UNIVERSITY OF CALIFORNIA
SANTA CRUZ

A UNIFYING MATHEMATICAL FRAMEWORK FOR NEURAL-SYMBOLIC SYSTEMS

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE AND ENGINEERING

by

Charles Dickens

August 2024

The Dissertation of Charles Dickens is approved:

Professor Lise Getoor, Chair

Professor Stephen Wright

Professor Yang Liu

Professor Qi Gong

Peter Biehl
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

A Unifying Mathematical Framework for Neural-Symbolic Systems
by

Charles Dickens

The field of Neural-Symbolic (NeSy) systems is growing rapidly. Proposed approaches show great promise in achieving symbiotic unions of neural and symbolic methods. However, NeSy has not yet reached its full potential; approaches are often ad-hoc, problem-specific, and not easily generalizable. In this dissertation, I identify four milestones that are necessary to realize general and practical NeSy AI: (1) *a mathematical framework*, (2) *modeling paradigms*, (3) *learning techniques*, and (4) *a practical implementation*. My research contributes to reaching all four milestones. First, I introduce Neural-Symbolic Energy-Based Models (NeSy-EBMs), a unifying mathematical framework for discriminative and generative modeling with probabilistic and non-probabilistic NeSy approaches. Energy-based modeling provides a unified perspective of many NeSy systems and is a connection to the broader machine learning literature. Next, I utilize NeSy-EBMs to introduce a taxonomy of modeling paradigms focusing on a system's neural-symbolic interface and reasoning capabilities. My primary modeling paradigms organize and illuminate the capabilities of existing NeSy systems. Moreover, I identify architectures that support compelling NeSy use cases. Then, I introduce a suite of four NeSy learning techniques: one for learning the neural and symbolic weights separately and three for end-to-end learning. Moreover, I prove theoretical continuity properties and sufficient conditions for the differentiability of a large class of NeSy-EBM losses. Further, I formalize the challenges of NeSy learning using the NeSy-EBM mathematical framework and discuss the applicability of my learning techniques to my modeling paradigms. Finally, I present Neural Probabilistic Soft Logic (NeuPSL), an open-source NeSy-EBM library designed to support every modeling paradigm and learning technique I introduce and facilitate real-world applications of the NeSy approach. Additionally, I introduce a novel formulation of the (Neu)PSL inference problem as a linearly constrained quadratic program (LCQP) to prove differentiability properties necessary for end-to-end learning. Further, I propose a new inference algorithm that leverages the LCQP formulation and naturally exploits warm-

starts, leading to over 100x learning runtime improvements. Through extensive empirical analysis across twelve datasets, I validate my proposed modeling, inference, and learning methodologies while simultaneously demonstrating the advantages of NeSy-EBMs in various real-world tasks, including image classification, graph node labeling, autonomous vehicle situation awareness, and natural language question answering.

Acknowledgments

First and foremost, I want to thank my advisor Dr. Lise Getoor. I could not imagine an advisor better suited to guide me during my PhD. Lise has taught me the research process, refined my writing and presentation skills, and pushed me to perform impactful and rigorous research. However, among the many things I have learned from her, I think the most important is the value of social relationships and collaborations. I am incredibly thankful for the many opportunities she has opened for me to work with amazing industry and academic researchers. Finally, it is so fun to have an advisor I can connect with through research and a shared passion for surfing.

I am intensely grateful for my wife, Lilly Dickens. I honestly could not have completed this dissertation without her support. Lilly has makes me laugh when I am the most stressed. I can count on her to bring a practical perspective to my problems and to help me prioritize.

Next, I want to express my appreciation for my family. To my parents, Heidi and Richard Dickens, from my first little league baseball team to completing my PhD, you have encouraged me to finish what I have started. You have taught me to work hard but to also enjoy life. I am thankful for the support from my siblings, Holly, Rich, and Heather. I want to extend a special thank you to Heather; thank you for being the first sibling to finish graduate school and showing me it's possible.

Thank you to the LINQS lab. I am extraordinarily lucky to have a group of kind, genuinely supportive, and intelligent labmates. I want to extend special thanks to Sriram Srinivasan, Eriq Augustine, and Connor Pryor. I appreciate Sriram Srinivasan for giving me an opportunity to work with him on a journal publication early in my PhD and for having a relentlessly positive spirit. I appreciate Eriq Augustine for sharing your expertise in software development and research and for providing detailed and valuable feedback. I appreciate Connor Pryor for our multiple successful partnerships. Connor listens carefully to his collaborators and gives insightful feedback, he offers unique and quality ideas, and he is a good friend.

I had the pleasure of collaborating with amazing industry and academic researchers

during my PhD. I thank my collaborators for their insights and hard work. I want to extend a special thank you to Dr. Stephen Wright and Changyu Gao for a productive partnership. Dr. Stephen Wright and Changyu Gao are optimization experts, and I appreciate their thoroughness and creative approach to research.

Finally, I am grateful to my dissertation reading committee: Dr. Lise Getoor, Dr. Stephen Wright, Dr. Yang Liu, and Dr. Qi Gong. Thank you for your valuable time and feedback, which has greatly improved my dissertation.

Portions of this dissertation were supported by National Science Foundation grants CCF-2023495; and an unrestricted gift from Google. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

Chapter 1

Introduction

The field of artificial intelligence (AI) and machine learning is experiencing dramatic and never before seen growth in both its capabilities and application. Recent remarkable developments, including the groundbreaking chatbots powered by large language models (LLMs), have been driven by advancements in deep neural network architectures and training techniques [Goodfellow et al., 2014, Vaswani et al., 2017, Kipf and Welling, 2017, Ho et al., 2020, Kaplan et al., 2020, Radford et al., 2021, OpenAI, 2024]. However, models built purely on unconstrained deep neural networks still struggle with tasks requiring consistency with knowledge and complex reasoning. For instance, LLMs are notorious for hallucinating, i.e., performing unsound reasoning or producing irrelevant or factually incorrect completions Maynez et al. [2020], Ji et al. [2023]. This significantly hinders the utility and trustworthiness of AI models built purely on deep neural networks. Symbolic frameworks, on the other hand, have complementary properties. They execute logical and mathematical programs to perform sound reasoning and consider domain knowledge, but struggle with low-level perception and generation.

The promise of mutually beneficial neural and symbolic integrations has driven significant advancements in machine learning research. Much of the recent progress has been achieved in the neural-symbolic (NeSy) AI literature [d'Avila Garcez et al., 2002, 2009, 2019]. NeSy is a large and quickly growing community that has been holding regular workshops since 2005 [NeSy2005] and began holding conferences

in 2024 [NeSy2024]. At a high level, NeSy research aims to build algorithms and architectures for combining neural and symbolic components [Xu et al., 2018, Yang et al., 2020, Cohen et al., 2020, Manhaeve et al., 2021a, Wang et al., 2019, Badreddine et al., 2022, Ahmed et al., 2022a, Pryor et al., 2023a]. Researchers have demonstrated multiple compelling applications of the NeSy approach.

1.1 Motivating NeSy Applications

Here, I highlight four use cases that motivate NeSy: (1) *constraint satisfaction and joint reasoning*, (2) *fine-tuning and adaptation*, (3) *few-shot and zero-shot reasoning*, and (4) *semi-supervised learning*. This list is not exhaustive, however, the efficacy of the NeSy approach in these use cases is well established, and I will utilize these applications in the empirical analysis. I define each of the use cases and the high-level motivation for employing NeSy techniques in such settings below. Then, in Section 2.2 I revisit the four use cases and discuss concrete related work.

Constraint Satisfaction and Joint Reasoning: In real-world settings, a deployed model's predictions must meet well-defined requirements. Additionally, leveraging known patterns or dependencies in the output can significantly improve a model's accuracy and trustworthiness. Constraint satisfaction is finding a prediction that satisfies all requirements. NeSy systems perform constraint satisfaction by reasoning across their output to provide a structured prediction, typically using some form of joint reasoning. In other words, NeSy systems integrate constraints and knowledge into the prediction process.

Fine-tuning and Adaptation: We are in the era of foundation models in AI [Bommasani et al., 2022]. It is now standard practice to adjust a model that is pre-trained on large amounts of general data (typically using self-supervision) for downstream tasks. Fine-tuning and adaptation are two methods for leveraging a pre-trained model to perform a specific task or to transfer to a new domain [Devlin et al., 2019, J. Hu et al., 2022]. Fine-tuning and adaptation adjust the pre-trained model to minimizing a learning objective over a dataset, both of which are specialized for the downstream tasks. These are necessary steps in the modern AI development

process. NeSy frameworks are used in the fine-tuning and adaptation steps to design principled learning objectives that integrate knowledge and constraints relevant to the downstream task and the application domain.

Few-Shot and Zero-Shot Reasoning: Training data for a downstream task may be limited or even non-existent. In *few-shot* settings, only a few examples are available, while in *zero-shot* settings, no explicit training data is provided for the task. In these settings, few-shot and zero-shot reasoning techniques are used to enable a model to generalize beyond the limited available training data. Leveraging pre-trained models and domain knowledge are key ideas for succeeding in few-shot and zero-shot contexts. NeSy techniques have been successfully applied for various few-shot and zero-shot settings. Integrating symbolic knowledge and reasoning enables better generalization from a small number of examples. NeSy systems can utilize symbolic knowledge to make deductions about unseen classes or tasks.

Semi-Supervised Learning: Semi-supervised approaches facilitate learning from labeled as well as unlabelled data by combining the goals of supervised and unsupervised machine learning [E. van Engelen and H. Hoos, 2020]. Supervised methods fit a model to predict an output label given a corresponding input, while unsupervised methods infer the underlying structure in the data. The ability to leverage both labeled and unlabelled data leads to performance improvements, better generalization, and reduced labeling costs. NeSy is a functional approach to semi-supervised learning that leverages knowledge and domain constraints to train a model. This is achieved with loss functions that encode domain knowledge and structure and depend only on the input and output; that is, the loss does not require a label.

1.2 Milestones on the Path to NeSy AI:

Despite the growing interest and multiple compelling use cases, NeSy approaches are often ad-hoc, problem-specific, and not easily generalizable. I identify four milestones that need to be reached to achieve practical and generally applicable NeSy systems: (1) *a mathematical framework*, (2) *modeling paradigms*, (3) *learning techniques*, and (3) *a practical implementation*. Each milestone represents significant progress and

depends on previous achievements.

A Mathematical Framework: First, NeSy needs a unifying mathematical framework. The framework needs to be a common foundation for understanding and communicating ideas within NeSy and to the broader AI and machine learning community. Moreover, NeSy research and system design has to have a common and formal starting point. Without such a framework, research in NeSy is at risk of being misunderstood, confined, and impractical.

Modeling Paradigms: Next, using the language of a mathematical framework, primary NeSy modeling paradigms must be established. Practitioners need a taxonomy of modeling paradigms to understand where NeSy systems can be applied. Likewise, researchers need to be aware of the strengths and weaknesses of systems to build new architectures and algorithms.

Learning Techniques: Following the creation of the mathematical framework and standard modeling paradigms, foundational learning techniques can be created. It is necessary for the learning techniques to be expressed with the unifying framework, have transparent trade-offs, and connect to the modeling paradigms. This requirement enables general progress in NeSy learning theory and educated algorithm design.

A Practical Implementation: Finally, the first three milestones culminate in a set of modeling paradigms and learning techniques that need to be supported by a NeSy modeling framework. Further, specialized inference algorithms that are tailored to the NeSy modeling framework need to be explored.

1.3 Contributions

The contributions of my research are aligned with the four milestones described in the previous section. Here, I provide a brief description of my contributions and cite my published research on the topics. I provide thorough details of the contributions in later chapters. Moreover, I demonstrate the value of the four use cases listed earlier in this section and simultaneously analyze my proposed methodologies in an extensive empirical analysis across multiple variations of twelve datasets. I deliver compelling results for real-world applications, including graph node classification,

computer vision object detection, and natural language question answering. Notably, my methodologies are shown to lower runtimes, enhance prediction performance, enforce constraints, and improve label and data efficiency.

A Mathematical Framework: To fill the need for a unifying mathematical NeSy framework, I introduce *Neural-Symbolic Energy-Based Models* (NeSy-EBMs) [Pryor et al., 2023a, Dickens et al., 2024b]. NeSy-EBMs are a family of Energy-Based Models (EBMs) [LeCun et al., 2006] defined by energy functions that are compositions of parameterized neural and symbolic components. The neural component is a collection of deep models, and its output is provided to the symbolic component, which measures the compatibility of variables using domain knowledge and constraints. This general formalization captures discriminative and generative modeling with probabilistic and non-probabilistic NeSy approaches. Further, energy-based modeling is an established and recognized perspective that connects NeSy to the broader machine learning literature.

Modeling Paradigms: I use the NeSy-EBM framework to introduce a general formalization of reasoning as mathematical programming. This formalization motivates a new NeSy taxonomy that categorizes models based on their reasoning capabilities [Dickens et al., 2024c]. Specifically, I organize approaches into three modeling paradigms that vary with increasing expressivity and complexity: *deep symbolic variables (DSVar)*, *deep symbolic parameters (DSPar)*, and *deep symbolic potentials (DSPot)*. These categories are differentiated by their neural-symbolic connection, i.e., the way in which the neural component output is utilized in the symbolic component. These primary NeSy modeling paradigms organize and illuminate the capabilities of existing NeSy systems. Moreover, I identify architectures that support the mentioned NeSy use cases. In my empirical analysis, I demonstrate the strengths and weaknesses of each NeSy modeling paradigm. Notably, I show how to use the DSVar and DSPar paradigms to obtain up to a 37% point improvement over neural baselines in a semi-supervised setting and a 19% improvement over standard prompting with GPT-4 in natural language question-answering.

Learning Techniques: Further, I develop a suite of principled neural and symbolic

parameter learning techniques for NeSy [Srinivasan et al., 2021, Dickens et al., 2024a,b]. NeSy-EBM predictions are typically obtained by finding a state of variables with high compatibility (i.e., low energy). The high compatibility state is found by minimizing the energy function via an optimization algorithm, for instance, an interior point method for continuous variables [Nocedal and Wright, 2006] or a branch-and-bound strategy for discrete problems [H. Papadimitriou and Steiglitz, 1998]. The complex prediction process makes finding a gradient or descent direction of a standard machine learning loss with respect to the parameters difficult. To formalize these challenges and propose solutions, I introduce a categorization of learning losses based on the complexity of the relation to the NeSy-EBM energy function. I derive general expressions for gradients of the categorized learning losses with respect to the neural and symbolic parameters when the loss is differentiable. Additionally, I introduce four NeSy-EBM learning techniques: one for learning the neural and symbolic weights separately and three for end-to-end learning. I discuss the strengths and limitations of each technique and describe its applicability using the NeSy-EBM framework and modeling paradigms. Briefly, the separate or modular learning approach is the most widely applicable with respect to the NeSy-EBM architecture and utilizes black-box optimization techniques. However, modular learning is not effective for some NeSy use cases. On the other hand, the end-to-end learning approaches employ ideas from bilevel optimization and reinforcement learning literature. The end-to-end learning techniques require some assumptions about the NeSy-EBM architecture but are necessary for certain important NeSy applications. In my empirical evaluation, I investigate the runtime and performance trade-offs of the numerous learning techniques I propose. Importantly, the bilevel algorithm balances the strengths of the direct gradient descent and reinforcement learning methods, achieving up to a 16% point prediction performance improvement over alternatives while maintaining a low runtime.

A Practical Implementation: The insights gained from the NeSy-EBMs framework, modeling paradigms, and learning techniques motivate an encompassing system that supports the primary modeling paradigms and differentiability properties to

support learning. For this reason, I introduce *Neural Probabilistic Soft Logic* (NeuPSL), a highly expressive and efficient framework for constructing NeSy-EBMs [Pryor et al., 2023a, Dickens et al., 2024b]. NeuPSL uses the principled and comprehensive semantics of Probabilistic Soft Logic (PSL) [Bach et al., 2017] to create a NeSy-EBM symbolic component. The neural component can then be seamlessly integrated with the PSL symbolic component and built using any deep modeling library. Further, to ensure differentiability properties and provide principled forms of gradients for learning, I present a new formulation and regularization of PSL inference as a constrained quadratic program. Additionally, I introduce a corresponding dual block coordinate descent (dual BCD) inference algorithm that leverages the new formulation with multiple advantages. The dual BCD algorithm produces principled gradients for parameter learning and empirical results demonstrate that it can utilize warm starts effectively, leading to over 100× learning runtime improvements over the current best inference method.

1.4 Organization

This dissertation is organized as follows. In Chapter 2, I discuss related work on NeSy frameworks, NeSy applications, EBMs, black-box optimization, and bilevel optimization. Then, I formally define NeSy-EBMs and introduce three fundamental NeSy modeling paradigms in Chapter 3. Next, I present a suite of NeSy learning techniques in Chapter 4. I introduce NeuPSL, a scalable and expressive NeSy-EBM implementation, in Chapter 5. NeuPSL instantiates NeSy-EBMs in an empirical analysis of NeSy use cases, modeling paradigms, and learning algorithms in Chapter 6. Finally, I discuss limitations and conclusions in Chapter 7 and Chapter 8, respectively.

Chapter 2

Background and Related Work

There is a long, rich history of research on the integration of symbolic knowledge and reasoning with neural networks, which has rapidly evolved in the past decade. In this work, I establish a unifying framework for achieving such integration by connecting two foundational areas of machine learning research: Neural-Symbolic (NeSy) AI and energy-based modeling (EBMs). I use black-box optimization techniques for separate neural and symbolic component parameter learning. Additionally, I use bilevel optimization techniques to propose a new family of end-to-end gradient-based NeSy learning algorithms. The remainder of this section provides an overview of the related work in NeSy frameworks, NeSy applications, EBMs, black-box optimization, and bilevel optimization.

