing these guarantees would promote the adoption of neural inference models in mission-critical domains, where predictability and transparency are essential.

Interactive Inference Systems

Another promising direction for future work is the development of interactive inference systems, particularly for applications requiring human oversight and interoperability. Human-in-the-loop frameworks enable users to iteratively refine evidence and query sets through natural feedback mechanisms, such as corrections, suggestions, or preference adjustments. Neural models can dynamically update their inference results in response, offering improved flexibility and user alignment. This setting is highly relevant in domains like scientific discovery, policy-making, and clinical decision support, where users often refine problem specifications as insights evolve.

Future work should explore designing neural inference models that support real-time updates with minimal retraining or computational overhead. Reinforcement learning can be employed to optimize the interaction loop, training models to anticipate user corrections and proactively query for the most informative feedback. Creating such adaptive systems would enhance both the usability and the trustworthiness of neural inference frameworks.

BIBLIOGRAPHY

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

Achterberg, T., T. Berthold, T. Koch, and K. Wolter (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*, pp. 6–20. Springer.

Alet, F., M. Bauza, K. Kawaguchi, N. G. Kuru, T. Lozano-Pérez, and L. Kaelbling (2021). Tailoring: Encoding inductive biases by optimizing unsupervised objectives at prediction time. In *Advances in Neural Information Processing Systems*, Volume 34, pp. 29206–29217. Curran Associates, Inc.

Allen-Zhu, Z., Y. Li, and Z. Song (2019). A convergence theory for deep learning via over-parameterization. In K. Chaudhuri and R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Volume 97 of *Proceedings of Machine Learning Research*, pp. 242–252. PMLR.

Antonucci, A., G. Corani, D. D. Mauá, and S. Gabaglio (2013). An ensemble of bayesian networks for multilabel classification. In F. Rossi (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 1220–1225. IJCAI/AAAI.

Arnab, A., S. Jayasumana, S. Zheng, and P. H. S. Torr (2016). Higher order conditional random fields in deep neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling (Eds.), *Computer Vision – ECCV 2016*, Cham, pp. 524–540. Springer International Publishing.

Arnab, A., S. Zheng, S. Jayasumana, B. Romera-Paredes, M. Larsson, A. Kirillov, B. Savchynskyy, C. Rother, F. Kahl, and P. H. Torr (2018). Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation: Combining Probabilistic Graphical Models with Deep Learning for Structured Prediction. *IEEE Signal Processing Magazine 35*(1), 37–52.

Arora, S., S. S. Du, W. Hu, Z. Li, and R. Wang (2019). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In K. Chaudhuri and R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Volume 97 of *Proceedings of Machine Learning Research*, pp. 322–332. PMLR.

Arya, D., D. K. Gupta, S. Rudinac, and M. Worring (2020). HyperSAGE: Generalizing inductive representation learning on hypergraphs. *ArXiv preprint abs/2010.04558*.

Arya, S., T. Rahman, and V. Gogate (2024a, April). Learning to Solve the Constrained Most Probable Explanation Task in Probabilistic Graphical Models. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Volume 238 of *Proceedings of Machine Learning Research*, pp. 2791–2799. PMLR.

Arya, S., T. Rahman, and V. Gogate (2024b). Neural Network Approximators for Marginal MAP in Probabilistic Circuits. *Proceedings of the AAAI Conference on Artificial Intelligence 38*(10), 10918–10926.

Arya, S., T. Rahman, and V. G. Gogate (2024c). A neural network approach for efficiently answering most probable explanation queries in probabilistic models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Arya, S., T. Rahman, and V. G. Gogate (2025). SINE: Scalable MPE inference for probabilistic graphical models using advanced neural embeddings. In *The 28th International Conference on Artificial Intelligence and Statistics*.

Arya, S., Y. Xiang, and V. Gogate (2023). Deep dependency networks for multi-label classification. *ArXiv preprint abs/2302.00633*.

Asghari, M., A. M. Fathollahi-Fard, S. M. J. Mirzapour Al-e-hashem, and M. A. Dulebenets (2022). Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey. *Mathematics 10*(2), 283.

Bai, S., F. Zhang, and P. H. S. Torr (2019). Hypergraph convolution and hypergraph attention. *ArXiv preprint abs/1901.08150*.

Barbu, A., D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz (2019). Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 9448–9458.

Bekker, J., J. Davis, A. Choi, A. Darwiche, and G. Van den Broeck (2015). Tractable learning for complex probability queries. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Volume 28. Curran Associates, Inc.