2.1 Neural-Symbolic Frameworks

NeSy empowers neural models with domain knowledge and reasoning through integrations with symbolic systems [d'Avila Garcez et al., 2002, 2009, 2019, De Raedt et al., 2020, Besold et al., 2022]. Various taxonomies have been proposed to categorize NeSy literature. Bader and Hitzler (2005), d'Avila Garcez et al. (2019), and most recently Besold et al. (2022) provide extensive surveys using characteristics such as knowledge representation, neural-symbolic connection, and applications to compare and describe methods. Similarly, the works of De Raedt et al. (2020) and Lamb et al. (2020) propose taxonomies to connect NeSy to statistical relational learning and graph

neural networks, respectively. Focused taxonomies are described by Giunchiglia et al. (2022) and van Krieken et al. (2022) for deep learning with constraints and symbolic knowledge representations and Dash et al. (2022) for integrating domain knowledge into deep neural networks. Marconato et al. (2023) characterizes the common reasoning mistakes made by NeSy models, and Marconato et al. (2024) presents an ensembling technique that calibrates the model's concept-level confidence to attempt to identify these mistakes. Recently, Wan et al. (2024) explored various NeSy AI approaches primarily focusing on workloads on hardware platforms, examining runtime characteristics and underlying compute operators. Finally, van Krieken et al. (2024) propose a language for NeSy called ULLER that aims to unify the representation of major NeSy systems, with the long-term goal of developing a shared Python library. Each of these surveys and taxonomies contributes to the comparison, understanding, and organization of the diverse collection of NeSy methodologies. I contribute to these efforts by introducing a common mathematical framework (Section 3.1) and a new taxonomy focused on the reasoning capabilities achievable by different NeSy modeling paradigms (Section 3.2).

I organize the exposition of related NeSy AI frameworks into three broad research areas: learning with constraints, differentiable reasoning layers, and reasoner agnostic systems. In the following subsections, I define each of the research areas and describe prominent examples of models belonging to the area.

2.1.1 Learning with Constraints

The essence of learning with constraints is using domain knowledge and common sense to construct a loss function [Giunchiglia et al., 2022, van Krieken et al., 2022]. This approach encodes the knowledge captured by the loss into the weights of the network. A key motivation is to ensure the compatibility of predictions with domain knowledge and common sense. Moreover, learning with constraints avoids potentially expensive post-prediction interventions that would be necessary with a model that is not aligned with domain knowledge. However, consistency with domain knowledge and sound reasoning are not guaranteed during inference for NeSy models in this

class.

Demeester et al. (2016), Rocktäschel and Riedel (2017), Diligenti et al. (2017), Bošnjak et al. (2017), and Xu et al. (2018) are prominent examples of the learning-with-constraints NeSy paradigm. Demeester et al. (2016) incorporates domain knowledge and common sense into natural language and knowledge base representations by encouraging partial orderings over embeddings via a regularization of the learning loss. Similarly, Rocktäschel and Riedel (2017) leverage knowledge represented as a differentiable loss derived from logical rules to train a matrix factorization model for relation extraction. Diligenti et al. (2017) use fuzzy logic to measure how much a model's output violates constraints, which is minimized during learning. Xu et al. (2018) introduces a loss function that represents domain knowledge and common sense by using probabilistic logic semantics. More recently, Giunchiglia et al. (2023) introduced an autonomous event detection dataset with logical requirements, and Stoian et al. (2023) shows that incorporating these logical requirements during the learning improves generalization.

2.1.2 Differentiable Reasoning Layers

Another successful area of NeSy is in differentiable reasoning layers. The primary difference between this family of NeSy approaches and learning with constraint is that an explicit representation of knowledge and reasoning is maintained in the model architecture during both learning and inference. A defining aspect of differentiable reasoning layers is the instantiation of knowledge and reasoning components as differentiable computation graphs. Differentiable reasoning layers support automatic differentiation during learning and symbolic reasoning during inference.

Pioneering works in differentiable reasoning include those of Wang et al. (2019), Cohen et al. (2020), Yang et al. (2020), Manhaeve et al. (2021a), Derkinderen et al. (2024), Badreddine et al. (2022), Ahmed et al. (2022a) and Ahmed et al. (2023a). Wang et al. (2019) integrates logical reasoning and deep models by introducing a differentiable smoothed approximation to a maximum satisfiability (MAXSAT) solver as a layer. Cohen et al. (2020) introduces a probabilistic first-order logic called TensorLog.

This framework compiles tractable probabilistic logic programs into differentiable layers. A TensorLog system is end-to-end differentiable and supports efficient parallelizable inference. Similarly, Yang et al. (2020) and Manhaeve et al. (2021a) compile tractable probabilistic logic programs into differentiable functions with their frameworks NeurASP and DeepProblog, respectively. NeurASP and DeepProblog use answer set programming [Brewka et al., 2011] and ProbLog [De Raedt et al., 2007] semantics, respectively. Winters et al. [2022] proposes DeepStochLog, a NeSy framework based on stochastic definite clause grammars that define a probability distribution over possible derivations. Recently, Maene and Raedt [2024] proposes DeepSoftLog, a superset of ProbLog, adding embedded terms that result in probabilistic rather than fuzzy semantics. The logic tensor network (LTN) framework proposed by Badreddine et al. (2022) uses neural network predictions to parameterize functions representing symbolic relations with real-valued or fuzzy logic semantics. The fuzzy logic functions are aggregated to define a satisfaction level. Predictions can be obtained by evaluating the truth value of all possible outputs and returning the highest-valued configuration. Badreddine et al. (2023) has expanded upon LTNs and presents a configuration of fuzzy operators for grounding formulas end-to-end in the logarithm space that is more effective than previous proposals. Recently, Ahmed et al. (2022a) introduced a method for compiling differentiable functions representing knowledge and logic using the semantics of probabilistic circuits (PCs) [Choi et al., 2020]. Their approach, called semantic probabilistic layers (SPLs), performs exact inference over tractable probabilistic models to enforce constraints over the predictions and uses the PC framework to ensure that the NeSy model is end-to-end trainable.

As pointed out by Cohen et al. (2020), answering queries in many (probabilistic) logics is equivalent to the weighted model counting problem, which is #P-complete or worse. Similarly, the MAXSAT problem studied by Wang et al. (2019) is NP-hard. Thus, since deep neural networks can be evaluated in time polynomial in their size, no polysize network can implement general logic queries unless #P=P, or MAXSAT solving, unless NP=P. For this reason, researchers have made progress

towards building more efficient differentiable reasoning systems by, for example, restricting the probabilistic logic to tractable families [Cohen et al., 2020, Ahmed et al., 2022a, Maene et al., 2024], or performing approximate inference [Wang et al., 2019, Manhaeve et al., 2021b, van Krieken et al., 2023].

2.1.3 Reasoner Agnostic Systems

More recently, researchers have sought to build NeSy frameworks with more general reasoning and knowledge representation capacities with expressive mathematical program blocks for reasoning. Mathematical programs are capable of representing cyclic dependencies across variables and ensuring the satisfaction of prediction constraints during learning and inference. Moreover, the system's high-level inference and training algorithms are agnostic to the solver used for the mathematical program.

Prominent reasoner-agnostic systems include the works of Amos and Kolter (2017), Agrawal et al. (2019a), Vlastelica et al. (2020), and Cornelio et al. (2023). Amos and Kolter (2017) integrate linearly constrained quadratic programming problems (LCQP) as layers in deep neural networks with their OptNet framework, and show that the solutions to the LCQP problems are differentiable with respect to the program parameters. The progress of OptNet was continued by the work of Agrawal et al. (2019a) with the application of domain-specific languages (DSLs) for instantiating the LCQP program layers. DSLs provide a syntax for specifying LCQPs representing knowledge and constraints, making optimization layers more accessible. Vlastelica et al. (2020) propose a method for computing gradients of solutions to mixed integer linear programs based on a continuous interpolation of the program's objective. In contrast to the works of Amos and Kolter (2017) and Agrawal et al. (2019a), the approach introduced by Vlastelica et al. (2020) supports integer constraints and achieves this by approximating the true gradient of the program output. Cornelio et al. (2023) takes a different approach from these three methods by employing reinforcement learning techniques to support more general mathematical programs. Specifically, the neural model's predictions are interpreted as a state in a Markov decision process. Actions from a policy are taken to identify components that violate constraints to obtain

a new state. The new state is provided to a solver, which corrects the violations, and a reward is computed. The solver is not assumed to be differentiable, and the REINFORCE algorithm [Williams, 1992] with a standard policy loss is used to train the system end-to-end without the need to backpropagate through the solver.

2.2 Neural-Symbolic Applications

In this section, I revisit the four motivating applications discussed in Section 1.1: (1) constraint satisfaction and joint reasoning, (2) fine-tuning and adaptation, (3) few-shot and zero-shot reasoning, and (4) semi-supervised learning.

2.2.1 Constraint Satisfaction and Joint Reasoning

A commonly used example of constraint satisfaction and joint reasoning is puzzle-solving. Many NeSy frameworks are introduced with an evaluation on visual Sudoku and its variants [Wang et al., 2019, Augustine et al., 2022]. In the visual Sudoku problem, puzzles are constructed with handwritten digits, and a model must classify the digits and infer numbers to fill in the empty cells using the rules of Sudoku. Empirical evaluations of NeSy systems that perform constraint satisfaction and joint reasoning on visual Sudoku problems can be found in Wang et al. [2019], Augustine et al. [2022], Pryor et al. [2023a], and Morra et al. [2023]. Similarly, Vlastelica et al. (2020) introduces the shortest path finding problem as a NeSy task. Images of terrain maps are partitioned into a grid, and the model must find a continuous lowest-cost path between two points. The works of Vlastelica et al. (2020) and Ahmed et al. [2022a] perform constraint satisfaction and joint reasoning with NeSy models for shortest path finding.

Constraint satisfaction and joint reasoning with NeSy models are also effective for real-world natural language tasks. For instance, Sachan et al. (2018) introduces the Nuts&Bolts NeSy system to build a pipeline for parsing physics problems. The NeSy system jointly infers a parsing from multiple components that incorporates domain knowledge and prevents the accumulation of errors that would occur from a naive composition. In another work, Zhang et al. (2023) propose GeLaTo (generating

language with tractable constraints) for imposing constraints on text generated from language models. GeLaTo generates text tokens by autoregressively sampling from a distribution constructed from a pre-trained language model and a tractable probabilistic model encoding the constraints. More recently, Pan et al. [2023] introduced the Logic-LM framework for integrating LLMs with symbolic solvers to improve complex problem-solving. Logic-LM formulates a symbolic model using an LLM that uses prompts of the syntax and semantics of the symbolic language. Finally, Abraham et al. (2024) introduced CLEVR-POC, which requires leveraging logical constraints to generate plausible answers to questions about a hidden object in a given partial scene. They then demonstrated remarkable performance improvements over neural methods by integrating an LLM with a visual perception network and a formal logical reasoner.

Computer vision systems also benefit from the constraint satisfaction and joint reasoning capabilities of NeSy models. For instance, semantic image interpretation (SII) is the task of extracting structured descriptions from images. Donadello et al. (2017) implemented a NeSy model for SII using the Logic Tensor Network (LTN) [Badreddine et al., 2022] framework for reasoning about "part-of" relations between objects with logical formulas. Similarly, Yi et al. (2019) propose a NeSy visual question-answering framework (NS-VQA). The authors employ deep representation learning for visual recognition to recover a structured representation of a scene and then language understanding to formulate a program from a question. A symbolic solver executes the formulated program to obtain an answer. Sikka et al. (2020) introduced Deep Adaptive Semantic Logic (DASL) for predicting relationships between pairs of objects in an image given the bounding boxes and object category labels, i.e., visual relationship detection. The DASL system allows a modeler to express knowledge using first-order logic and to combine domain-specific neural components into a single deep network. A DASL model is trained to maximize a measured truth value of the knowledge.

2.2.2 Fine-tuning and Adaptation

Giunchiglia et al. (2022) provides a recent survey of the use of logically specified background knowledge to train neural models. NeSy learning losses are applied in the work of Giunchiglia et al. (2023) to fine-tune a neural system for autonomous vehicle situation awareness [Singh et al., 2021]. In another computer vision task, Arrotta et al. (2024) develop a NeSy loss for training a neural model to perform context-aware human activity recognition. NeSy fine-tuning and adaptation have also been explored in the natural language processing literature. Recently, Ahmed et al. (2023b) proposed the pseudo-semantic loss for detoxifying large language models. The authors disallow a list of toxic words and show this intuitive approach steers a language model's generation away from harmful language and achieves state-of-the-art detoxification scores. Feng et al. (2024) has explored directly learning the reasoning process of logical solvers within the LLM to avoid parsing errors. Finally, Cunnington et al. (2024) introduced NeSyGPT, which fine-tunes a vision-language foundation model to extract symbolic features from raw data before learning an answer set program.

2.2.3 Few-Shot and Zero-Shot Reasoning

Providing recommendations for new items or users can be viewed as a few-shot or zero-shot problem. Kouki et al. (2015) introduce the HyPER (hybrid probabilistic extensible recommender) framework for incorporating and reasoning over a wide range of information sources. By combining multiple information sources via logical relations, the authors outperformed the state-of-the-art approaches of the time. More recently, Carraro et al. [2022] developed an LTN-based recommender system to overcome data sparsity. This model uses background knowledge to generalize predictions for new items and users quickly. Few-shot and zero-shot reasoning tasks are also prevalent in object navigation. The ability to navigate to novel objects and unfamiliar environments is vital for the practical use of embodied agents in the real world. In this context, Zhou et al. (2023) presents a method for "exploration with soft commonsense constraints" (ESC). ESC first employs a pre-trained vision and language

model for semantic scene understanding, then a language model to reason from the spatial relations, and finally PSL to leverage symbolic knowledge and reasoning to guide exploration. In natural language processing, Pryor et al. (2023b) infers the latent dialog structure of a goal-oriented conversation using domain knowledge to overcome the challenges of limited data and out-of-domain generalization. Further, the previously cited works of Pan et al. (2023) and Sikka et al. (2020) also find that the few-shot and zero-shot capabilities neural models can be improved with the NeSy approach. Specifically, in Pan et al. (2023), the authors achieve performance gains over 39% by integrating symbolic reasoners with LLMs and in Sikka et al. (2020) the addition of commonsense reasoning and knowledge improves performance by over 10% in data-scarce settings.

2.2.4 Semi-Supervised Learning

Early work on semi-supervision with knowledge was carried out by Chang et al. (2007), who unify and leverage task-specific constraints to encode structure in the input and output data and possible labels. They evaluate their semi-supervised learning method on the task of named entity recognition in citations as well as advertisements. More recently, Ahmed et al. (2022b) introduced the neuro-symbolic entropy regularization loss to encourage model confidence in predictions satisfying a set of constraints on the output. They demonstrate that the regularization improves model performances in the task of entity relation extraction in text. Additionally, Stoian et al. (2023) studied the effect of various t-norms used to soften the logical constraints for the symbolic component and demonstrated on a challenging road event detection dataset with logical requirements [Giunchiglia et al., 2023] that incorporating a symbolic loss drastically improves performance.

2.3 Energy-Based Models

My NeSy framework makes use of Energy-Based Models (EBMs) [LeCun et al., 2006]. EBMs measure the compatibility of a collection of observed (input) variables $\mathbf{x} \in \mathcal{X}$ and target (output) variables $\mathbf{y} \in \mathcal{Y}$ via a scalar-valued energy function:

$E : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$. Low energy states represent high compatibility. EBMs are appealing due to their generality in both modeling and application. For instance, EBMs can be used to perform density estimation by defining conditional, joint, and marginal Gibbs distributions with the energy function:

$$P(\mathbf{y}|\mathbf{x}) := \frac{e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}} e^{-\beta E(\hat{\mathbf{y}},\mathbf{x})}}, \tag{2.1}$$

$$P(\mathbf{y},\mathbf{x}) := \frac{e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}, \hat{\mathbf{x}} \in \mathcal{X}} e^{-\beta E(\hat{\mathbf{y}},\hat{\mathbf{x}})}}, \tag{2.2}$$

$$P(\mathbf{x}) := \frac{\int_{\hat{\mathbf{y}} \in \mathcal{Y}} e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}, \hat{\mathbf{x}} \in \mathcal{X}} e^{-\beta E(\hat{\mathbf{y}},\hat{\mathbf{x}})}}. \tag{2.3}$$

A fundamental motivation for the use of the Gibbs distribution is that any density function can be represented by the distribution shown above with a (potentially un-normalized) energy function E. For this reason, EBMs are a unified framework for probabilistic and non-probabilistic approaches and are applicable for generative and discriminative modeling.

EBMs are applied throughout machine learning to model data and provide predictions. The Boltzmann machine [Ackley et al., 1985, Salakhutdinov and Larochelle, 2010] and Helmholtz machine [Dayan et al., 1995] are some of the earliest EBMs to appear in the machine learning literature. Hinton (2002) is another seminal work that shows EBMs to be useful for building mixture-of-expert models. Specifically, a single complex distribution is produced by multiplying many simple distributions together and then renormalizing.

More recently, the EBM framework has been utilized for generative modeling [Zhao et al., 2017, Du and Mordatch, 2019, Du et al., 2023]. Zhao et al. (2017) introduce energy-based generative adversarial networks (EBGANs), which view the GAN discriminator as an energy function that attributes low energies (high compatibility) to points near the data manifold. The EBGAN approach is a principled framework for using GAN discriminators with a variety of architectures and learning loss functionals to achieve more stable training than traditional GANs. Du and Mordatch [2019] advocate for using EBMs *directly* for generative modeling, citing as

motivation their simplicity, stability, parameter efficiency, flexibility of generation, and compositionality. They show generative results that achieve performance close to modern GANs, achieving state-of-the-art results in out-of-distribution classification, adversarially robust classification, and other tasks. In more recent work, Du et al. (2023) propose an energy-based parameterization of diffusion models to support compositional generation.

The EBM framework was shown recently to improve discriminative modeling [Grathwohl et al., 2020, Liu et al., 2020]. Grathwohl et al. (2020) reinterpret discriminative classifiers as EBMs to propose the joint energy-based model (JEM). A JEM allows the parameters of the model to be fit on unlabeled data with a likelihood-based loss, leading to improved accuracy, robustness, calibration, and out-of-distribution detection. Similarly, Liu et al. (2020) developed an EBM for out-of-distribution detection to achieve state-of-the-art performance. Liu et al. (2020) creates a purely discriminative training objective (in contrast with the probabilistic approach of JEM) and shows that unnormalized energy scores can be used directly for out-of-distribution detection.

A primary challenge of the EBM framework is learning with a potentially intractable partition function induced by the Gibbs distributions in (2.1), (2.2), and (2.3). Some of the earliest EBMs worked around the partition function using the contrastive divergence algorithm [Hinton, 2002] to estimate derivatives of the negative log-likelihood loss of an EBM with Markov chain Monte Carlo (MCMC) sampling from the Gibbs distribution. Later work on EBMs has improved the traditional biased MCMC sampling-based approximation methods with a sampler based on stochastic gradient Langevin dynamics (SGLD) [Welling and Teh, 2011]. For instance, Du and Mordatch (2019) use SGLD for training generative EBMs and Grathwohl et al. (2020) for discriminative models with a negative log-likelihood loss.

Score matching is an alternative probabilistic approach to training an EBM that fits the slope (or score) of the model density to the score of the data distribution, avoiding the need to estimate the Gibbs distribution partition function [Hyvarinen, 2005, P. Kingma and LeCun, 2010, Song and Ermon, 2019]. Hyvarinen (2005) initially

proposed score matching for estimating non-normalized statistical models. Later, P. Kingma and LeCun (2010) used score matching to train an EBM for image denoising and super-resolution. Song and Ermon (2019) suggested training an EBM to approximate the score of the data distribution that is then used with Langevin dynamics for generation.

EBMs may also be trained via non-probabilistic losses that do not require estimating the Gibbs distribution partition function [LeCun et al., 1998, Collins, 2002, Scellier and Bengio, 2017]. For instance, the perceptron loss, which is the difference between the energy of the observed training data and the minimum value of the energy function (see Section 4.2 for a formal definition), has been used for recognizing handwritten digits [LeCun et al., 1998] and part-of-speech tagging [Collins, 2002]. More recently, Scellier and Bengio (2017) proposed *equilibrium propagation*, a two-phase learning algorithm for training EBMs with a twice differentiable energy function. The equilibrium propagation algorithm can be used to train EBMs with an arbitrary differentiable loss. A step of the learning algorithm proceeds by minimizing the energy given some input (the free phase) and then minimizing the energy augmented with a cost function (the nudged phase). The gradient of the learning objective is a function of the results of these two minimizations.

The EBM framework has proven effective for a wide range of tasks in both generative and discriminative modeling. The versatility of EBMs supports modeling complex dependencies, the composition and fusion of models, and leveraging both labeled and unlabeled data. Moreover, EBMs provide a common theoretical framework spanning probabilistic and non-probabilistic methods.

2.4 Black-Box Optimization

Black-box optimization, also referred to as derivative-free optimization, is a class of optimization techniques where the gradient of the objective is treated as an unknown and is only queried for evaluations at points in the problem domain [Nocedal and Wright, 2006, R. Conn et al., 2009, Snoek et al., 2012, Bergstra and Bengio, 2012, Shahriari et al., 2016]. The goal of black-box optimization is to find the best

possible value for a set of parameters $\mathbf{w}$ from a domain $\mathcal{W}$ that optimizes a function $\gamma(\mathbf{w}) : \mathcal{W} \to \mathbb{R}$:

$$\arg\min_{\mathbf{w}\in\mathcal{W}} \gamma(\mathbf{w}). \tag{2.4}$$

A high-level framework for black-box optimization is to define a search space, choose a point $\tilde{\mathbf{w}}$ from the space, evaluate $\gamma(\tilde{\mathbf{w}})$ to update a model of the function γ, and repeat. The selection and evaluation process can be simple and parallelized for algorithms like random grid search. On the other hand, more complex algorithms assume a sequential setup to ensure optimal selection of the next point to evaluate. For example, methods following the Bayesian optimization (BO) framework (Section 4.3.1) employ Gaussian process regression (GPR) to develop an approximation of γ and use strategies, called acquisition functions, to choose the next point. In the following subsections, background on GPR and acquisition functions is provided.

2.4.1 Gaussian Process Regression

A Gaussian process (GP) describes a distribution over a function space and is fully characterized by a mean function $\mu_0 : \mathcal{W} \to \mathbb{R}$ and a covariance matrix $\mathbf{K} = [K_{i,j}]$ Rasmussen and Williams [2005]. More formally, consider a finite ordered set of $s > 0$ parameter values, $\mathbf{W} = \{\mathbf{w}_1 \in \mathcal{W}, \cdots, \mathbf{w}_s \in \mathcal{W}\}$. Then, for any input $\mathbf{w}_i \in \mathbf{W}$, let $h_i = \gamma(\mathbf{w}_i)$ represent the function γ evaluated at $\mathbf{w}_i$. In GPR, a prior distribution is assumed over $\mathbf{h} = [h_1, \cdots, h_s]$ that is jointly Gaussian. Specifically, the prior distribution over $\mathbf{g}$ is $\mathbf{g} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$, where $m_i = \mu_0(\mathbf{w}_i)$. To allow for a more general setting with noisy function evaluations, let $\tilde{h}_i$ be the noisy output of the function $\gamma(\mathbf{w}_i)$, i.e., $\tilde{h}_i = \gamma(\mathbf{w}_i) + \epsilon$, where ϵ is Gaussian $\mathcal{N}(0, \sigma)$ noise. As the prior and noise are both Gaussian, if the vector $\tilde{\mathbf{h}} = [\tilde{h}_i, \cdots, \tilde{h}_s]$ of function evaluations at a set of points $\mathbf{W}$ is observed, then the likelihood of the noisy function evaluation at a new point $\mathbf{w}_{s+1}$ is also jointly Gaussian. Hence, the posterior distribution over $\tilde{h}_{s+1}$ given

$\tilde{\mathbf{h}}$ is $\tilde{h}_{s+1}|\tilde{\mathbf{h}} \sim \mathcal{N}(\mu_s(\mathbf{w}_{s+1}), \sigma_s(\mathbf{w}_{s+1}))$ where

$$\mu_s(\mathbf{w}_{s+1}) = \mu_0(\mathbf{w}_{s+1}) + K_{s+1}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(h - m)$$
$$\sigma_s(\mathbf{w}_{s+1}) = K_{s+1,s+1} - K_{s+1}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}K_{s+1}$$

and K_{s+1} is the vector of covariances between the new input $\mathbf{w}_{s+1}$ and every observed input $\mathbf{w}_i \in \mathbf{W}$. Assuming one has a method for computing covariances between any two points in the input space, then the posterior mean and variance of the noisy output of the function γ at any point can be computed using the above expressions. In GPR, a kernel function, $k(\cdot, \cdot)$, is used, and it plays the crucial role of defining the covariance between points in the input space, i.e., $K_{i,j} = \mathbb{E}\left[(\gamma(\mathbf{w}_i) - \mu_0(\mathbf{w}_i))(\gamma(\mathbf{w}_j) - \mu_0(\mathbf{w}_j))\right] = k(\mathbf{w}_i, \mathbf{w}_j)$. Consequentially, k encodes assumptions about the function the GP is developing a distribution over. The choice of kernel function is often the key to finding the best approximation of the true function.

2.4.2 Kernel Functions

In its most general form, a kernel function, $k(\cdot, \cdot)$, is any mapping of two inputs from a space $\mathcal{W}$ to $\mathbb{R}$. For a valid GP, the choice of kernel function must correspond to an inner product in some inner product space. Formally, for any two inputs $\mathbf{w}_i, \mathbf{w}_j \in \mathcal{W}$, $k(\cdot, \cdot)$ is equal to the inner product of the inputs after being mapped to an inner product space $\mathcal{H}$ via a transformation $\Psi : \mathcal{W} \to \mathcal{H}$, i.e.,

$$k : \mathcal{X} \times \mathcal{W} \to \mathbb{R}, \qquad (\mathbf{w}_i, \mathbf{w}_j) \mapsto k(\mathbf{w}_i, \mathbf{w}_j)$$

where

$$k(\mathbf{w}_i, \mathbf{w}_j) = \langle \Psi(\mathbf{w}_i), \Psi(\mathbf{w}_j) \rangle_{\mathcal{H}} \tag{2.5}$$

Requiring that a kernel, $k(\cdot, \cdot)$, corresponds to an inner product ensures that the matrix defined by the inputs $\mathbf{w}_1, \cdots, \mathbf{w}_s \in \mathcal{W}$ and the kernel, namely the Gram matrix $\mathbf{K} = [K_{i,j}]$ where $K_{i,j} = k(\mathbf{w}_i, \mathbf{w}_j)$, is positive semi-definite, and hence can

be used as a covariance matrix. Kernels with this property are referred to as positive semi-definite kernels or covariance functions.

There is a suite of positive semi-definite kernels that have been proposed in the literature, each equipped with different properties that may be more or less suited for a problem domain [Genton, 2001, Schölkopf and Smola, 2002, Rasmussen and Williams, 2005]. For instance, the stationary class of kernels, which includes the widely used squared exponential (employed in later Section 4.3.1), and the Matérn class of kernels [Matérn, 1960] assumes the covariance between inputs is translation invariant. More formally, stationary kernels assume that for some vector $\Delta_{\mathbf{w}} \in \mathcal{W}$, $k(\mathbf{w}_i, \mathbf{w}_j) = k(\mathbf{w}_i + \Delta_{\mathbf{w}}, \mathbf{w}_j + \Delta_{\mathbf{w}})$. Another common assumption is that of rotational invariance. This property is held by a class of kernels called dot product kernels, which only depend on the inputs $\mathbf{w}_i, \mathbf{w}_j$ through $\mathbf{w}_i \cdot \mathbf{w}_j$. One example of a dot product kernel is the inhomogeneous polynomial kernel $k(\mathbf{w}_i, \mathbf{w}_j) = (\sigma_0^2 + \mathbf{w}_i \cdot \mathbf{w}_j)$.

A domain will typically not satisfy the assumptions made by a single standard kernel. It is then necessary to design a specialized covariance function for the setting. At a high level, two approaches for designing covariance functions that meet a set of desired properties for a problem are: 1) proposing a novel positive semi-definite kernel function or 2) creating a new kernel from existing covariance functions through positive semi-definite preserving operations Genton [2001].

2.4.3 Acquisition Functions

An acquisition function α for BO determines the next weight configuration to evaluate the function γ, i.e., $\mathbf{w}_{next} = \arg\max_{\mathbf{w} \in \mathbf{W}} \alpha(\mathbf{w})$. The evaluation of $\gamma(\mathbf{w}_{next})$ is used to update the distribution described by the underlying GPR of a BO algorithm. There is a tradeoff between exploring a variety of weights to allow the GPR to develop a broad picture of the objective and exploiting previous observations to target promising regions of the search space. Four well-studied acquisition functions for balancing this tradeoff are:

Upper confidence bound

The upper confidence bound (UCB) is an optimistic policy with provable cumulative regret bounds [Srinivas et al., 2010]. The UCB acquisition function is:

$$\alpha(\mathbf{W}) = \mu(\mathbf{w}) + \psi \cdot \sigma(\mathbf{w})$$

where μ and σ are the mean and variance predicted by the GP and $\psi \geq 0$ is a hyperparameter set to achieve optimal regret bounds.

Thompson Sampling

Thompson sampling (TS) is an information-based policy that considers the posterior distribution over the weights $\mathbf{W}$ [Thompson, 1933]. The TS acquisition function is:

$$\begin{aligned} \alpha(\mathbf{W}) &= \tilde{p}(\mathbf{w}) \\ \tilde{p}(\mathbf{w}) &\sim \mathcal{N}(\mu(\mathbf{w}), \sigma(\mathbf{w})) \end{aligned}$$

where $\tilde{p}$ are samples obtained from the distribution computed at the point $\mathbf{w}$.

Probability of Improvement

Probability of improvement (PI) is an improvement-based policy that favors points that are likely to improve an incumbent target τ [Kushner, 1964]. The PI acquisition function is:

$$\alpha(\mathbf{W}) = \mathbb{P}(\gamma(\mathbf{w}) > \tau) = \mathcal{F}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)$$

where $\mathcal{F}$ is the standard normal cumulative distribution function and τ is set adaptively to the current best observed value for γ.

Expected improvement

Expected improvement (EI) is an improvement-based policy similar to PI [Mockus et al., 1978]. Instead of probability, it measures the expected amount of improvement.

The EI acquisition function is:

$$\alpha(\mathbf{W}) = \Big\{(\mu(\mathbf{w}) - \tau)\mathcal{F}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big) + (\sigma(\mathbf{w}))\mathcal{F}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)\Big\},$$

where $\mathcal{F}$ is the probability density function of a standard normal distribution.

2.5 Bilevel Optimization

Finally, in this dissertation, I use bilevel optimization as a natural formulation of learning for a general class of NeSy systems [Bracken and McGill, 1973, Colson et al., 2007, F. Bard, 2013, Dempe and Zemkoho, 2020]. The NeSy learning objective is a function of predictions obtained by solving a lower-level program that encapsulates symbolic reasoning. In the broader deep learning community, bilevel optimization also arises in hyperparameter optimization [Pedregosa, 2016], meta-learning [Franceschi et al., 2018, Rajeswaran et al., 2019], generative adversarial networks [Goodfellow et al., 2014], and reinforcement learning [Sutton and Barto, 2018]. Researchers typically take one of the following three approaches to bilevel optimization.

Implicit Differentiation. There is a long history of research on analyzing the stability of solutions to optimization problems using implicit differentiation [Fiacco and McCormick, 1968, Robinson, 1980, Bonnans and Shapiro, 2000]. These methods compute or approximate the Hessian matrix at the lower-level problem solution to derive an analytic expression for the gradient of the upper-level objective, sometimes called a hypergradient. Bilevel algorithms of this type make varying assumptions about the problem structure [Do et al., 2007, Pedregosa, 2016, Ghadimi and Wang, 2018, Rajeswaran et al., 2019, Giovannelli et al., 2022, Khanduri et al., 2023]. Building on these foundational techniques, the deep learning community has proposed architectures that contain layers that are functions of convex programs with analytic expressions for gradients derived from implicit differentiation [Amos and Kolter, 2017, Agrawal et al., 2019a,b, Wang et al., 2019].

Automatic Differentiation. This approach unrolls inference into a differentiable computational graph [Stoyanov et al., 2011, Domke, 2012, Belanger et al., 2017,

Ji et al., 2021], and then leverages automatic differentiation techniques [Griewank and Walther, 2008]. However, unrolling the inference computation creates a large, complex computational graph that can accumulate numerical errors dependent on the solver.

Bilevel Value-Function Approach. An increasingly popular approach is to reformulate the bilevel problem as a single-level constrained program using the optimal value of the lower-level objective (the value-function) to develop principled gradient-based algorithms that do not require the calculation of Hessian matrices for the lower-level problem [V. Outrata, 1990, J. Ye and L. Zhu, 1995, Liu et al., 2021, Sow et al., 2022, Liu et al., 2022, 2023, Kwon et al., 2023]. Existing bilevel value-function approaches are not directly applicable to NeSy systems as they typically assume the lower-level problem to be unconstrained and the objective to be smooth. Bilevel optimization with constraints in the lower level problem, is an open area of research. Until now, implicit differentiation methods are applied with strong assumptions about the structure of the lower-level problem [Giovannelli et al., 2022, Khanduri et al., 2023]. The bilevel learning framework I present in this dissertation is, to the best of my knowledge, the first value-function approach to work with lower-level problem constraints.

Chapter 3

A Unifying Mathematical Framework and A Taxonomy of Modeling Paradigms for NeSy

In this chapter, I introduce Neural-symbolic energy-based models (NeSy-EBMs): a unifying mathematical framework for NeSy. Intuitively, NeSy-EBMs formalize the neural-symbolic interface as a composition of functions. Then, I build a taxonomy of NeSy modeling paradigms that is expressed using the NeSy-EBM framework. The theory and notation introduced in this chapter are used throughout the rest of this dissertation.

3.1 Neural Symbolic Energy-Based Models

NeSy-EBMs are a family of EBMs [LeCun et al., 2006] that integrate deep architectures with explicit encodings of symbolic relations via an energy function. EBM energy functions measure the compatibility of variables where low energy states correspond to high compatibility. For NeSy-EBMs, high compatibility indicates that the variables are consistent with domain knowledge and common sense. In the following section, the formal NeSy-EBM definition is grounded with intuitive examples of NeSy modeling paradigms.

As diagrammed in Fig. 3.1, a NeSy-EBM energy function composes a neural component with a symbolic component, represented by the functions $\mathbf{g}_{nn}$ and $\mathbf{g}_{sy}$, respectively. The neural component is a deep model (or collection of deep models) parameterized by weights from a domain $\mathcal{W}_{nn}$, that takes a neural input from a domain $\mathcal{X}_{nn}$ and outputs a real-valued vector of dimension d_{nn}. The symbolic component encodes domain knowledge and is parameterized by weights from a domain $\mathcal{W}_{sy}$. It maps inputs from a domain $\mathcal{X}_{sy}$, target (or output) variables from $\mathcal{Y}$, and neural outputs from Range($\mathbf{g}_{nn}$) to a scalar value. In other words, the symbolic component measures the compatibility of targets, inputs, and neural outputs with domain knowledge.