Besag, J. (1975). Statistical Analysis of Non-Lattice Data. *The Statistician 24*, 179–195.

Bioucas-Dias, J. and M. Figueiredo (2016). Bayesian image segmentation using hidden fields: Supervised, unsupervised, and semi-supervised formulations. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 523–527.

Cai, D., M. Song, C. Sun, B. Zhang, S. Hong, and H. Li (2022). Hypergraph structure learning for hypergraph neural networks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, Vienna, Austria, pp. 1923–1929. International Joint Conferences on Artificial Intelligence Organization.

Chen, H., H. Zhang, P.-Y. Chen, J. Yi, and C.-J. Hsieh (2018). Attacking visual language grounding with adversarial examples: A case study on neural image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia, pp. 2587–2597. Association for Computational Linguistics.

Chen, J. and P. Schwaller (2024). Molecular hypergraph neural networks. *The Journal of Chemical Physics 160*(14), 144307.

Chen, L., A. G. Schwing, A. L. Yuille, and R. Urtasun (2015). Learning deep structured models. In F. R. Bach and D. M. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, Volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1785–1794. JMLR.org.

Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence 40*(4), 834–848.

Chen, T., M. Xu, X. Hui, H. Wu, and L. Lin (2019). Learning semantic-specific graph representation for multi-label image recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 522–531. IEEE.

Chen, X. and A. L. Yuille (2014). Articulated pose estimation by a graphical model with image dependent pairwise relations. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1736–1744.

Chizat, L. and F. R. Bach (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 3040–3050.

Cho, J. H. and B. Hariharan (2019, November). On the Efficacy of Knowledge Distillation . In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA, pp. 4793–4801. IEEE Computer Society.

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

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

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

Choi, Y., A. Vergari, and G. Van den Broeck (2020a). Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, University of California, Los Angeles.

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

Chua, T.-S., J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng (2009). NUS-WIDE: A real-world web image database from National University of Singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, Santorini, Fira Greece, pp. 1–9. ACM.

Cohen, G., S. Afshar, J. Tapson, and A. van Schaik (2017). EMNIST: An extension of MNIST to handwritten letters. *ArXiv preprint abs/1702.05373*.

Conaty, D., C. P. de Campos, and D. D. Mauá (2017). Approximation complexity of maximum A posteriori inference in sum-product networks. In G. Elidan, K. Kersting, and A. T. Ihler (Eds.), *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence 42*(2-3), 393–405.

Craley, J., E. Johnson, and A. Venkataraman (2019). Integrating Convolutional Neural Networks and Probabilistic Graphical Modeling for Epileptic Seizure Detection in Multichannel EEG. In A. C. S. Chung, J. C. Gee, P. A. Yushkevich, and S. Bao (Eds.), *Information Processing in Medical Imaging*, Volume 11492, pp. 291–303. Cham: Springer International Publishing. Series Title: Lecture Notes in Computer Science.

Cui, Z., H. Wang, T. Gao, K. Talamadupula, and Q. Ji (2022a). Variational message passing neural network for Maximum-A-Posteriori (MAP) inference. In J. Cussens and K. Zhang (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*, pp. 464–474. PMLR.

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

Darestani, M. Z., J. Liu, and R. Heckel (2022). Test-time training can close the natural distribution shift performance gap in deep learning based compressed sensing. In *Proceedings of the 39th International Conference on Machine Learning*, Volume 162 of *Proceedings of Machine Learning Research*, pp. 4754–4776. PMLR.

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

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

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

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

de Campos, C. P. (2011). New complexity results for MAP in bayesian networks. In T. Walsh (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2100–2106. IJCAI/AAAI.

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

Dechter, R. (2019). *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing.

Dechter, R., A. Ihler, V. Gogate, J. Lee, B. Pezeshki, A. Raichev, and N. Cohen (2022). UAI 2022 competition.

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

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

Deng, J., W. Dong, R. Socher, L. Li, K. Li, and F. Li (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society.

Di Mauro, N., A. Vergari, and F. Esposito (2016). Multi-label classification with cutset networks. In A. Antonucci, G. Corani, and C. P. Campos (Eds.), *Proceedings of the Eighth International Conference on Probabilistic Graphical Models*, Volume 52 of *Proceedings of Machine Learning Research*, Lugano, Switzerland, pp. 147–158. PMLR.

Dong, Y., W. Sawin, and Y. Bengio (2020). HNHN: Hypergraph networks with hyperedge neurons. *ArXiv preprint abs/2006.12278*.

Donti, P. L., D. Rolnick, and J. Z. Kolter (2020). DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*.