Definition 1

*A **NeSy-EBM energy function** is a mapping parameterized by neural and symbolic weights from domains $\mathcal{W}_{nn}$ and $\mathcal{W}_{sy}$, respectively, and quantifies the compatibility of a target variable from a domain $\mathcal{Y}$ and neural and symbolic inputs from the domains $\mathcal{X}_{nn}$ and $\mathcal{X}_{sy}$, respectively, with a scalar value:*

$$E : \mathcal{Y} \times \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn} \to \mathbb{R}. \tag{3.1}$$

*A NeSy-EBM energy function is a composition of a **neural** and **symbolic component**. Neural weights parameterize the neural component, which outputs a real-valued vector of dimension d_{nn}:*

$$\mathbf{g}_{nn} : \mathcal{X}_{nn} \times \mathcal{W}_{nn} \to \mathbb{R}^{d_{nn}}. \tag{3.2}$$

The symbolic component maps the symbolic variables, symbolic parameters, and a real-valued vector of dimension d_{nn} to a scalar value:

$$g_{sy} : \mathcal{Y} \times \mathcal{X}_{sy} \times \mathcal{W}_{sy} \times \mathbb{R}^{d_{nn}} \to \mathbb{R}. \tag{3.3}$$

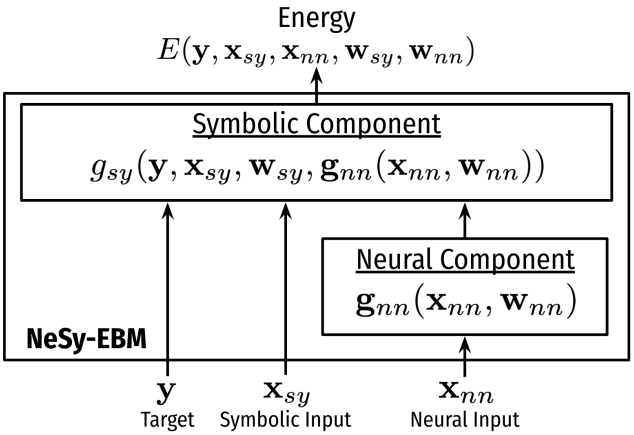


Figure 3.1: A neural-symbolic energy-based model.

The NeSy-EBM energy function is

$$E : (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \mapsto g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \quad \square$$

Given inputs and parameters $(\mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn}$, NeSy-EBM energy functions can be used to define several inference tasks, for instance:

- *Prediction, classification, and decision making*: Find targets minimizing the energy function.

$$\arg\min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}). \tag{3.4}$$

- *Ranking*: Sort a set of targets in order of increasing energy.

$$E(\mathbf{y}^{\mathbf{r}_1}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \leq \cdots \leq E(\mathbf{y}^{\mathbf{r}_p}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \tag{3.5}$$

- *Detection*: Determine if a target, $\mathbf{y}$, is below a threshold τ.

$$D(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \tau) := \begin{cases} 1 & E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \leq \tau \\ 0 & o.w. \end{cases} \tag{3.6}$$

- *Density estimation*: Estimate the conditional probability of a target, $\mathbf{y}$. The energy function is used to define a probability density, such as a Gibbs distri-

bution.

$$P(\mathbf{y}|\mathbf{x}_{sy}\mathbf{x}_{nn};\mathbf{w}_{sy},\mathbf{w}_{nn}) := \frac{e^{-\beta E(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})}}{\int_{\hat{\mathbf{y}}\in\mathcal{Y}} e^{-\beta E(\hat{\mathbf{y}},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})}}, \tag{3.7}$$

where β is the positive inverse temperature parameter.

- *Generation*: Sample a target variable state using a distribution defined by the energy function.

$$\mathbf{y} \sim P(\mathbf{y}|\mathbf{x}_{sy}\mathbf{x}_{nn};\mathbf{w}_{sy},\mathbf{w}_{nn}). \tag{3.8}$$

This dissertation focuses on the first and most common task in this list: prediction, classification, and decision-making (3.4). Prediction with NeSy-EBMs captures various reasoning frameworks, including probabilistic, logical, arithmetic, and their combinations. It can represent standard applications of prominent NeSy systems, including, DeepProbLog [Manhaeve et al., 2021a], LTNs [Badreddine et al., 2022], Semantic Probabilistic Layers [Ahmed et al., 2022a], and NeuPSL [Pryor et al., 2023a], to name a few.

3.2 A Taxonomy of Modeling Paradigms

Using the NeSy-EBM framework, I introduce a taxonomy of NeSy modeling paradigms determined by the neural-symbolic interface. The modeling paradigms are characterized by how the neural component is utilized in the symbolic component to define the prediction program in (3.4). To formalize the modeling paradigms, I introduce an additional layer of abstraction I refer to as *symbolic potentials*, denoted by ψ. Further, I collect symbolic potentials into *symbolic potential sets*, denoted by $\mathbf{\Psi}$. Symbolic potentials organize the arguments of the symbolic component by the role they play in formulating the prediction program in (3.4).

Definition 2

*A **symbolic potential** ψ is a function of variables from a domain V_ψ and parameters*

from a domain $Params_\psi$, outputting a scalar value:

$$\psi : V_\psi \times Params_\psi \to \mathbb{R}. \tag{3.9}$$

*A **symbolic potential set**, denoted by $\mathbf{\Psi}$, is a set of potentials indexed by $\mathbf{J}_\mathbf{\Psi}$.* □

Table 3.1: Summary of typical characteristics of the Deep Symbolic Variables (DSVar), Deep Symbolic Parameters (DSPar), and Deep Symbolic Potentials (DSPot) modeling paradigms.

		DSVar	DSPar	DSPot
Definition	$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) :=$	$\psi([\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})], \mathbf{w}_{sy})$ $+I_\mathcal{Y}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$	$\psi([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})])$	$\mathbf{\Psi}_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy})$
Properties	Fast Learning	✓		
	Fast Inference	✓		
	Expressive		✓	✓
	Open Domain			✓
Applications	Constraint Satisfaction		✓	✓
	Fine-tuning	✓	✓	
	Few/Zero-Shot		✓	✓
	Semi-Supervised	✓	✓	
Examples		Giunchiglia et al. [2022] Pryor et al. [2023b] Xu et al. [2018]	Manhaeve et al. [2021a] Badreddine et al. [2022] Ahmed et al. [2022a]	Pan et al. [2023] Dickens et al. [2024c] Olausson et al. [2023]

A "modeling paradigm" is a specification of the set of symbolic potentials and the domains of the potentials belonging to the set. I describe three foundational modeling paradigms in the following sections in increasing order of sophistication: *deep symbolic variables (DSVar)*, *deep symbolic parameters (DSPar)*, and *deep symbolic potentials (DSPot)*. Table 3.1 presents a summary of these modeling paradigms. It is important to note that many NeSy systems can represent multiple paradigms, such as DeepProbLog [Manhaeve et al., 2021a], Logic Tensor Networks [Badreddine et al., 2022], Semantic Probabilistic Layers [Ahmed et al., 2022a], and NeuPSL [Pryor et al., 2023a]. However, the examples listed are specific instances of the corresponding paradigm. While the properties and applications are generally representative, there are instances where a modeling paradigm may not precisely conform to the table.

3.2.1 Deep Symbolic Variables

The deep symbolic variables (DSVar) paradigm trains neural components efficiently with a loss that captures domain knowledge. Representative methods following this paradigm include semantic loss networks [Xu et al., 2018] and learning with logical constraints [Giunchiglia et al., 2022]. Concisely, the neural component directly

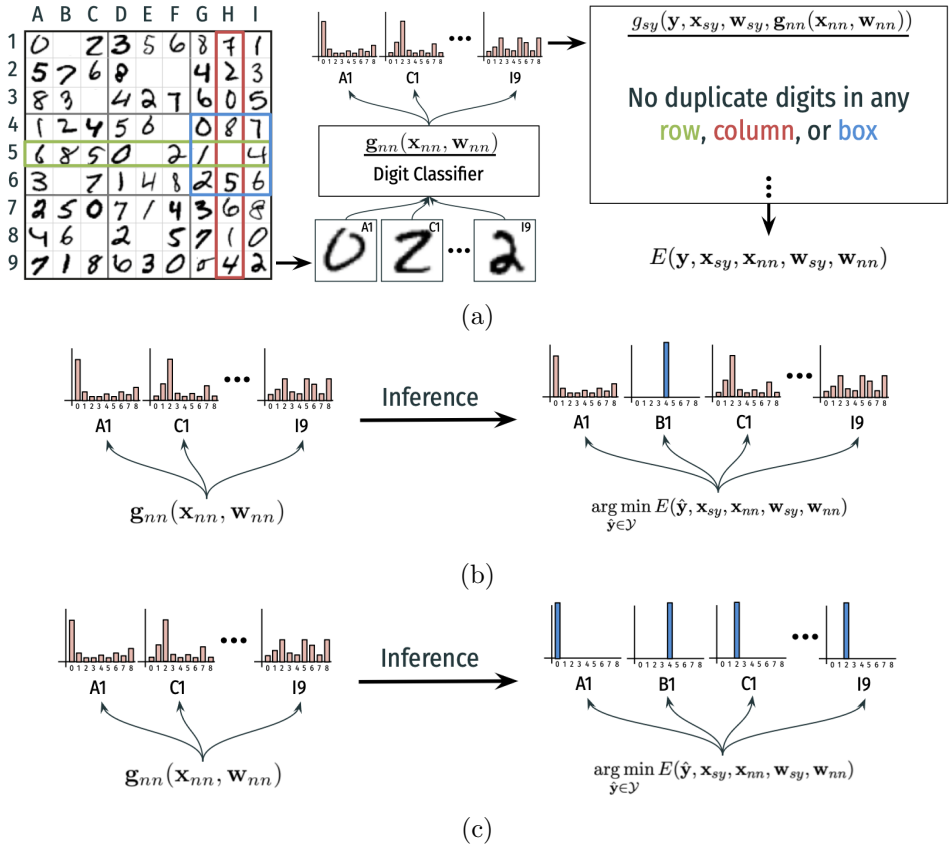


Figure 3.2: (a) A NeSy-EBM for solving a Sudoku board constructed from handwritten digits. The neural component classifies handwritten digits. Then, the symbolic component uses the digit classifications and Sudoku rules to quantify the compatibility of the inputs, neural predictions, and targets. (b) In the DSVar modeling paradigm inference process, the neural component predicts squares with digits, while the symbolic component measures incompatibility and predicts the latent (blank) squares. (c) In the DSPar modeling paradigm inference process, the neural component predicts squares with digits, and the symbolic component can alter these predictions to adhere to symbolic constraints.

predicts the values of targets in a single symbolic potential. In other words, there is a one-to-one mapping from the neural output to the targets. Note, however, that the mapping is not necessarily *onto*, i.e., there may be target variables without a corresponding neural output. For this discussion of modeling paradigms, I use the term "latent" to refer to target variables without a neural output in a DSVar model.

Definition 3

In the ***deep symbolic variables*** *(DSVar) modeling paradigm the symbolic potential set is a singleton* $\mathbf{\Psi} = \{\psi\}$ *with a trivial index set* $\mathbf{J}_{\mathbf{\Psi}} = \{1\}$ *such that* $\mathbf{\Psi}_1 = \psi$. *Further, the neural prediction is treated as a variable by the symbolic potential; thus*

$V_\psi = \mathcal{Y} \times \mathcal{X}_{sy} \times \mathbb{R}^{d_{nn}}$. *Then, the symbolic parameters are the symbolic weights,* $Params_\psi = \mathcal{W}_{sy}$. *The neural component controls the NeSy-EBM prediction via this function:*

$$I_{\mathcal{Y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \begin{cases} 0 & \mathbf{y}_i = [\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]_i \,, \forall i \in \{1, \cdots, d_{nn}\} \\ \infty & o.w. \end{cases}, \quad (3.10)$$

where $\mathbf{y}_i$ *and* $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})_i$ *denote the* i*'th entry of the variable and neural output vectors, respectively. Then, the symbolic component expressed via the symbolic potential is:*

$$\begin{aligned} & g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ & \quad := \psi([\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})] \,, \mathbf{w}_{sy}) + I_{\mathcal{Y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), \end{aligned} \quad (3.11)$$

where $[\cdot]$ *denotes concatenation.* □

The DSVar modeling paradigm typically yields the most straightforward prediction program compared to the other modeling paradigms. This is because the neural model fixes a subset of the decision variables, making the prediction program smaller. This is achieved by adding the function (Equation 3.10) in the definition above to the symbolic potential so infinite energy is assigned to variable values that do not match the neural model's predictions. However, for the same reason that this modeling paradigm typically has a simpler prediction program, the symbolic component cannot be used to resolve constraint violations made by the neural component. Rather, DSVar models rely on learning to train a neural component to adhere to constraints. For this reason, DSVar models typically have fast inference and learning processes but cannot be applied for constraint satisfaction, as stated in Table 3.1. The DSVar paradigm is demonstrated in the following example.

Example 1

Visual Sudoku [Wang et al., 2019] puzzle solving is the problem of recognizing handwritten digits in non-empty puzzle cells and reasoning with the rules of Sudoku (no

repeated digits in any row, column, or box) to fill in empty cells. Fig. 3.2 shows a partially complete Sudoku puzzle created with MNIST images [LeCun et al., 1998] and a NeSy-EBM designed for visual Sudoku solving. The neural component is a digit classifier predicting the label of MNIST images, and the symbolic component quantifies rule violations.

Formally, the target variables, $\mathbf{y}$*, are the categorical labels of both the handwritten digits and the empty entries in the puzzle, i.e., the latent variables. The symbolic inputs,* $\mathbf{x}_{sy}$*, indicate whether two puzzle positions are in the same row, column, or box. The neural model,* $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$*, is the categorical label of the handwritten digits predicted by the neural component. Then, the symbolic parameters,* $\mathbf{w}_{sy}$*, are used to shape the single symbolic potential function,* ψ*, that quantifies the amount of Sudoku rule violations.*

The DSVar modeling paradigm is applied to fit neural parameters with a knowledge-informed loss in a semi-supervised setting in the empirical analysis. However, neural model predictions cover a subset of the target values, and the model cannot resolve rule violations. Therefore, when the neural model predicts digit labels that violate a Sudoku rule, the predicted target variables will also violate the rule.

3.2.2 Deep Symbolic Parameters

The deep symbolic parameters (DSPar) modeling paradigm allows targets and neural predictions to be unequal or represent different concepts. Prominent NeSy frameworks supporting this technique include DeepProbLog [Manhaeve et al., 2021a], semantic probabilistic layers [Ahmed et al., 2022a], and logic tensor networks [Badreddine et al., 2022]. Succinctly, the neural component is applied as a parameter in the symbolic potential. This paradigm allows the symbolic component to correct constraint violations made by the neural component during prediction. For this reason, DSPar's inference and learning processes are typically more complex than the DSVar model but can perform constraint satisfaction, as indicated in Table 3.1.

Definition 4

In the ***deep symbolic parameters*** *(DSPar) modeling paradigm, the symbolic poten-*

tial set is a singleton $\mathbf{\Psi} = \{\psi\}$ *with a trivial index set* $\mathbf{J}_{\mathbf{\Psi}} = \{1\}$ *such that* $\mathbf{\Psi}_1 = \psi$. *Further, the neural prediction is treated as a parameter by the symbolic potential, thus* $Params_\psi = \mathcal{W}_{sy} \times \mathbb{R}^{d_{nn}}$. *Then the symbolic variables are the targets and the symbolic inputs:* $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$. *The symbolic component expressed via the single symbolic potential is:*

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \psi([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]). \quad \square$$

This paradigm is demonstrated in the following example.

Example 2

Again, consider the Visual Sudoku puzzle-solving problem illustrated in Fig. 3.2. As in the DSVar model, the neural component of the DSPar model is a digit classifier predicting the label of MNIST images. However, the digit classifications of the neural component are used as initial predictions in the symbolic component, as a prior for a probabilistic model. Then, the symbolic component is used to quantify rule violations as well as the difference between neural outputs and target variables.

The target variables, $\mathbf{y}$, *are the categorical labels of both the handwritten digits and the puzzle's empty entries. The symbolic inputs,* $\mathbf{x}_{sy}$, *indicate whether two puzzle positions are in the same row, column, or box. The neural model,* $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ *consists of the categorical labels of the handwritten digits predicted by the neural component. The symbolic parameters* $\mathbf{w}_{sy}$ *are used to shape the single symbolic potential function* ψ *that quantifies the amount of Sudoku rule violations.*

The DSPar modeling paradigm is widely applicable. For instance, the DSPar modeling paradigm is applied for constraint satisfaction, fine-tuning, few-shot, and semi-supervised settings in the empirical analysis. Note, however, that DSVar and DSPar models have only a single fixed symbolic potential. This property makes these paradigms well-suited for dedicated tasks but less applicable to open-ended settings, where the relevant domain knowledge depends on context. To address this challenge, the following modeling paradigm leverages generative modeling to perform in open-ended tasks.

3.2.3 Deep Symbolic Potentials

Deep-symbolic potentials (DSPot), the most advanced paradigm I propose, enhances deep models with symbolic reasoning tools. The Logic-LM pipeline proposed by Pan et al. (2023) is an excellent example of this modeling paradigm. At a high level, the neural component is a generative model that samples symbolic potentials from a set to define the symbolic component. Specifically, input data is used as context to retrieve relevant domain knowledge and formulate a program to perform inference in open-ended problems.