Du, S. S., X. Zhai, B. Póczos, and A. Singh (2019). Gradient descent provably optimizes over-parameterized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Elidan, G. and A. Globerson (2010). *The 2010 UAI Approximate Inference Challenge*.

Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision 88*(2), 303–338.

Fan, H., Y. Li, B. Xiong, W.-Y. Lo, and C. Feichtenhofer (2020). Pyslowfast. `https://github.com/facebookresearch/slowfast`.

Feichtenhofer, C., H. Fan, J. Malik, and K. He (2019). Slowfast networks for video recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 6201–6210. IEEE.

Feng, Y., H. You, Z. Zhang, R. Ji, and Y. Gao (2019). Hypergraph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 3558–3565. AAAI Press.

Fey, M. and J. E. Lenssen (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Figueiredo, L. F. d., A. Paes, and G. Zaverucha (2021). Transfer learning for boosted relational dependency networks through genetic algorithm. In *International Conference on Inductive Logic Programming*, pp. 125–139. Springer.

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

Furlanello, T., Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar (2018). Born-again neural networks. In J. G. Dy and A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Volume 80 of *Proceedings of Machine Learning Research*, pp. 1602–1611. PMLR.

Gámez, J. A., J. L. Mateo, T. D. Nielsen, and J. M. Puerta (2008). Robust classification using mixtures of dependency networks. In *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models*, pp. 129–136.

Gámez, J. A., J. L. Mateo, and J. M. Puerta (2006). Dependency networks based classifiers: learning models by using independence test. In *Third European Workshop on Probabilistica Graphical Models (PGM06)*, pp. 115–122. Citeseer.

Geißler, B., A. Martin, A. Morsi, and L. Schewe (2012). Using Piecewise Linear Functions for Solving MINLPs. In J. Lee and S. Leyffer (Eds.), *Mixed Integer Nonlinear Programming*, The IMA Volumes in Mathematics and Its Applications, New York, NY, pp. 287–314. Springer.

Germain, M., K. Gregor, I. Murray, and H. Larochelle (2015). MADE: masked autoencoder for distribution estimation. In F. R. Bach and D. M. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, Volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 881–889. JMLR.org.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017). Neural message passing for quantum chemistry. In D. Precup and Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR.

Globerson, A. and T. S. Jaakkola (2007). Fixing max-product: Convergent message passing algorithms for MAP lp-relaxations. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.), *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pp. 553–560. Curran Associates, Inc.

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`.

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

Goh, G., N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah (2021). Multimodal neurons in artificial neural networks. *Distill*.

Grattafiori, A., A. Dubey, A. Jauhri, A. Pandey, and et al (2024). The llama 3 herd of models. *ArXiv preprint abs/ 2407*.

Guo, Q. and Q. Dou (2021). Semantic Image Segmentation based on SegNetWithCRFs. *Procedia Computer Science 187*, 300–306.

Guo, X. and Y. Weng (2020). Deep Dependency Network for Multi-label Text Classification. In Y. Peng, Q. Liu, H. Lu, Z. Sun, C. Liu, X. Chen, H. Zha, and J. Yang (Eds.), *Pattern Recognition and Computer Vision*, Lecture Notes in Computer Science, Cham, pp. 298–309. Springer International Publishing.

Guo, Y. and S. Gu (2011). Multi-label classification using conditional dependency networks. In T. Walsh (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 1300–1305. IJCAI/AAAI.

Guo, Y. and W. Xue (2013). Probabilistic multi-label classification with sparse feature learning. In F. Rossi (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 1373–1379. IJCAI/AAAI.

Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.

Haaren, J. V. and J. Davis (2012a). Markov network structure learning: A randomized feature generation approach. In J. Hoffmann and B. Selman (Eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press.

Haaren, J. V. and J. Davis (2012b). Markov network structure learning: A randomized feature generation approach. In J. Hoffmann and B. Selman (Eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press.

Han, K., W. Wan, H. yan Yao, and L. Hou (2017). Image crowd counting using convolutional neural network and markov random field. *Journal of Advanced Computational Intelligence and Intelligent Informatics* (4), 632–638.

Hardt, M. and Y. Sun (2024). Test-time training on nearest neighbors for large language models. In *The Twelfth International Conference on Learning Representations*.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society.

Heckerman, D., D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research 1*(Oct), 49–75.

Heo, B., J. Kim, S. Yun, H. Park, N. Kwak, and J. Y. Choi (2019). A comprehensive overhaul of feature distillation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 1921–1930. IEEE.

Hinton, G. E., O. Vinyals, and J. Dean (2015). Distilling the knowledge in a neural network. *CoRR abs/1503.02531*.

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

Huang, J. and J. Yang (2021). UniGNN: A unified framework for graph and hypergraph neural networks. *ArXiv preprint abs/2105.00956*.

Ihler, A. T., N. Flerova, R. Dechter, and L. Otten (2012). Join-graph based cost-shifting schemes. In N. de Freitas and K. P. Murphy (Eds.), *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pp. 397–406. AUAI Press.

Jacot, A., C. Hongler, and F. Gabriel (2018). Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 8580–8589.

Johnson, M. J., D. Duvenaud, A. B. Wiltschko, R. P. Adams, and S. R. Datta (2016). Composing graphical models with neural networks for structured representations and fast inference. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2946–2954.

Karpathy, A. and F. Li (2015). Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 3128–3137. IEEE Computer Society.

Kim, J., S. Park, and N. Kwak (2018). Paraphrasing complex network: Network compression via factor transfer. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2765–2774.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kirillov, A., D. Schlesinger, S. Zheng, B. Savchynskyy, P. H. S. Torr, and C. Rother (2016). Joint Training of Generic CNN-CRF Models with Stochastic Optimization.

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

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

Knöbelreiter, P., C. Reinbacher, A. Shekhovtsov, and T. Pock (2017). End-to-end training of hybrid CNN-CRF models for stereo. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 1456–1465. IEEE Computer Society.

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

Komodakis, N., N. Paragios, and G. Tziritas (2007). MRF optimization via dual decomposition: Message-passing revisited. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pp. 1–8. IEEE Computer Society.

Kong, D., Y. Chen, H. Ma, X. Yan, and X. Xie (2019). Adaptive graphical model network for 2d handpose estimation. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, pp. 83. BMVA Press.

Kong, X., B. Cao, and P. S. Yu (2013). Multi-label classification by mining label and instance correlations from heterogeneous information networks. In I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, and R. Uthurusamy (Eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pp. 614–622. ACM.

Kotary, J., F. Fioretto, and P. Van Hentenryck (2021). Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems 34*, 24981–24992.

Krishnan, R. G., U. Shalit, and D. Sontag (2015). Deep kalman filters. *stat 1050*, 25.

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

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114.

Kuck, J., S. Chakraborty, H. Tang, R. Luo, J. Song, A. Sabharwal, and S. Ermon (2020). Belief propagation neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

Kumar, M. (2007). Converting some global optimization problems to mixed integer linear problems using piecewise linear approximations. Master's thesis, University of Missouri–Rolla.

Lake, B. M., T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences 40*, e253.

Larochelle, H. and I. Murray (2011). The neural autoregressive distribution estimator. In G. Gordon, D. Dunson, and M. Dudík (Eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Volume 15 of *Proceedings of Machine Learning Research*, Fort Lauderdale, FL, USA, pp. 29–37. PMLR.

Larsson, M., J. Alvén, and F. Kahl (2017). Max-Margin Learning of Deep Structured Models for Semantic Segmentation. In P. Sharma and F. M. Bianchi (Eds.), *Image Analysis*, Lecture Notes in Computer Science, Cham, pp. 28–40. Springer International Publishing.

Larsson, M., A. Arnab, F. Kahl, S. Zheng, and P. Torr (2018, January). A Projected Gradient Descent Method for CRF Inference allowing End-To-End Training of Arbitrary Pairwise Potentials. Technical Report arXiv:1701.06805, arXiv. arXiv:1701.06805 [cs] type: article.

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

Lee, J., R. Marinescu, and R. Dechter (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.

Lee, J., S. S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Sohl-Dickstein (2020). Finite versus infinite neural networks: an empirical study. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lee, J., L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington (2019). Wide neural networks of any depth evolve as linear models under gradient descent. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8570–8581.

Lee, S.-i., V. Ganapathi, and D. Koller (2006). Efficient structure learning of markov networks using l_1-regularization. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems*, Volume 19. MIT Press.

Li, K. and J. Malik (2017). Learning to optimize. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Li, S.-Y., S.-J. Huang, and S. Chen (2021). Crowdsourcing aggregation with deep Bayesian learning. *Science China Information Sciences 64*(3), 130104.

Li, Y., X. Xu, Y. Su, and K. Jia (2023). On the Robustness of Open-World Test-Time Training: Self-Training with Dynamic Prototype Expansion. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Paris, France, pp. 11802–11812. IEEE.

Li, Z., Y. Zhang, B. Sui, Z. Xing, and Q. Wang (2022). FPGA Implementation for the Sigmoid with Piecewise Linear Fitting Method Based on Curvature Analysis. *Electronics 11*(9), 1365.