Definition 5

*In the **deep symbolic potentials** modeling paradigm, the symbolic potential set* $\mathbf{\Psi}$ *is the set of all potential functions that can be created by a NeSy framework.* $\mathbf{\Psi}$ *is indexed by the output of the neural component, i.e.,* $\mathbf{J}_\Psi = Range(\mathbf{g}_{nn})$ *and* $\mathbf{\Psi}_{\mathbf{g}_{nn}(\mathbf{x}_{nn},\mathbf{w}_{nn})}$ *is the potential function indexed by the neural prediction. The variable and parameter domains of the sampled symbolic potential are* $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$*, and* $Params_\psi = \mathcal{W}_{sy}$*, respectively. The symbolic component expressed via the symbolic potential is:*

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \mathbf{\Psi}_{\mathbf{g}_{nn}(\mathbf{x}_{nn},\mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy}). \quad \square$$

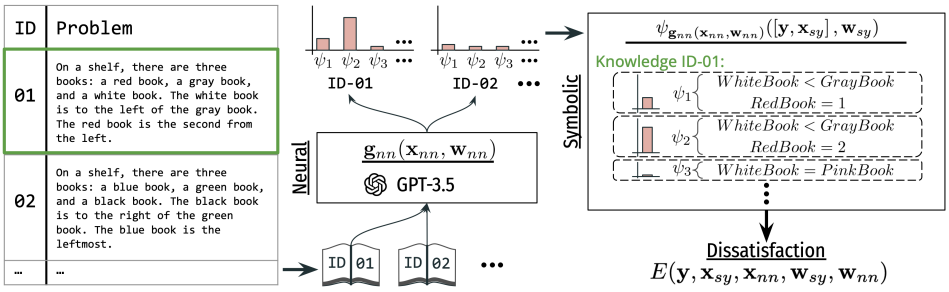


Figure 3.3: A deep symbolic potential model for answering questions about a set of objects' order described in natural language. The neural component is an LLM that generates syntax to create a symbolic potential. The symbolic potential is used to perform deductive reasoning and answer the question. See Example 3 for details.

This paradigm is demonstrated in the following example.

Example 3

Question answering is the problem of giving a response to a question posed in natural language. Fig. 3.3 shows a set of word problems asking for the order of a set of objects given information expressed in natural language and a NeSy-EBM designed for question answering. The neural component is a large language model (LLM) that is prompted with a word problem and tasked with generating a program within the syntax of a symbolic framework. The symbolic framework uses the generated program to instantiate a symbolic component used to perform deductive reasoning.

Formally, the target variables, $\mathbf{y}$*, represent object positions, and there is no symbolic input,* $\mathbf{x}_{sy}$*, in this example. The neural input,* $\mathbf{x}_{nn}$*, is a natural language prompt that includes the word problem. The neural model,* $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$*, is an LLM that generates syntax for a declarative symbolic modeling framework that creates the symbolic potential. For instance, the symbolic potential generated by the neural model* $\mathbf{\Psi}_{\mathbf{g}_{nn}(\mathbf{x}_{nn},\mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy})$ *could be the total amount of violation of arithmetic constraints representing ordering. Finally, the symbolic parameters,* $\mathbf{w}_{sy}$*, shape the symbolic potential function.*

DSPot is the only applicable paradigm for truly open-ended tasks. Moreover, DSPot enhances generative models, such as LLMs, with consistent symbolic reasoning capabilities. This feature is demonstrated in constraint satisfaction and joint reasoning experiments in the empirical analysis. DSPot's limitation is that the neural component must learn to sample from a large potential set. For instance, in the example, an LLM must reliably generate syntax to define a symbolic potential for solving the word problem. LLMs require a substantial amount of computational resources to train and then fine-tune for a specific NeSy framework. Furthermore, the inference time is dependent on the sampled symbolic potential. If the neural component samples a complex symbolic potential, inference may be slow. These strengths and limitations are outlined in Table 3.1.

Chapter 4

A Suite of Learning Techniques for NeSy

Having identified a variety of modeling paradigms, I turn to learning. This chapter formalizes the NeSy-EBM learning problem, identifies challenges, and proposes effective solutions. At a high level, NeSy-EBM learning is finding weights of an energy function that associates higher compatibility scores (lower energy) to targets and neural outputs near their true labels provided in training data. Further, predictions with NeSy-EBMs are obtained by minimizing a complex mathematical program, raising several obstacles to learning. For instance, NeSy-EBM predictions may not be differentiable with respect to the model parameters, and a direct application of automatic differentiation may not be possible or may fail to produce principled descent directions for the learning objective. Moreover, I will show that even when predictions are differentiable, their gradients are functions of properties of the energy function at its minimizer that are prohibitively expensive to compute. I create general and principled learning frameworks for NeSy-EBMs that address these challenges.

This chapter is organized into four sections. I begin with preliminary notation and a general definition of NeSy-EBM learning. Then, I present a classification of learning losses and establish differentiability properties of NeSy-EBMs. Finally, the learning losses motivate and organize the exposition of four NeSy-EBM learning frameworks, one for learning the neural and symbolic weights separately and three

for end-to-end learning.

4.1 NeSy-EBM Learning

I use the following notation and general definition of NeSy-EBM learning throughout this section. The training dataset, denoted by $\mathcal{S}$, is comprised of P samples and indexed by $\{1, \cdots, P\}$. Each sample, $\mathcal{S}_i$ where $i \in \{1, \cdots, P\}$, is a tuple of *inputs*, *labels*, and *latent variable domains*. Sample *inputs* consist of neural inputs, $\mathbf{x}^i_{nn}$ from $\mathcal{X}_{nn}$, and symbolic inputs, $\mathbf{x}^i_{sy}$ from $\mathcal{X}_{sy}$. Similarly, sample *labels* consist of neural and symbolic labels, which are truth values corresponding to a subset of the neural predictions and target variables, respectively. Neural labels, denoted by $\mathbf{t}^i_{nn}$, are $d^i_{nn} \leq d_{nn}$ dimensional real vectors from a domain $\mathcal{T}^i_{nn}$, i.e., $\mathbf{t}^i_{nn} \in \mathcal{T}^i_{nn} \subseteq \mathbb{R}^{d^i_{nn}}$. Target labels, denoted by $\mathbf{t}^i_{\mathcal{Y}}$, are from a domain $\mathcal{T}^i_{\mathcal{Y}}$ that is a $d_{\mathcal{T}^i_{\mathcal{Y}}} \leq d_{\mathcal{Y}}$ dimensional subspace of the target domain $\mathcal{Y}$, i.e., $\mathbf{t}^i_{\mathcal{Y}} \in \mathcal{T}^i_{\mathcal{Y}}$. Lastly, the neural and symbolic *latent variable domains* are subspaces of the range of the neural component and the target domain, respectively, corresponding to the set of unlabeled variables. The range of the neural component, $\mathbb{R}^{d_{nn}}$, is a superset of the Cartesian product of the neural latent variable domain, denoted by $\mathcal{Z}^i_{nn}$, and $\mathcal{T}^i_{nn}$, i.e., $\mathbb{R}^{d_{nn}} \supseteq \mathcal{T}^i_{nn} \times \mathcal{Z}^i_{nn}$. Similarly, the target domain $\mathcal{Y}$ is a superset of the Cartesian product of the latent variable domain, denoted by $\mathcal{Z}^i_{\mathcal{Y}}$, and $\mathcal{T}^i_{\mathcal{Y}}$, i.e., $\mathcal{Y} \supseteq \mathcal{T}^i_{\mathcal{Y}} \times \mathcal{Z}^i_{\mathcal{Y}}$. With this notation, the training dataset is expressed as follows:

$$\mathcal{S} := \{(\mathbf{t}^1_{\mathcal{Y}}, \mathbf{t}^1_{nn}, \mathcal{Z}^1_{nn}, \mathcal{Z}^1_{\mathcal{Y}}, \mathbf{x}^1_{sy}, \mathbf{x}^1_{nn}), \cdots, (\mathbf{t}^P_{\mathcal{Y}}, \mathbf{t}^P_{nn}, \mathcal{Z}^P_{nn}, \mathcal{Z}^P_{\mathcal{Y}}, \mathbf{x}^P_{sy}, \mathbf{x}^P_{nn})\}. \tag{4.1}$$

A learning objective, denoted by $\mathcal{L}$, is a functional that maps an energy function and a training dataset to a scalar value. Formally, let $\mathcal{E}$ be a family of energy functions indexed by weights from $\mathcal{W}_{sy} \times \mathcal{W}_{nn}$:

$$\mathcal{E} := \{E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \,|\, (\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}\}. \tag{4.2}$$

Then, a learning objective is the function:

$$\mathcal{L} : \mathcal{E} \times \{\mathcal{S}\} \to \mathbb{R}. \tag{4.3}$$

Learning objectives follow the standard empirical risk minimization framework and are separable over elements of $\mathcal{S}$ as a sum of *per-sample loss functionals* denoted by L^i for each $i \in \{1, \cdots, P\}$. A loss functional for the sample $\mathcal{S}_i \in \mathcal{S}$ is the function:

$$L^i : \mathcal{E} \times \{\mathcal{S}_i\} \to \mathbb{R}. \tag{4.4}$$

A regularizer, denoted by $\mathcal{R} : \mathcal{W}_{sy} \times \mathcal{W}_{nn} \to \mathbb{R}$, is added to the learning objective and NeSy-EBM learning is the following minimization problem:

$$\begin{aligned} &\underset{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}}{\arg\min} \mathcal{L}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \\ &= \underset{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}}{\arg\min} \frac{1}{P} \sum_{i=1}^{P} L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}). \end{aligned} \tag{4.5}$$

4.2 Learning Losses

A NeSy-EBM learning loss functional, L^i, is separable into three parts: *neural*, *value-based*, and *minimizer-based* losses. In this section, I formally define each of the three loss types. At a high level, the neural loss measures the quality of the neural component independent from the symbolic component. Then, the value-based and minimizer-based losses measure the quality of the NeSy-EBM as a whole. Moreover, value-based and minimizer-based losses are functionals mapping a parameterized energy function and a training sample to a real value and are denoted by $L_{Val} : \mathcal{E} \times \mathcal{S} \to \mathbb{R}$ and $L_{Min} : \mathcal{E} \times \mathcal{S} \to \mathbb{R}$, respectively. The learning loss components are aggregated via summation:

$$
\begin{aligned}
& L^i(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) && && (4.6)\\
&= L_{NN}(\mathbf{g}_{nn}(\mathbf{x}^i_{nn},\mathbf{w}_{nn}),\mathbf{t}^i_{nn}) && \text{Neural}\\
&\quad + L_{Val}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) && \text{Value-Based}\\
&\quad + L_{Min}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) && \text{Minimizer-Based}
\end{aligned}
$$

4.2.1 Neural Learning Losses

Neural learning losses are scalar functions of the neural network output and the neural labels and are denoted by $L_{NN} : \text{Range}(\mathbf{g}_{nn}) \times \mathcal{T}^i_{nn} \to \mathbb{R}$. For example, a neural learning loss may be the familiar binary cross-entropy loss applied in many categorical prediction settings. Minimizing a neural learning loss with respect to neural component parameters is achievable via backpropagation and standard gradient-based algorithms.

4.2.2 Value-Based Learning Losses

Value-based learning losses depend on the model weights strictly via minimizing values of an objective defined with the energy. More formally, denote an objective function by f, which maps a compatibility score, target variables, and the training sample to a scalar value:

$$
f : \mathbb{R} \times \mathcal{Y} \times \{\mathcal{S}_i\} \to \mathbb{R}. \tag{4.7}
$$

An *optimal value-function*, denoted by V, is the value of f composed with the energy function and minimized over the target variables:

$$
\begin{aligned}
V(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i) &:= \min_{\hat{\mathbf{y}}\in\mathcal{Y}} f\left(E(\hat{\mathbf{y}},\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\hat{\mathbf{y}},\mathcal{S}_i\right)\\
&:= \min_{\hat{\mathbf{y}}\in\mathcal{Y}} f(g_{sy}(\hat{\mathbf{y}},\mathbf{x}^i_{sy},\mathbf{w}_{sy},\mathbf{g}_{nn}(\mathbf{x}^i_{nn},\mathbf{w}_{nn})),\hat{\mathbf{y}},S_i) \qquad (4.8)
\end{aligned}
$$

Value-based learning losses are functions of one or more optimal value functions. In this work, I consider three instances of optimal value functions: 1) *latent*, $V_{\mathcal{Z}}$, 2) *full*, $V_{\mathcal{Y}}$, 3) and *convolutional*, V_{conv}. The latent optimal value function is the minimizing value of the energy over the latent targets. Further, the labeled targets are fixed to their true values using the following indicator function:

$$I_{\mathcal{T}_{\mathcal{Y}}^i}(\mathbf{y}, \mathbf{t}_{\mathcal{Y}}^i) := \begin{cases} 0 & \mathbf{y} = \mathbf{t}_{\mathcal{Y}}^i \\ \infty & \text{o.w.} \end{cases}. \tag{4.9}$$

The full optimal value function is the minimizing value of the energy over all of the targets. Lastly, the convolutional optimal value function is the infimal convolution of the energy function and a function $d : \mathcal{Y} \times \mathcal{Y} \to \mathcal{R}$ scaled by a positive real value $\lambda \in \mathcal{R}$. Formally:

$$\begin{aligned} V_{\mathcal{Z}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + I_{\mathcal{T}_{\mathcal{Y}}^i}(\hat{\mathbf{y}}, \mathbf{t}_{\mathcal{Y}}^i), \\ &= \min_{\hat{\mathbf{z}} \in \mathcal{Z}_{\mathcal{Y}}^i} E((\mathbf{t}_{\mathcal{Y}}^i, \hat{\mathbf{z}}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \qquad \text{latent} \end{aligned} \tag{4.10}$$

$$V_{\mathcal{Y}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) := \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \qquad \text{full} \tag{4.11}$$

$$V_{conv}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i; \mathbf{y}, \lambda) := \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + \lambda \cdot d(\hat{\mathbf{y}}, \mathbf{y}). \qquad \text{convolutional} \tag{4.12}$$

An illustration of an example latent optimal value-function is provided in Fig. 4.1. Intuitively, the latent optimal value-function is the greatest lower bound of the set of symbolic components defined for each latent variable.

The simplest value-based learning loss is the *energy loss*, denoted by L_{Energy}. The energy loss is the latent optimal value function,

$$L_{Energy}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) := V_{\mathcal{Z}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i). \tag{4.13}$$

Minimizing the energy loss encourages the parameters of the energy function to produce low energies given the observed true values of the input and target variables.

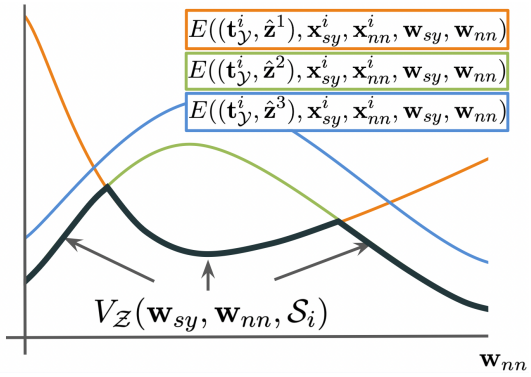


Figure 4.1: An illustrated example of a latent optimal value-function with a scalar neural component output and a discrete latent variable domain $\mathcal{Z} := \{\hat{\mathbf{z}}^1, \hat{\mathbf{z}}^2, \hat{\mathbf{z}}^3\}$.

This loss is motivated by the intuition that the energy should be low for the desired values of the targets. Notice, however, that the loss does not consider the energy of incorrect target variable values. An extreme illustration of the issue this causes involves two energy functions. In the first function, the minimizing point corresponds to the desired true values of the targets, while in the second function, the maximizing point corresponds to the desired true values of the targets. Despite these differences, both functions could technically have the same energy loss; however, the first energy function is clearly preferred. Thus, the energy loss does not universally lead to energy functions with better predictions.

The *Structured Perceptron* loss, denoted by L_{SP}, pushes the energy of the current energy minimizer up and the energy of the true values of the targets down [LeCun et al., 1998, Collins, 2002]. Specifically, the structured perceptron loss is the difference between the latent and full optimal value functions,

$$L_{SP}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) := V_{\mathcal{Z}}(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i) - V_{\mathcal{Y}}(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i). \tag{4.14}$$

Although the structured perceptron loss will technically encourage the target's desired values to be an energy minimizer, i.e., a valid prediction, it still has degenerate solutions for some energy function architectures. For instance, one could minimize the energy for all target values, leading to a collapsed energy function (equal energy for all targets) with no predictive power.

The energy and structured perceptron losses require regularization and specific

energy architectures to work well in practice. For instance, energy architectures that naturally push up on other target values when pushing down on the desired targets. Energies with limited total energy mass are examples of functions with this property.

The gradient of a value-based loss with respect to neural and symbolic weights is non-trivial since both the energy function and the point the energy function is evaluated at are dependent on the neural output and symbolic weights, as exemplified by the definition of an optimal value function in (4.8). Nonetheless, Milgrom and Segal (2002) delivers a general theorem providing the gradient of optimal value-functions with respect to problem parameters, if they exist. I specialize their result in the following theorem for optimal value-functions of NeSy-EBMs.