Liang, G., X. Lan, J. Wang, J. Wang, and N. Zheng (2018). A Limb-Based Graphical Model for Human Pose Estimation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 48*(7), 1080–1092. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems.

Lin, G., C. Shen, A. van den Hengel, and I. D. Reid (2016). Efficient piecewise training of deep structured models for semantic segmentation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 3194–3203. IEEE Computer Society.

Lin, M.-H., J. G. Carlsson, D. Ge, J. Shi, and J.-F. Tsai (2013). A Review of Piecewise Linearization Methods. *Mathematical Problems in Engineering 2013*, e101376.

Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer.

Liu, B., X. Liu, H. Ren, J. Qian, and Y. Wang (2022). Text multi-label learning method based on label-aware attention and semantic dependency. *Multimedia Tools and Applications 81*(5), 7219–7237.

Liu, C., B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy (2017). Progressive neural architecture search. *ArXiv preprint abs/1712.00559*.

Liu, F., C. Shen, and G. Lin (2015). Deep convolutional neural fields for depth estimation from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 5162–5170. IEEE Computer Society.

Liu, H., Z. Wu, L. Li, S. Salehkalaibar, J. Chen, and K. Wang (2022). Towards Multi-domain Single Image Dehazing via Test-time Training. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, LA, USA, pp. 5821–5830. IEEE.

Liu, J. (2008). *Monte Carlo strategies in scientific computing*. New York, Berlin, Heidelberg: Springer Verlag.

Liu, S., L. Zhang, X. Yang, H. Su, and J. Zhu (2021). Query2Label: A Simple Transformer Way to Multi-Label Classification. *ArXiv preprint abs/2107.10834*.

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

Liu, Y., P. Kothari, B. van Delft, B. Bellot-Gurlet, T. Mordan, and A. Alahi (2021). TTT++: When Does Self-Supervised Test-Time Training Fail or Thrive? In *Advances in Neural Information Processing Systems*, Volume 34, pp. 21808–21820. Curran Associates, Inc.

Loconte, L. and G. Gala (2022). DeeProb-kit: a python library for deep probabilistic modelling.

Lowd, D. (2012). Closed-form learning of markov networks from dependency networks. In N. de Freitas and K. P. Murphy (Eds.), *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pp. 533–542. AUAI Press.

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

Lowd, D. and A. Rooshenas (2015). The libra toolkit for probabilistic models. *Journal of Machine Learning Research 16*, 2459–2463.

Lowd, D. and A. Shamaei (2011). Mean field inference in dependency networks: An empirical study. In W. Burgard and D. Roth (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

Marinescu, R. and R. Dechter (2009a). AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence 173*(16-17), 1457–1491.

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

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

Marinescu, R. and R. Dechter (2019). Counting the optimal solutions in graphical models. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12091–12101.

Mateescu, R., K. Kask, V. Gogate, and R. Dechter (2010). Join-graph propagation algorithms. *Journal of Artificial Intelligence Research 37*, 279–328.

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

Mei, J., Y. Jiang, and K. Tu (2018). Maximum A posteriori inference in sum-product networks. In S. A. McIlraith and K. Q. Weinberger (Eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 1923–1930. AAAI Press.

Minsky, M. L. (1991). Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine 12*(2), 34–34.

Mirzadeh, S. I., K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar (2025). GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*.

Mirzadeh, S. I., M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh (2020). Improved Knowledge Distillation via Teacher Assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 34, pp. 5191–5198.

Naim, I., Y. Song, Q. Liu, H. Kautz, J. Luo, and D. Gildea (2014, Jun.). Unsupervised alignment of natural language instructions with video segments. *Proceedings of the AAAI Conference on Artificial Intelligence 28*(1).

Naim, I., Y. C. Song, Q. Liu, L. Huang, H. Kautz, J. Luo, and D. Gildea (2015). Discriminative unsupervised alignment of natural language instructions with corresponding video segments. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Denver, Colorado, pp. 164–174. Association for Computational Linguistics.

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

Neville, J. and D. Jensen (2003). Collective classification with relational dependency networks. In *Workshop on Multi-Relational Data Mining (MRDM-2003)*, pp. 77.

Nguyen, H. D., X.-S. Vu, and D.-T. Le (2021). Modular Graph Transformer Networks for Multi-Label Image Classification. *Proceedings of the AAAI Conference on Artificial Intelligence 35*(10), 9092–9100.

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

Osowiechi, D., G. A. V. Hakim, M. Noori, M. Cheraghalikhani, I. B. Ayed, and C. Desrosiers (2022). Tttflow: Unsupervised test-time training with normalizing flow. *IEEE Workshop/Winter Conference on Applications of Computer Vision*.

Otten, L. (2012). DAOOPT: Sequential and distributed AND/OR branch and bound for MPE problems.

Otten, L. and R. Dechter (2012). A case study in complexity estimation: Towards parallel branch-and-bound over graphical models. In N. de Freitas and K. P. Murphy (Eds.), *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pp. 665–674. AUAI Press.

Papagiannopoulou, C., G. Tsoumakas, and I. Tsamardinos (2015). Discovering and exploiting deterministic label relationships in multi-label learning. In L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams (Eds.), *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pp. 915–924. ACM.

Park, J. D. and A. Darwiche (2004). Complexity results and approximation strategies for map explanations. *J. Artif. Int. Res. 21*(1), 101–133.

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

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). Pytorch: An imperative style, high-performance deep learning library. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pearl, J. and R. Dechter (1990). Identifying independence in bayesian networks. *Networks 20*(5), 507–534.

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

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

Ping, W., Q. Liu, and A. T. Ihler (2015). Decomposition bounds for marginal MAP. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 3267–3275.

Poon, H. and P. M. Domingos (2011). Sum-product networks: A new deep architecture. In F. G. Cozman and A. Pfeffer (Eds.), *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pp. 337–346. AUAI Press.

Qin, T., S. R. Etesami, and C. A. Uribe (2024). Faster convergence of local SGD for over-parameterized models. *Transactions on Machine Learning Research*.

Qu, X., H. Che, J. Huang, L. Xu, and X. Zheng (2021). Multi-layered Semantic Representation Network for Multi-label Image Classification. *ArXiv preprint abs/2106.11596*.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever (2021). Learning transferable visual models from natural language supervision. *International Conference on Machine Learning*.

Rahman, T. and V. Gogate (2016a). Learning ensembles of cutset networks. In D. Schuurmans and M. P. Wellman (Eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 3301–3307. AAAI Press.

Rahman, T. and V. Gogate (2016b). Merging strategies for sum-product networks: From trees to graphs. In A. T. Ihler and D. Janzing (Eds.), *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*. AUAI Press.

Rahman, T., S. Jin, and V. Gogate (2019). Look ma, no latent variables: Accurate cutset networks via compilation. In K. Chaudhuri and R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Volume 97 of *Proceedings of Machine Learning Research*, pp. 5311–5320. PMLR.

Rahman, T., P. Kothalkar, and V. Gogate (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*, pp. 630–645. Springer.

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

Regneri, M., M. Rohrbach, D. Wetzel, S. Thater, B. Schiele, and M. Pinkal (2013). Grounding action descriptions in videos. *Transactions of the Association for Computational Linguistics 1*, 25–36.

Romero, A., N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio (2015). Fitnets: Hints for thin deep nets. In Y. Bengio and Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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

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

Rovatti, R., C. D'Ambrosio, A. Lodi, and S. Martello (2014). Optimistic MILP modeling of non-linear optimization problems. *European Journal of Operational Research 239*(1), 32–45.

Salakhutdinov, R. and I. Murray (2008). On the quantitative analysis of deep belief networks. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, Volume 307 of *ACM International Conference Proceeding Series*, pp. 872–879. ACM.

Satorras, V. G. and M. Welling (2021). Neural enhanced belief propagation on factor graphs. In A. Banerjee and K. Fukumizu (Eds.), *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, Volume 130 of *Proceedings of Machine Learning Research*, pp. 685–693. PMLR.

Schwing, A. G. and R. Urtasun (2015). Fully Connected Deep Structured Networks. Technical report, arXiv.

Selman, B., H. A. Kautz, and B. Cohen (1993). Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability*.

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

Sigurdsson, G. A., G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta (2016). Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, pp. 510–526. Springer.

Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Song, J., L. Wang, L. V. Gool, and O. Hilliges (2017). Thin-slicing network: A deep structured model for pose estimation in videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5563–5572. IEEE Computer Society.

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

Sun, Y., X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt (2020). Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In S. P. Singh and S. Markovitch (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 4278–4284. AAAI Press.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2818–2826. IEEE Computer Society.

Tan, M., Q. Shi, A. van den Hengel, C. Shen, J. Gao, F. Hu, and Z. Zhang (2015). Learning graph structure for multi-label image classification via clique generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 4100–4109. IEEE Computer Society.