Theorem 6

Consider the weights $\mathbf{w}_{sy} \in \mathcal{W}_{sy}$ *and* $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ *and the sample*

$$\mathcal{S}_i = (\mathbf{t}^i_{\mathbf{y}}, \mathbf{t}^i_{nn}, \mathcal{Z}^i_{nn}, \mathcal{Z}^i_{\mathcal{Y}}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}) \in \mathcal{S}.$$

Suppose there exists a minimizer of the objective function f*,*

$$\mathbf{y}^* \in \arg\min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(E(\hat{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \hat{\mathbf{y}}, \mathcal{S}_i),$$

such that $f(E(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{y}^*, \mathcal{S}_i)$ *is finite.*

If the optimal value-function:

$$\begin{aligned} V(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(E(\hat{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \hat{\mathbf{y}}, \mathcal{S}_i), \\ &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(g_{sy}(\hat{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})), \hat{\mathbf{y}}, S_i), \end{aligned}$$

is differentiable with respect to the neural weights, $\mathbf{w}_{nn}$*, then the gradient of* V *with respect to* $\mathbf{w}_{nn}$ *is:*

$$\nabla_{\mathbf{w}_{nn}} V(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) = \frac{\partial}{\partial 1} f(E(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{y}^*, \mathcal{S}_i) \cdot \nabla_5 E(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \tag{4.15}$$

where $\frac{\partial}{\partial 1} f$ *is the partial derivative of* f *with respect to its* $1st$ *argument, and* $\nabla_5 E$ *is*

the gradient of the energy with respect to its $5th$ *argument with all other arguments fixed.*

Similarly, if V *is differentiable with respect to the symbolic weights,* $\mathbf{w}_{sy}$*, then the gradient of* V *with respect to* $\mathbf{w}_{sy}$ *is:*

$$\begin{aligned}&\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) \\ &\quad = \frac{\partial}{\partial 1} f(E(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{y}^*, \mathcal{S}_i) \cdot \nabla_4 E(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}).\end{aligned}$$

Proof. I first establish the partial derivative of the optimal value-function with respect to each component of the neural output, $\mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})$. Then, I use the chain rule to derive the expression for the gradient of the optimal value-function with respect to the neural weights, $\mathbf{w}_{nn}$.

For an arbitrary index $j \in \{1, \cdots, d_{nn}\}$, let $\mathbf{e}^j$ be the $j'th$ standard basis vector of $\mathbb{R}^{d_{nn}}$, i.e., $\mathbf{e}^j \in \mathbb{R}^{d_{nn}}$ such that $\mathbf{e}^j_j = 1$ and $\mathbf{e}^j_k = 0$ for $k \neq j$. Further, to clarify the relationship between the optimal value-function and the neural component output, define the following function:

$$\begin{aligned}\overline{V} : \mathcal{W}_{sy} \times \mathbb{R}^{d_{nn}} \times \mathcal{S}_i &\to \mathbb{R} \\ (\mathbf{w}_{sy}, \mathbf{u}, S_i) &\mapsto \min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(g_{sy}(\hat{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{u}), \hat{\mathbf{y}}, \mathcal{S}_i)\end{aligned}$$

In other words, the optimal value-function, V, is equal to $\overline{V}$ evaluated at the neural output:

$$V(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) \triangleq \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}), \mathcal{S}_i).$$

For any $\delta \in \mathbb{R}$, by definition:

$$\begin{aligned}&f(g_{sy}(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}) + \delta \mathbf{e}^j), \mathbf{y}^*, S_i) - f(g_{sy}(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})), \mathbf{y}^*, S_i) \\ &\geq \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}) + \delta \mathbf{e}^j, S_i) - \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}), S_i).\end{aligned}$$

For $\delta \neq 0$, dividing both sides by δ and taking the limit as $\delta \to 0+$ and as $\delta \to 0-$

yields upper and lower bounds relating partial derivatives of f to $\overline{V}$ when f and $\overline{V}$ are right and left hand differentiable, respectively.

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j+} f(g_{sy}(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})), \mathbf{y}^*, S_i) \\
&\quad \geq \frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j+} \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}), S_i), \\
&\frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j-} f(g_{sy}(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})), \mathbf{y}^*, S_i) \\
&\quad \leq \frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j-} \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}), S_i),
\end{aligned}$$

Then, by the squeeze theorem, the partial derivatives of $\overline{V}$ with respect to each component of the neural output when $\overline{V}$ is differentiable are:

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j} f(g_{sy}(\mathbf{y}^*, \mathbf{x}^i_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})), \mathbf{y}^*, S_i) \\
&\quad = \frac{\partial}{\partial \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})_j} \overline{V}(\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn}), S_i).
\end{aligned}$$

Then, the chain rule of differentiation and the partial derivatives of $\overline{V}$ with respect to each component of the neural output derived above yields the gradient in (4.15).

A similar approach is used to obtain gradients with respect to symbolic weights in (4.16). □

Theorem 6 holds for arbitrary target variable domains and energy functions and is, therefore, widely applicable. However, it is important to emphasize that Theorem 6 states *if* the value-function is differentiable, *then* the gradients have the form provided in (4.15) and (4.16). Milgrom and Segal (2002) also provide sufficient conditions for guaranteeing the differentiability of optimal value-functions with arbitrary decision variable domains. Beyond Milgrom and Segal's (2002) work, there is extensive literature on analyzing the sensitivity of optimal value-functions and guaranteeing their differentiability, including the seminal papers of Danskin [1966] on parameterized objective functions and Rockafellar [1974] for parameterized constraints. I direct the reader to the cited articles for properties that guarantee differentiability of value-functions and, hence, NeSy-EBM value-based losses.

The conditions ensuring differentiability of the optimal value-functions as well as the tractability of computing the gradient of the symbolic component with respect to its arguments in (4.15) and (4.16) directly connect to the energy function architecture and modeling paradigms discussed in the previous chapter. Specifically, if principled gradient-based learning is desired, then practitioners must design the symbolic potential such that it is 1) differentiable with respect to the neural output and symbolic potentials, 2) the gradient of the symbolic potential with respect to its arguments is tractable, and 3) it satisfies sufficient conditions for ensuring differentiability of its minimizing value over the targets.

Performance metrics are not always aligned with value-based losses. Moreover, they are known to have degenerate solutions [LeCun et al., 2006, Pryor et al., 2023a]. For example, without a carefully designed inductive bias, the energy loss in (4.13) may only learn to reduce the energy of all target variables without improving the predictive performance of the NeSy-EBM. One fundamental cause of this issue is that value-based losses are not directly functions of the NeSy-EBM prediction as defined in (3.4), i.e., value-based losses are not functions of an energy minimizer, which is what I turn to next.

4.2.3 Minimizer-Based Learning Losses

A *minimizer-based* loss is a composition of a differentiable loss, such as cross-entropy or mean squared error, with the energy minimizer. Intuitively, minimizer-based losses penalize parameters yielding predictions distant from the labeled training data. In the remainder of this section, I formally define minimizer-based learning losses. Further, for completeness, I derive general expressions for gradients of minimizer-based losses with respect to symbolic and neural weights. However, I will show that direct computation of minimizer-based loss gradients requires prohibitive assumptions on the energy function and can be impractical to compute. Moreover, the derivation of the gradients motivates learning algorithms that do not perform direct gradient descent on minimizer-based losses. For this reason, in the following section I propose algorithms that do not require minimizer gradients.

To ensure a minimizer-based loss is well-defined, I assume a unique energy minimizer exists, denoted by $\mathbf{y}^*$, for every training sample. This assumption is formalized below.

Assumption 1

The energy function is minimized over the targets at a single point for every input and weight and is, therefore, a function:

$$\begin{aligned}\mathbf{y}^* : \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn} &\to \mathcal{Y} \\ (\mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) &\mapsto \arg\min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})\end{aligned}$$

Under Assumption 1, d is a mapping of targets and labels to a scalar value:

$$d : \mathcal{Y} \times \mathcal{T}_{\mathcal{Y}}^i \to \mathbb{R}, \tag{4.16}$$

and a minimizer-based loss is a composition of d and $\mathbf{y}^*$:

$$\begin{aligned}L_{Min}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) &:= d(\arg\min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathcal{Y}}^i) \\ &:= d(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathcal{Y}}^i)\end{aligned} \tag{4.17}$$

To ensure principled direct gradient-based learning, it is necessary to assume that the minimizer is differentiable.

Assumption 2

The minimizer, $\mathbf{y}^*$, is differentiable with respect to the weights at every point in $\mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn}$.

Under Assumption 2, the chain rule of differentiation yields the gradient of a

minimizer-based loss with respect to the neural and symbolic weights:

$$\nabla_{\mathbf{w}_{sy}} L_{Min}(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})), \mathbf{t}^i_{\mathcal{Y}})$$
$$= \nabla_3 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})^T \nabla_1 d(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}^i_{\mathcal{Y}}), \tag{4.18}$$

$$\nabla_{\mathbf{w}_{nn}} L_{Min}(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})), \mathbf{t}^i_{\mathcal{Y}})$$
$$= \nabla_4 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})^T \nabla_1 d(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}^i_{\mathcal{Y}}), \tag{4.19}$$

where $\nabla_3 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ and $\nabla_4 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ are the Jacobian matrices of the unique energy minimizer with respect to the third and fourth arguments of $\mathbf{y}^*$, the symbolic and neural weights, respectively, and $\nabla_1 d(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}^i_{\mathcal{Y}})$ is the gradient of the supervised loss with respect to its first argument.

A primary challenge of minimizer-based learning is computing the Jacobian matrices of partial derivatives, $\nabla_3 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ and $\nabla_4 \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$. To derive explicit expressions for them typically demands the following additional assumption on the continuity properties of the energy function.

Assumption 3

The energy, E, is twice differentiable with respect to the targets at the minimizer, $\mathbf{y}^*$, and the Hessian matrix of second-order partial derivatives with respect to the targets, $\nabla_{1,1} E(\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$, is invertible. Further, the minimizer is the unique target satisfying first-order conditions of optimality, i.e.,

$$\forall \mathbf{y} \in \mathcal{Y}, \quad \nabla_1 E(\mathbf{y}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = 0 \iff \mathbf{y} = \mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$$

Assumption 3 is satisfied by energy functions that are, for instance, smooth and strongly convex in the targets. Under Assumption 3, the first-order optimality condition establishes the minimizer as an implicit function of the weights, and implicit

differentiation yields the following equalities:

$$\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})\nabla_3\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}) \quad (4.20)$$
$$= -\nabla_{1,4}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})$$
$$\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})\nabla_4\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}) \quad (4.21)$$
$$= -\nabla_{1,5}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})$$

Solving for the Jacobians of the minimizer:

$$\nabla_3\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}) = -\big(\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})^{-1} \quad (4.22)$$
$$\nabla_{1,4}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})\big),$$
$$\nabla_4\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}) = -\big(\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})^{-1} \quad (4.23)$$
$$\nabla_{1,5}E(\mathbf{y}^*(\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})\big).$$

The Jacobians in (4.22) and (4.23) applied to (4.18) and (4.19), respectively, are referred to as hypergradients in the machine learning literature and are utilized in hyperparameter optimization and meta-learning [Do et al., 2007, Pedregosa, 2016, Rajeswaran et al., 2019]. Oftentimes, approximations of the (inverse) Hessian matrices are made to estimate the hypergradient.

4.3 Learning Algorithms

Next, I present four principled techniques for learning the neural and symbolic weights of a NeSy-EBM to minimize the losses introduced in the previous section: 1) Modular, 2) Gradient Descent, 3) Bilevel Value-Function Optimization, and 4) Stochastic Policy Optimization. The four techniques are defined, and I discuss their strengths and limitations in relation to the motivating applications in Section 1.1 and modeling paradigms in Section 3.2.

4.3.1 Modular Learning

The first and most straightforward NeSy-EBM learning technique is to train and connect the neural and symbolic components as independent modules. For instance, the neural component can be trained via backpropagation and Adam to optimize a neural loss given neural labels. Then, the symbolic component can be trained using an appropriate black-box or gradient-based method to optimize a value or minimizer-based loss. The neural component weights are frozen during the symbolic weight learning process. In this subsection I first discuss the general tradeoffs of modular learning techniques and then introduce three black-box methods for symbolic parameter learning. Black-box approaches make little to no assumptions about the learning objective and symbolic component architecture and are feasible for low-dimensional parameters spaces. Black-box approaches are therefore used to optimize losses directly related to the downstream task of the model. The gradient-based symbolic parameter learning algorithms that will be introduced in the following sections are also applicable to modular learning.

By definition, modular learning algorithms are not trained end-to-end, i.e., the neural and symbolic parameters are not jointly optimized to minimize the learning loss. For this reason, modular approaches may struggle to find a weight setting with a learning loss as low as end-to-end techniques. Moreover, modular approaches are not suitable for every motivating NeSy usecase presented in Section 1.1, for instance fine-tuning and adaptation. Additionally, they require labels to train the neural component. Thus, modular learning is not used to learn neural parameters in unsupervised or semi-supervised settings.

Nevertheless, modular learning approaches are appealing and widely used for their simplicity and general applicability. Importantly, no assumptions are made about the neural-symbolic interface; hence, modular learning is effective for every modeling paradigm presented in Section 3.2. Notably, minimizers and value-functions of DSPot models are typically non-differentiable with respect to the neural weights due to the complex neural-symbolic interface. However, because modular techniques are not end-to-end, this is not an issue. Moreover, modular learning can be used to

train a NeSy-EBM for constraint satisfaction and joint reasoning, zero-shot reasoning, and reasoning with noisy data.

Random Grid Search

Algorithm 1 Random Grid Search for Symbolic Parameter Learning

Require: Parameter Grid: $\tilde{\mathcal{W}}_{sy}$, Max Samples: b

$\mathbf{W}_{explored} \leftarrow \{\}$
for all $k \in \{1, \cdots, b\}$ **do**
  $\mathbf{w}_{sy}^{(k)} \leftarrow \text{RandomSample}(\tilde{\mathcal{W}}_{sy} \setminus \mathbf{W}_{explored})$
  if $(k = 1)$ or $\left(\gamma(\mathbf{y}^*(\mathbf{w}_{sy}^{(k)})) < \gamma(\mathbf{y}^*(\mathbf{w}_{sy}^*))\right)$ **then**
    $\mathbf{w}_{sy}^* \leftarrow \mathbf{w}_{sy}^{(k)}$
  $\mathbf{W}_{explored} \leftarrow \mathbf{W}_{explored} \cup \left\{\mathbf{w}_{sy}\right\}$
return $\mathbf{w}_{sy}^*$

The most straightforward approach to fitting the symbolic parameters of a NeSy-EBM is an exploration over a finite grid of weight configurations $\tilde{\mathcal{W}}_{sy} \subset \mathcal{W}_{sy}$. For each configuration, $\mathbf{w}_{sy} \in \tilde{\mathcal{W}}_{sy}$, $\gamma(\mathbf{y}^*(\mathbf{w}_{sy}))$ is evaluated. Finally, the weight configuration with the lowest objective value is selected.

An exhaustive grid search is usually infeasible due to a combinatorial explosion in the grid size with respect to the dimension of the parameter space. Thus, to make the approach tractable, only b unique samples from $\tilde{\mathcal{W}}_{sy}$ are evaluated. This approach is called *random grid search* (RGS). The complete algorithm for RGS is shown in Algorithm 1.

Continuous Random Search

Algorithm 2 Continuous Random Search for Symbolic Parameter Learning

Require: Dirichlet Parameter: $\mathrm{A} \in \mathbb{R}_+^r$, Max Samples: b

for all $k \in \{1, \cdots, b\}$ **do**
  $\mathbf{w}_{sy}^{(k)} \sim \text{Dirichlet}(\mathrm{A})$;
  if $(k = 1)$ or $\left(\gamma(\mathbf{y}^*(\mathbf{w}_{sy}^{(k)})) < \gamma(\mathbf{y}^*(\mathbf{w}_{sy}^*))\right)$ **then**
    $\mathbf{w}_{sy}^* \leftarrow \mathbf{w}_{sy}^{(k)}$;
return $\mathbf{w}_{sy}^*$

A primary difficulty of applying RGS is defining a grid over the search space that includes a variety of weights yielding different predictions. While specifying a grid

might seem straightforward, several properties of the symbolic parameter space make specifying the right grid non-trivial. For instance, the symbolic parameter domain for symbolic components intantiated by the NeSy-EBM library I introduce in Chapter 5 is the probability simplex. In this case, one may be tempted to simply define a grid of evenly spaced points in the unit hypercube and then project the point onto the simplex. However, this leads to a sampling bias towards configurations that are near the center of the simplex.

Continuous random search (CRS) is similar to RGS in that, rather than exploring the entire space, a finite number of weight configurations are chosen for evaluation and the highest-performing configuration is returned. The difference is that CRS does not require a discrete grid of weights. Rather, CRS samples continuous points from the symbolic parameter space. For weights constrained to the r-dimensional unit simplex, points are sampled from a r-dimensional Dirichlet distribution, and the configuration with the lowest objective value is returned. The Dirichlet distribution is parametrized by the r-dimensional concentration hyperparameter $A \in \mathbb{R}^r_+$. The complete algorithm is provided in Algorithm 2

Bayesian Optimization

Algorithm 3 Bayesian Optimization for Symbolic Parameter Learning

Require: Dirichlet Parameter: A $\in \mathbb{R}^r_+$, Initial Sample Size: B, Max Iterations: $MaxIter$

$\mathbf{W}_{InitialSample} \leftarrow \{\mathbf{w}_{sy,1}, \cdots, \mathbf{w}_{sy,B} \,|\, \mathbf{w}_{sy,1} \sim Dirichlet(A)\}$;
for all $k \in \{1, \cdots, MaxIter\}$ **do**
  $\mathbf{w}^{(k)}_{sy} \leftarrow \arg\max_{\mathbf{w}_{sy} \in \mathbf{W}_{InitialSample}} \alpha(\mathbf{w}_{sy})$;
  Update GPR model with $\gamma(\mathbf{y}^*(\mathbf{w}^*_{sy}))$;
  if $(k = 1)$ or $\left(\gamma(\mathbf{y}^*(\mathbf{w}^{(k)}_{sy})) < \gamma(\mathbf{y}^*(\mathbf{w}^*_{sy}))\right)$ **then**
    $\mathbf{w}^*_{sy} \leftarrow \mathbf{w}^{(k)}_{sy}$
return $\mathbf{w}^*_{sy}$

Previously discussed black box symbolic weight learning approaches make no assumptions about the search space and the evaluation function. The exploration process of black-box methods can be made more efficient by making a reasonable assumption that the objective value of two weight configurations $\mathbf{w}_{sy,1}$ and $\mathbf{w}_{sy,1}$

are likely to be similar ($\gamma(\mathbf{w}_{sy,1}) \approx \gamma(\mathbf{w}_{sy,1})$) if the distance between them is small. This implies that when searching the symbolic parameter space, one could use previous observations to decide either to continue exploring or exploit and sample configurations near previous weights that performed well.