Tarantola, C. and E. Blanc (2002). Dependency networks and bayesian networks for web mining. *WIT Transactions on Information and Communication Technologies 28*.

Tompson, J., A. Jain, Y. LeCun, and C. Bregler (2014). Joint training of a convolutional network and a graphical model for human pose estimation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1799–1807.

Ucla-Starai (2023). Density-Estimation-Datasets.

Velickovic, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio (2018). Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Vergari, A., Y. Choi, A. Liu, S. Teso, and G. Van den Broeck (2021). A compositional atlas of tractable circuit operations for probabilistic inference. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*, Volume 34, pp. 13189–13201. Curran Associates, Inc.

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 3156–3164. IEEE Computer Society.

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

Wainwright, M. J., J. Lafferty, and P. Ravikumar (2006). High-dimensional graphical model selection using ℓ_1-regularized logistic regression. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems*, Volume 19. MIT Press.

Wang, D., E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell (2021). Tent: Fully test-time adaptation by entropy minimization. *International Conference on Learning Representations*.

Wang, H., M. Huang, and X. Zhu (2008). A generative probabilistic model for multi-label classification. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 628–637. IEEE.

Wang, J., Z. Zhang, C. Xie, Y. Zhou, V. Premachandran, J. Zhu, L. Xie, and A. Yuille (2018). Visual concepts and compositional voting. *Annals of Mathematical Sciences and Applications 3*(1), 151–188.

Wang, L., Y. Liu, H. Di, C. Qin, G. Sun, and Y. Fu (2021). Semi-supervised dual relation learning for multi-label classification. *IEEE Transactions on Image Processing 30*, 9125–9135.

Wang, R., R. Ridley, X. Su, W. Qu, and X. Dai (2021). A novel reasoning mechanism for multi-label text classification. *Information Processing & Management 58*(2), 102441.

Wang, S., J. Wang, Z. Wang, and Q. Ji (2014). Enhancing multi-label classification by modeling dependencies among labels. *Pattern Recognition 47*(10), 3405–3413.

Weng, W., B. Wei, W. Ke, Y. Fan, J. Wang, and Y. Li (2023). Learning label-specific features with global and local label correlation for multi-label classification. *Applied Intelligence 53*(3), 3017–3033.

Wu, A., E. K. Buchanan, M. Whiteway, M. Schartner, G. Meijer, J.-P. Noel, E. Rodriguez, C. Everett, A. Norovich, E. Schaffer, N. Mishra, C. D. Salzman, D. Angelaki, A. Bendesky, T. I. B. L. The International Brain Laboratory, J. P. Cunningham, and L. Paninski (2020). Deep graph pose: a semi-supervised deep graphical model for improved animal pose tracking. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Eds.), *Advances in Neural Information Processing Systems*, Volume 33, pp. 6040–6052. Curran Associates, Inc.

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

Yadati, N., M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. P. Talukdar (2019). Hypergcn: A new method for training graph convolutional networks on hypergraphs. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 1509–1520.

Yang, C., L. Xie, C. Su, and A. L. Yuille (2019). Snapshot distillation: Teacher-student optimization in one generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 2859–2868. Computer Vision Foundation / IEEE.

Yang, W., W. Ouyang, H. Li, and X. Wang (2016). End-to-end learning of deformable mixture of parts and deep convolutional neural networks for human pose estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 3073–3082. IEEE Computer Society.

Yao, K., B. Peng, G. Zweig, D. Yu, X. Li, and F. Gao (2014). Recurrent conditional random field for language understanding. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, pp. 4077–4081. IEEE.

Yedidia, J. S., W. T. Freeman, and Y. Weiss (2000). Generalized belief propagation. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pp. 689–695. MIT Press.

Yim, J., D. Joo, J. Bae, and J. Kim (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 7130–7138. IEEE Computer Society.

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

Yu, D., B. Yang, Q. Wei, A. Li, and S. Pan (2022). A probabilistic graphical model based on neural-symbolic reasoning for visual relationship detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10609–10618.

Zagoruyko, S. and N. Komodakis (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Zamzam, A. S. and K. Baker (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)*, pp. 1–6.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2017). Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Zhang, M., N. Wang, Y. Li, and X. Gao (2020). Neural Probabilistic Graphical Model for Face Sketch Synthesis. *IEEE Transactions on Neural Networks and Learning Systems 31*(7), 2623–2637.

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

Zheng, S., S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr (2015). Conditional random fields as recurrent neural networks. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1529–1537. IEEE Computer Society.