Bayesian Optimization for Weight Learning (BOWL) uses GPR (Section 2.4) to build a probabilistic model of the objective function over the symbolic parameter space. The algorithm for symbolic parameters constrained to the r-dimensional probability simplex is detailed in Algorithm 3. First, an initial sample of B symbolic weight configurations is sampled from a r-dimensional Dirichlet distribution parameterized by the concentration hyperparameter $A \in \mathbb{R}_+^r$: $Dirichlet(A)$. Then, for each iteration, a weight configuration is chosen using an acquisition function, denoted by $\alpha(\mathbf{w}_{sy}) : \Delta^r \to \mathbb{R}$. Next, the objective is evaluated and used to update the GPR model of the objective function. Finally, the weight configuration that resulted in the best value for γ after a specified number of iterations is returned. To adapt Algorithm 3 to a different symbolic parameter space only the sampling process on line 1 needs to be change accordingly.

In this dissertation, I use the *squared exponential kernel*, k_{SE}:

$$k(\mathbf{w}_{sy,i}, \mathbf{w}_{sy,j}) = k_{SE}(\mathbf{w}_{sy,i}, \mathbf{w}_{sy,j}) = \tilde{\sigma} \cdot exp\{\frac{\|\mathbf{w}_{sy,i} - \mathbf{w}_{sy,j}\|_2^2}{2\rho^2}\},$$

where $\tilde{\sigma}$ is the *amplitude* parameter and ρ is the *characteristic length-scale* parameter of k_{SE}. The scaling factor ρ affects the smoothness of the approximation (the larger the value, the more smooth the approximation), and the number of iterations that are required to create a low variance approximation of the objective function.

As mentioned in Section 2.4.2, the squared exponential kernel is a stationary kernel. In fact, the kernel can be written as a function of only the absolute value of the Euclidean distance of its inputs. Kernels with this property are called isotropic, or radial basis functions Rasmussen and Williams [2005]. Furthermore, the squared exponential kernel approximates the objective γ with a smooth function. This approximation is justified by the continuity properties proven for the NeSy-EBM inference program of NeuPSL.

4.3.2 Gradient Descent

A conceptually simple but oftentimes difficult in-practice technique for end-to-end NeSy-EBM training is direct gradient descent. Specifically, the gradients derived in the previous section are directly used with a gradient-based algorithm to optimize a NeSy-EBM loss with respect to both the neural and symbolic weights. Backpropagation and Theorem 6 produce relatively inexpensive gradients for neural and value-based losses for a general class of NeSy-EBMs. Moreover, for a smaller family of NeSy-EBMs, gradients of energy minimizers exist and may be cheap to compute. For instance, if the energy minimizer is determined via a simple closed-form expression (e.g., if inference is an unconstrained strongly convex quadratic program or a finite computation graph).

As shown in Section 4.2, learning loss gradients for fully expressive NeSy-EBMs only exist under certain conditions. Further, computing the gradients generally requires expensive second-order information about the energy function at the minimizer. For this reason, direct gradient descent only applies to a relatively small class of NeSy-EBMs with specialized architectures that ensure principled and efficient gradient computation. Such specialized architectures are less likely to support more complex modeling paradigms such as DSPar and DSPot. However, provided a NeSy-EBM with such an architecture, gradient descent techniques can be used in all motivating applications cited in Section 1.1.

4.3.3 Bilevel Value-Function Optimization

As shown in Section 4.2.3, minimizer gradients are relatively more computationally expensive to compute and require more assumptions than value-function gradients. In this subsection, I introduce a technique for optimizing a minimizer-based loss with only first-order gradients. This technique is built on the fact that the general definition of NeSy-EBM learning (4.5) is naturally formulated as bilevel optimization. In other words, the NeSy learning objective is a function of variable values obtained

by solving a lower-level inference problem that is symbolic reasoning:

$$\begin{aligned}\underset{\substack{(\mathbf{w}_{sy},\mathbf{w}_{nn})\in\mathcal{W}_{sy}\times\mathcal{W}_{nn}\\(\hat{\mathbf{y}}^1,\cdots,\hat{\mathbf{y}}^P)\in\mathcal{Y}_1\times\cdots\times\mathcal{Y}_P}}{\arg\min}\ \frac{1}{P}\sum_{i=1}^{P}\Bigg(&L_{NN}(\mathbf{g}_{nn}(\mathbf{x}^i_{nn},\mathbf{w}_{nn}),\mathbf{t}^i_{nn})+L_{Val}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i)\\&+d(\hat{\mathbf{y}}^i,\mathbf{t}^i_{\mathcal{Y}})\Bigg)+\mathcal{R}(\mathbf{w}_{sy},\mathbf{w}_{nn})\end{aligned}\tag{4.24}$$

$$\text{s.t.}\qquad \hat{\mathbf{y}}^i\in\underset{\tilde{\mathbf{y}}\in\mathcal{Y}}{\arg\min}\,E(\tilde{\mathbf{y}},\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn}),\quad\forall i\in\{1,\cdots,P\}.$$

Regardless of the continuity and curvature properties of the upper and lower level objectives, (4.24) is equivalent to the following:

$$\begin{aligned}\underset{\substack{(\mathbf{w}_{sy},\mathbf{w}_{nn})\in\mathcal{W}_{sy}\times\mathcal{W}_{nn}\\(\hat{\mathbf{y}}^1,\cdots,\hat{\mathbf{y}}^P)\in\mathcal{Y}_1\times\cdots\times\mathcal{Y}_P}}{\arg\min}\ \frac{1}{P}\sum_{i=1}^{P}\Bigg(&L_{NN}(\mathbf{g}_{nn}(\mathbf{x}^i_{nn},\mathbf{w}_{nn}),\mathbf{t}^i_{nn})+L_{Val}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i)\\&+d(\hat{\mathbf{y}}^i,\mathbf{t}^i_{\mathcal{Y}})\Bigg)+\mathcal{R}(\mathbf{w}_{sy},\mathbf{w}_{nn})\end{aligned}\tag{4.25}$$

$$\text{s.t.}\qquad E(\hat{\mathbf{y}}^i,\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})-V_{\mathcal{Y}}(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i)\leq 0,\quad\forall i\in\{1,\cdots,P\}.$$

The formulation in (4.25) is referred to as a *value-function* approach in bilevel optimization literature [V. Outrata, 1990, Liu et al., 2021, 2022, Sow et al., 2022, Kwon et al., 2023]. Value-function approaches view the bilevel program as a single-level constrained optimization problem by leveraging the value-function as a tight lower bound on the lower-level objective.

The inequality constraints in (4.25) do not satisfy any of the standard *constraint qualifications* that ensure the feasible set near the optimal point is similar to its linearized approximation [Nocedal and Wright, 2006]. This raises a challenge for providing theoretical convergence guarantees for constrained optimization techniques. Following a recent line of value-function approaches to bilevel programming [Liu et al., 2021, Sow et al., 2022, Liu et al., 2023], I overcome this challenge by allowing at most an $\iota > 0$ violation in each constraint in (4.25). With this relaxation, strictly feasible points exist and, for instance, the linear independence constraint qualification (LICQ) can hold.

Another challenge that arises from (4.25) is that the energy function of NeSy-

EBMs is typically non-smooth with respect to the targets and even infinite-valued to represent constraints implicitly. As a result, penalty or augmented Lagrangian functions derived from (4.25) are intractable. Therefore, I substitute each instance of the energy function evaluated at the training sample $\mathcal{S}_i$, where $i \in \{1, \cdots, P\}$, and with weights $\mathbf{w}_{sy}$ and $\mathbf{w}_{nn}$ in the constraints of (4.25) with the following function:

$$\begin{aligned} M(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \rho) &:= \inf_{\tilde{\mathbf{y}} \in \mathcal{Y}} \left(E(\tilde{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + \frac{1}{2\rho} \|\tilde{\mathbf{y}} - \hat{\mathbf{y}}^i\|_2^2 \right), \quad (4.26) \\ &= V_{conv}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i; \hat{\mathbf{y}}^i, \frac{1}{2\rho}) \end{aligned}$$

where ρ is a positive scalar. For convex E, (4.26) is the Moreau envelope of the energy function [Rockafellar, 1970, Parikh and Boyd, 2013]. In general, even for non-convex energy functions, M is finite for all $\mathbf{y} \in \mathcal{Y}$ and it preserves global minimizers and minimum values, i.e.,

$$\mathbf{y}^*(\mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \arg\min_{\hat{\mathbf{y}}^i \in \mathcal{Y}} M(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \rho), \quad (4.27)$$

$$V_{\mathcal{Y}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) = \min_{\hat{\mathbf{y}}^i \in \mathcal{Y}} M(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \rho). \quad (4.28)$$

When the energy function is a lower semi-continuous convex function, its Moreau envelope is convex, finite, and continuously differentiable, and its gradient with respect to $\hat{\mathbf{y}}^i$ is:

$$\nabla_{\hat{\mathbf{y}}^i} M(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \rho) = \frac{1}{\rho} \left(\hat{\mathbf{y}}^i - \arg\min_{\tilde{\mathbf{y}} \in \mathcal{Y}} \left(\rho E(\tilde{\mathbf{y}}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + \frac{1}{2} \|\tilde{\mathbf{y}} - \hat{\mathbf{y}}^i\|_2^2 \right) \right). \quad (4.29)$$

Convexity is a sufficient but not necessary condition to ensure M is differentiable with respect to $\hat{\mathbf{y}}^i$. See Bonnans and Shapiro [2000] for results regarding the sensitivity of optimal value-functions to perturbations. Further, as M is a value-function, gradients of M with respect to weights are derived using Theorem 6.

I propose the following relaxed and smoothed value-function approach to finding

an approximate solution of (4.24):

$$\underset{\substack{(\mathbf{w}_{sy},\mathbf{w}_{nn})\in\mathcal{W}_{sy}\times\mathcal{W}_{nn}\\(\hat{\mathbf{y}}^1,\cdots,\hat{\mathbf{y}}^P)\in\mathcal{Y}_1\times\cdots\times\mathcal{Y}_P}}{\arg\min} \frac{1}{P}\sum_{i=1}^{P}\Bigg(L_{NN}(\mathbf{g}_{nn}(\mathbf{x}^i_{nn},\mathbf{w}_{nn}),\mathbf{t}^i_{nn}) + L_{Val}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy},\mathbf{w}_{nn}),\mathcal{S}_i) + d(\hat{\mathbf{y}}^i,\mathbf{t}^i_{\mathcal{Y}})\Bigg) + \mathcal{R}(\mathbf{w}_{sy},\mathbf{w}_{nn}) \tag{4.30}$$

$$\text{s.t.} \qquad M(\hat{\mathbf{y}}^i,\mathcal{S}_i,\mathbf{w}_{sy},\mathbf{w}_{nn};\rho) - V_{\mathcal{Y}}(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i) \leq \iota, \quad \forall i \in \{1,\cdots,P\},$$

The formulation (4.30) is the core of my proposed NeSy-EBM learning framework outlined in Algorithm 4 below. The algorithm proceeds by approximately solving instances of (4.30) in a sequence defined by a decreasing ι. This is a graduated approach to solving (4.25) with instances of (4.30) that are increasingly tighter approximations.

Algorithm 4 Bilevel Value-Function Optimization for NeSy-EBM Learning

Require: Moreau Param.: ρ, Starting weights: $(\mathbf{w}_{sy},\mathbf{w}_{nn}) \in \mathcal{W}_{sy}\times\mathcal{W}_{nn}$

1: $\hat{\mathbf{y}}^i \leftarrow (\mathbf{t}^i_{\mathcal{Y}}, \arg\min_{\hat{\mathbf{z}}\in\mathcal{Z}^i_{\mathcal{Y}}} E((\mathbf{t}^i_{\mathcal{Y}},\hat{\mathbf{z}}),\mathbf{x}^i_{sy},\mathbf{x}^i_{nn},\mathbf{w}_{sy},\mathbf{w}_{nn})), \quad \forall i = 1,\cdots,P$

2: $\iota \leftarrow \max_{i\in\{1,\cdots,P\}} M(\hat{\mathbf{y}}^i,\mathcal{S}_i,\mathbf{w}_{sy},\mathbf{w}_{nn};\rho) - V_{\mathcal{Y}}(\mathbf{w}_{sy},\mathbf{w}_{nn},\mathcal{S}_i)$

3: **for** $t = 0,1,2,\cdots$ **do**

4: Find $\mathbf{w}_{sy},\mathbf{w}_{nn},\mathbf{y}^1,\cdots,\mathbf{y}^P$ minimizing (4.30) with ι

5: **if** Stopping criterion satisified **then**

6: Stop with: $\mathbf{w}_{sy},\mathbf{w}_{nn},\mathbf{y}^1,\cdots,\mathbf{y}^P$

7: $\iota \leftarrow \frac{1}{2}\cdot\iota$

I suggest starting points for each $\hat{\mathbf{y}}^i$ to be the corresponding latent inference minimizer and ι to be the maximum difference in the value-function and the smooth energy function. At this suggested starting point, the supervised loss is initially 0, and the subproblem reduces to minimizing the learning objective without increasing the most violated constraint. Then, the value for ι is halved every time an approximate solution to the subproblem, (4.30), is reached. The outer loop of the NeSy-EBM learning framework may be stopped by either watching the progress of a training or validation evaluation metric or by specifying a final value for ι.

Each instance of (4.30) in Algorithm 4 can be optimized using only first-order gradient-based methods. Specifically, for my empirical analysis I employ the bound-constrained augmented Lagrangian algorithm, Algorithm 17.4 from Nocedal and

Wright (2006). The augmented Lagrangian algorithm finds approximate minimizers of the problem's augmented Lagrangian for a fixed setting of the penalty parameters using gradient descent. To simplify notation, let the constraints in (4.30) be denoted by:

$$c(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota) := M(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \rho) - V_{\mathcal{Y}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) - \iota, \tag{4.31}$$

for each constraint indexed $i \in \{1, \cdots, P\}$. Moreover, let

$$\mathbf{c}(\mathbf{y}^1, \cdots, \mathbf{y}^P, \mathcal{S}, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota) := [c(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota)]_{i=1}^P. \tag{4.32}$$

The augmented Lagrangian function corresponding to (4.30) introduces a quadratic penalty parameter μ and P linear penalty parameters $\lambda := [\lambda_i]_{i=1}^P$, as follows:

$$\begin{aligned}
&\mathcal{L}_A(\hat{\mathbf{y}}^1, \cdots, \hat{\mathbf{y}}^P, \mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}, \mathbf{s}; \lambda, \mu, \iota) \qquad (4.33)\\
&:= \frac{1}{P} \sum_{i=1}^{P} \left(L_{NN}(\mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{t}_{nn}^i) + L_{Val}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) + d(\hat{\mathbf{y}}^i, \mathbf{t}_{\mathcal{Y}}^i) \right)\\
&\quad + \frac{\mu}{2} \sum_{i=1}^{P} \left(c(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota) + s_i \right)^2 + \sum_{i=1}^{P} \lambda_i \left(c(\hat{\mathbf{y}}^i, \mathcal{S}_i, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota) + s_i \right) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}),
\end{aligned}$$

where I introduced P slack variables, $\mathbf{s} = [s_i]_{i=1}^P$, for each inequality constraint. The bound-constrained augmented Lagrangian algorithm provides a principled method for updating the penalty parameters and ensures fundamental convergence properties of my learning framework. Notably, the limit points of the iterate sequence are stationary points of $\|\mathbf{c}(\mathbf{y}^1, \cdots, \mathbf{y}^P, \mathcal{S}, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \iota) + \mathbf{s}\|^2$ when the problem has no feasible points. When the problem is feasible, and LICQ holds at the limits, they are KKT points of (4.30) (Theorem 17.2 in Nocedal and Wright [2006]). Convergence rates and stronger guarantees are possible by analyzing the structure of the energy function for specific NeSy-EBMs.

The bilevel value-function optimization technique in Algorithm 4 is an end-to-end algorithm for minimizing a general NeSy-EBM learning loss with only first-order value-function gradients. Thus, Algorithm 4 is a more practical and widely applicable technique for NeSy-EBM learning than modular and direct gradient descent methods.

Specifically, the bilevel approach can be employed for a broader class of NeSy-EBMs than direct gradient descent methods and for every motivating application. Moreover, I demonstrate that it can be used to train DSVar and DSPot NeSy-EBMs in my empirical analysis.

4.3.4 Stochastic Policy Optimization

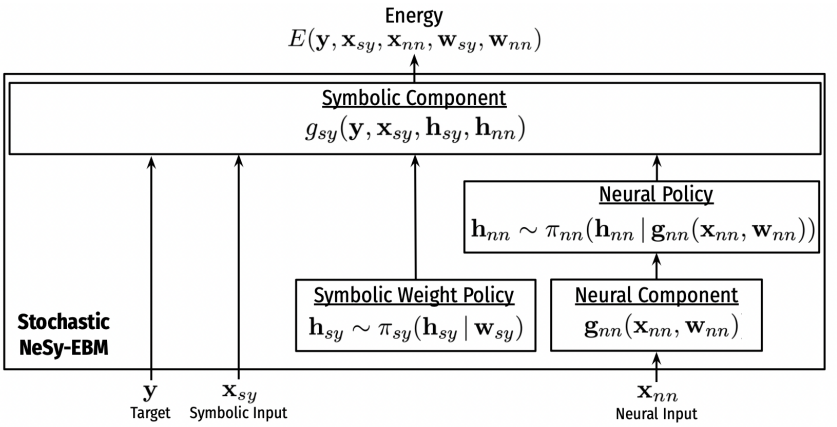


Figure 4.2: A stochastic NeSy-EBM. The symbolic weights and the neural component parameterize stochastic policies. A sample from the policies is drawn to produce arguments of the symbolic component.