Zhou, W., Z. Xia, P. Dou, T. Su, and H. Hu (2023). Double Attention Based on Graph Attention Network for Image Multi-Label Classification. *ACM Transactions on Multimedia Computing, Communications, and Applications 19*(1), 1–23.

Zhu, M., J. Li, N. Wang, and X. Gao (2021). Learning Deep Patch representation for Probabilistic Graphical Model-Based Face Sketch Synthesis. *International Journal of Computer Vision 129*(6), 1820–1836.

Zhu, W., Y. Huang, D. Xu, Z. Qian, W. Fan, and X. Xie (2021). Test-time training for deformable multi-scale image registration. *IEEE International Conference on Robotics and Automation*.

Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2018). Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 8697–8710. IEEE Computer Society.

BIOGRAPHICAL SKETCH

Shivvrat Arya earned his bachelor's degree in 2019 in computer science and engineering from the Indian Institute of Information Technology Vadodara, India. He later joined The University of Texas at Dallas, where he is currently pursuing a master's degree in computer science and a PhD under the supervision of Dr. Vibhav Gogate and Dr. Yu Xiang. Shivvrat's research lies at the intersection of machine learning, deep learning, and computer vision, with a focus on tractable neurosymbolic AI for explainable and efficient inference. His work has been supported by DARPA, NSF, and AFOSR funding and has resulted in Best Paper Awards as well as spotlight and oral presentations at AI/ML conferences such as NeurIPS and AAAI.

CURRICULUM VITAE

Shivvrat Arya

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Email: `shivvrat.arya@utdallas.edu`

Educational History:

BTech, Computer Science and Engineering, IIIT Vadodara, India
MS, Computer Science, The University of Texas at Dallas
PhD, Computer Science, The University of Texas at Dallas

Neural Solvers for Fast, Accurate Probabilistic Inference
PhD Dissertation
Department of Computer Science, The University of Texas at Dallas
Advisors: Dr. Vibhav Gogate and Dr. Yu Xiang

Publications:

Conference Papers

1. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "SINE: Scalable MPE Inference for Probabilistic Graphical Models using Advanced Neural Embeddings," *28th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2025.

2. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "A Neural Network Approach for Efficiently Answering Most Probable Explanation Queries in Probabilistic Models," *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)* Spotlight Presentation (Top 3% papers), 2024.

3. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "Learning to Solve the Constrained Most Probable Explanation Task in Probabilistic Graphical Models," *27th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024.

4. **Shivvrat Arya**, Yu Xiang, Vibhav Gogate, "Deep Dependency Networks and Advanced Inference Schemes for Multi-Label Classification," *27th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024.

5. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "Neural Network Approximators for Marginal MAP in Probabilistic Circuits," *38th Annual AAAI Conference on Artificial Intelligence (AAAI)* Oral Presentation (Top 6% papers), 2024.

6. Rohith Peddi, **Shivvrat Arya**, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Eric Ragan, Nicholas Ruozzi, Yu Xiang, Vibhav Gogate, "CaptainCook4D: A Dataset for Understanding Errors in Procedural Activities," *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track (D&B)*, 2024.

Journal Papers

1. Chiradeep Roy*, Mahsan Nourani*, **Shivvrat Arya***, Mahesh Shanbhag, Tahrima Rahman, Eric D. Ragan, Nicholas Ruozzi, Vibhav Gogate, "Explainable Activity Recognition in Videos using Deep Learning and Tractable Probabilistic Models," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2023. (*These authors contributed equally.)

Workshop Papers

1. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "A Neural Network Approach for Efficiently Answering Most Probable Explanation Queries in Probabilistic Models," *UAI Tractable Probabilistic Modeling (TPM) Best Paper Award*, 2024.

2. **Shivvrat Arya**, Tahrima Rahman, Vibhav Gogate, "Neural Network Approximators for Marginal MAP in Probabilistic Circuits," *UAI Tractable Probabilistic Modeling (TPM)*, 2024.

3. Benjamin Rheault, **Shivvrat Arya**, Akshay Vyas, Jikai Wang, Rohith Peddi, Brett Benda, Vibhav Gogate, Nicholas Ruozzi, Yu Xiang, Eric Ragan, "Predictive Task Guidance with Artificial Intelligence in Augmented Reality," *IEEE Conference on Virtual Reality and 3D User Interfaces (IEEE VR)*, 2024.

4. Rohith Peddi, **Shivvrat Arya**, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Nicholas Ruozzi, Eric Ragan, Yu Xiang, Vibhav Gogate, "Put on your detective hat: What's wrong in this video?," *DMLR Data-centric Machine Learning Research (DMLR Workshop)*, 2023.