Finally, another approach to NeSy-EBM learning that avoids directly computing the energy minimizer's gradients with respect to the weights is to reformulate NeSy learning as stochastic policy optimization. Fig. 4.2 shows the modifications to the standard NeSy-EBM framework to create a stochastic NeSy-EBM. The symbolic and neural weights are used to condition a symbolic weight and neural policy, denoted by π_{sy} and π_{nn}, respectively. Samples from the policies replace the symbolic weights and neural output as arguments of the symbolic component. Specifically, given symbolic and neural weights $\mathbf{w}_{sy}$ and $\mathbf{w}_{nn}$ and input features $\mathbf{x}_{nn}^i$ from a training sample $\mathcal{S}_i \in \mathcal{S}$, $\mathbf{h}_{sy}$ and $\mathbf{h}_{nn}^i$ are random variables with the following conditional distributions:

$$\mathbf{h}_{sy} \sim \pi_{sy}(\mathbf{h}_{sy} \mid \mathbf{w}_{sy}), \tag{4.34}$$

$$\mathbf{h}_{nn}^i \sim \pi_{nn}(\mathbf{h}_{nn}^i \mid \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn})). \tag{4.35}$$

Moreover, the random variables $\mathbf{h}_{sy}$ and $\mathbf{h}^i_{nn}$ are modeled independently, thus the conditional joint distribution, denoted by π, is:

$$\pi(\mathbf{h}_{sy}, \mathbf{h}^i_{nn} \,|\, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) := \pi_{sy}(\mathbf{h}_{sy} \,|\, \mathbf{w}_{sy}) \cdot \pi_{nn}(\mathbf{h}^i_{nn} \,|\, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) \quad (4.36)$$

The stochastic NeSy-EBM energy is the symbolic component evaluated at a sample from the joint distribution above:

$$E(\mathbf{y}, \mathbf{x}^i_{sy}, \mathbf{x}^i_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) := g_{sy}(\mathbf{y}, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}) \quad (4.37)$$

The NeSy-EBM energy and all of the NeSy-EBM per-sample loss functionals discussed in Section 4.2 are, therefore, random variables with distributions that are defined by π. Under the stochastic policy optimization framework, loss functionals are denoted by the function J^i for each $i \in \{1, \cdots, P\}$ such that:

$$J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) := L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \quad (4.38)$$

Learning is minimizing the expected value of the stochastic loss functional and is formulated as:

$$\underset{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}}{\arg\min} \frac{1}{P} \sum_{i=1}^{P} \mathbb{E}_\pi \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \right] + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}), \quad (4.39)$$

where $\mathbb{E}_\pi$ is the expectation over the joint distribution π.

I apply gradient-based learning algorithms to find an approximate solution to (4.39). The policy gradient theorem [Williams, 1992, Sutton et al., 1999, Sutton and Barto, 2018] yields the following expression for the gradients of the expected value of

a loss functional:

$$\nabla_{\mathbf{w}_{nn}} \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \right] \tag{4.40}$$
$$= \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \cdot \nabla_{\mathbf{w}_{nn}} \log \pi(\mathbf{h}_{sy}, \mathbf{h}^i_{nn} \,|\, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) \right].$$
$$\nabla_{\mathbf{w}_{sy}} \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \right] \tag{4.41}$$
$$= \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \cdot \nabla_{\mathbf{w}_{sy}} \log \pi(\mathbf{h}_{sy}, \mathbf{h}^i_{nn} \,|\, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) \right].$$

The expression for the gradient of the expected loss functional above motivates a family of gradient estimators. Notably, the REINFORCE gradient estimator for NeSy-EBM learning is:

$$\nabla_{\mathbf{w}_{nn}} \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \right] \tag{4.42}$$
$$\approx \frac{1}{N} \sum_{k=1}^{N} \left(J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}^{(k)}_{sy}, \mathbf{h}^{i(k)}_{nn}), \mathcal{S}_i) \nabla_{\mathbf{w}_{nn}} \log \pi(\mathbf{h}^{(k)}_{sy}, \mathbf{h}^{i(k)}_{nn} \,|\, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) \right),$$
$$\nabla_{\mathbf{w}_{sy}} \mathbb{E}_{\pi} \left[J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}_{sy}, \mathbf{h}^i_{nn}), \mathcal{S}_i) \right] \tag{4.43}$$
$$\approx \frac{1}{N} \sum_{k=1}^{N} \left(J^i(g_{sy}(\cdot, \mathbf{x}^i_{sy}, \mathbf{h}^{(k)}_{sy}, \mathbf{h}^{i(k)}_{nn}), \mathcal{S}_i) \nabla_{\mathbf{w}_{sy}} \log \pi(\mathbf{h}^{(k)}_{sy}, \mathbf{h}^{i(k)}_{nn} \,|\, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}^i_{nn}, \mathbf{w}_{nn})) \right),$$

where each $\mathbf{h}^{(k)}_{sy}$ and $\mathbf{h}^{i(k)}_{nn}$ for $k \in \{1, \cdots, N\}$ is an independent sample of the random variables.

Stochastic policy optimization techniques are broadly applicable for end-to-end training of NeSy-EBMs because they are agnostic to the neural-symbolic interface and the symbolic inference process. Moreover, they can be used for every motivating application and modeling paradigm. The tradeoff with the stochastic policy approach, however, is the high variance in the sample estimates for the policy gradient. This is a common challenge in policy optimization that becomes more prominent with increasing dimensionality of the policy output space [Sutton and Barto, 2018]. Thus, learning with a stochastic policy optimization approach may take significantly more iterations to converge compared to the other presented techniques.

Chapter 5

Neural Probabilistic Soft Logic and Deep Hinge-Loss Markov Random Fields

Chapters 4-6 covered a general mathematical framework, modeling paradigms, and learning algorithms for NeSy EBMs. Here, I introduce *Neural Probabilistic Soft Logic* (NeuPSL), an expressive framework for constructing a broad class of NeSy-EBMs by extending the probabilistic soft logic (PSL) probabilistic programming language [Bach et al., 2017]. NeuPSL is designed to be expressive and efficient to support every modeling paradigm and easily be used for a range of applications. I begin by presenting the essential syntax and semantics of NeuPSL, encompassing Deep Hinge-Loss Markov Random Fields (deep HL-MRF), the underlying probabilistic graphical model (see Bach et al. [2017] for an in-depth introduction to PSL syntax and semantics). Then, I present a new formulation and regularization of (Neu)PSL inference as a constrained quadratic program. This formulation is utilized to guarantee differentiability properties and provide principled gradients to support end-to-end neural and symbolic parameter learning. Moreover, I introduce a new deep HL-MRF inference algorithm that leverages the new formulation to naturally produce statistics necessary for computing gradients for learning and fully leverage warm-starts to improve learning runtime.

5.1 Neural Probabilistic Soft Logic

NeuPSL is a declarative language used to construct NeSy-EBMs. Intuitively, NeuPSL provides a syntax for encoding dependencies between relations and attributes of entities and for integrating neural components in a symbolic model. Specifically, dependencies and neural component compositions are expressed as first-order logical or arithmetic statements referred to as *rules*. Each rule is a template for instantiating, i.e., *grounding*, potentials or constraints to define the NeuPSL energy function. Every rule is grounded over a set of domains, $\mathbf{D} = \{D_1, D_2, \cdots\}$, where each of the domains D_i is a finite set of elements referred to as *constants*. For instance, referring to the visual Sudoku problem described in Example 2, the constant "A1" can denote the cell at position $A1$ in a Sudoku puzzle and the constant "1" can denote the digit 1. Constants are grouped and aligned with a corresponding domain from $\mathbf{D}$ using placeholders or *variables*. Relations between constants are *predicates*. In NeuPSL, a predicate is referenced using its unique identifier. For instance, CellDigit is a predicate that can represent whether a cell contains a specified digit. Another example is the predicate SudokuViolation representing whether a Sudoku rule is violated given the digits in two specified cells. Finally, the predicate NeuralClassifier is a predicate that represents the predicted digit in a cell made by a neural network classifier. Predicates with specified constant domains are *atoms*. NeuPSL extends PSL with *deep atoms*: atoms backed by a deep model.

Definition 7

Atom. An ***atom****, A, is a predicate associated with a list of $k > 0$ domains $D'_1, \cdots, D'_k$ from $\mathbf{D}$:*

$$A : (D'_1 \times \cdots \times D'_k) \to [0, 1]$$

where k is the corresponding predicate's arity.

A ***deep atom****, DA, with domains $D'_1, \cdots, D'_k$ from $\mathbf{D}$ is an atom parameterized*

by a set of weights $\mathbf{w}_{nn}$ *from a domain* $\mathcal{W}_{nn}$

$$DA : (D'_1 \times \cdots \times D'_k; \mathbf{w}_{nn}) \to [0, 1].$$

A ***ground atom*** *is an atom with constant arguments.* □

Building on the definition above, a NeuPSL rule is a symbolic relation between atoms.

Definition 8

Rule. A ***rule****, R, is a function of* $s \geq 1$ *variables* $v_1, \cdots, v_s$ *from the domains* $D'_1, \cdots, D'_s \in \mathbf{D}$*, respectively:*

$$R : (D'_1 \times \cdots \times D'_s) \to [0, 1]$$
$$v_1, \cdots, v_s \mapsto R(v_1, \cdots, v_s)$$

Moreover, a rule is a composition of $l \geq 1$ *atoms,* $A_1, \cdots A_l$*.*

All rules are either associated with a non-negative weight and a value $q \in \{1, 2\}$*, or are unweighted. The weight (or absence of) and value q of a rule determine the structure of the potentials the rule instantiates. A weighted rule is referred to as a* ***soft rule****, and an unweighted rule is referred to as a* ***hard rule****.*

A ***logical rule*** *is expressed as a logical implication of atoms.*

An ***arithmetic rule*** *is expressed as a linear inequality of atoms.*

A ***ground rule*** *is a rule with constant arguments, i.e., a rule with only ground atoms.* □

For instance, the following is an example of two rules for solving visual Sudoku with NeuPSL.

$$\begin{aligned} 1.0: \quad & \textsc{NeuralClassifier}(\texttt{Pos}, \texttt{Digit}) = \textsc{CellDigit}(\texttt{Pos}, \texttt{Digit}) \\ & \textsc{CellDigit}(\texttt{Pos1}, \texttt{Digit1}) \wedge \textsc{SudokuViolation}(\texttt{Pos1}, \texttt{Pos2}, \texttt{Digit1}, \texttt{Digit2}) \\ & \quad \to \neg \textsc{CellDigit}(\texttt{Pos2}, \texttt{Digit2}) \quad . \end{aligned}$$

The first rule in the example above is soft as it is weighted with weight 1.0. Moreover, the first rule is arithmetic and encodes a dependency between the digit label predicted by a neural classifier and the atom CellDigit(`Pos`, `Digit`), i.e., if the neural classifier predicts the digit `Digit` is in position `Pos`, then the `Digit` is in position `Pos`. The second rule is a hard logical rule that encodes the rules of Sudoku. Moreover, the rule the second rule can be read: if the digit `Digit1` is in position `Pos1` and the `Digit2` in `Pos2` causes a Sudoku rule violation, then `Digit2` is not in `Pos2`.

Rules are grounded by performing every distinct substitution of the variables in the atoms for constants in their respective domain. For example, every substitution for the `Pos` and `Digit` variable arguments from the domains of non-empty Sudoku puzzle cell positions, $A1, \cdots, I9$, and digits $1, \cdots, 9$ is realized to ground the first rule:

$$
\begin{aligned}
1.0: \quad & \textsc{NeuralClassifier}(\text{“}A1\text{”}, \text{“1”}) = \textsc{CellDigit}(\text{“}A1\text{”}, \text{“1”}) \\
& \vdots \\
1.0: \quad & \textsc{NeuralClassifier}(\text{“19”}, \text{“9”}) = \textsc{CellDigit}(\text{“}I9\text{”}, \text{“9”})
\end{aligned}
$$

Similarly, every substitution for the `Pos1`, `Pos2`, `Digi1`, and `Digit2` variable arguments from the domains of all Sudoku puzzle cell positions, $A1, \cdots, I9$, and digits $1, \cdots, 9$ is realized to ground the second rule:

$$
\begin{aligned}
& \textsc{CellDigit}(\text{“}A1\text{”}, \text{“1”}) \wedge \textsc{SudokuViolation}(\text{“}A1\text{”}, \text{“}A2\text{”}, \text{“1”}, \text{“1”}) \\
& \quad \rightarrow \neg \textsc{CellDigit}(\text{“}A2\text{”}, \text{“1”}) \quad . \\
& \vdots \\
& \textsc{CellDigit}(\text{“}I9\text{”}, \text{“9”}) \wedge \textsc{SudokuViolation}(\text{“}I9\text{”}, \text{“}I8\text{”}, \text{“9”}, \text{“9”}) \\
& \quad \rightarrow \neg \textsc{CellDigit}(\text{“}I8\text{”}, \text{“9”}) \quad .
\end{aligned}
$$

5.2 Deep-Hinge Loss Markov Random Fields

The rule instantiation process described in the previous section results in a set of ground atoms. Each ground atom is mapped to either an observed variable, $x_{sy,i}$, target variable, y_i, or a neural function with inputs $\mathbf{x}_{nn}$ and parameters $\mathbf{w}_{nn,i}$: $g_{nn,i}(\mathbf{x}_{nn}, \mathbf{w}_{nn,i})$. Specifically, all atoms instantiated from a deep atom are mapped to a neural function, and the observed and target atom partitions are pre-specified. Further, variables are aggregated into the vectors $\mathbf{x}_{sy} = [x_{sy_i}]_{i=1}^{n_x}$ and $\mathbf{y} = [y_i]_{i=1}^{n_y}$ and neural outputs are aggregated into the vector $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$.

The ground rules and variables are used to define linear inequalities in a standard form: $\ell(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0$, where ℓ is a linear function of its arguments. To achieve this, logical rules are first converted into disjunctive normal form. Then, the rules are translated into linear inequalities using an extended interpretation of the logical operators, namely Łukasiewicz logic [Klir and Yuan, 1995]. Similarly, arithmetic rules define one or more standard form inequalities that preserve the rules' dependencies via algebraic operations.

Linear inequalities instantiated from hard ground rules are constraints in NeuPSL. Further, linear inequalities instantiated from soft ground rules define *potential functions* of the form:

$$\phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := (\max\{\ell(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), 0\})^q. \tag{5.1}$$

Intuitively, the value of potential is the, possibly squared, level of dissatisfaction of the linear inequality created by the ground rule. Further, each potential is associated with the weight of its instantiating rule. Weight sharing among the potentials is formalized by defining a partitioning using the instantiating rules, i.e., every potential instantiated by the same rule belongs to the same partition and shares a weight. The potentials and weights from the instantiation process are used to define a tractable class of graphical models, which we refer to as *deep hinge-loss Markov random fields* (Deep HL-MRF):

Definition 9 (Deep Hinge-Loss Markov Random Field)

Let $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$ *be functions with corresponding weights* $\mathbf{w}_{nn} = [\mathbf{w}_{nn,i}]_{i=1}^{n_g}$ *and inputs* $\mathbf{x}_{nn}$ *such that* $g_{nn,i} : (\mathbf{w}_{nn,i}, \mathbf{x}_{nn}) \mapsto [0,1]$*. Let* $\mathbf{y} \in [0,1]^{n_y}$ *and* $\mathbf{x}_{sy} \in [0,1]^{n_x}$*. A* ***deep hinge-loss potential*** *is a function of the form:*

$$\phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := (\max\{\mathbf{a}_{\phi,\mathbf{y}}^T \mathbf{y} + \mathbf{a}_{\phi,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_\phi, 0\})^q \tag{5.2}$$

where $\mathbf{a}_{\phi,\mathbf{y}} \in \mathbb{R}^{n_y}$*,* $\mathbf{a}_{\phi,\mathbf{x}_{sy}} \in \mathbb{R}^{n_x}$*, and* $\mathbf{a}_{\phi,\mathbf{g}_{nn}} \in \mathbb{R}^{n_g}$ *are variable coefficient vectors,* $b_\phi \in \mathbb{R}$ *is a vector of constants, and* $q \in \{1, 2\}$*. Let* $\mathcal{T} = [\tau_i]_{i=1}^r$ *denote an ordered partition of a set of* m *deep hinge-loss potentials. Further, define*

$$\mathbf{\Phi}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \left[\sum_{k \in \tau_i} \phi_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))\right]_{i=1}^r. \tag{5.3}$$

Let $\mathbf{w}_{sy}$ *be a vector of* r *non-negative symbolic weights corresponding to the partition* $\mathcal{T}$*. Then, a* ***deep hinge-loss energy function*** *is:*

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) := \mathbf{w}_{sy}^T \mathbf{\Phi}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \tag{5.4}$$

Let $\mathbf{a}_{c_k,\mathbf{y}} \in \mathbb{R}^{n_y}$*,* $\mathbf{a}_{c_k,\mathbf{x}_{sy}} \in \mathbb{R}^{n_x}$*,* $\mathbf{a}_{c_k,\mathbf{g}_{nn}} \in \mathbb{R}^{n_g}$*, and* $b_{c_k} \in \mathbb{R}$ *for each* $k \in 1, \ldots, q$ *and* $q \geq 0$ *be vectors defining linear inequality constraints and a feasible set:*

$$\mathbf{\Omega}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) :=$$
$$\left\{\mathbf{y} \in [0,1]^{n_y} \mid \mathbf{a}_{c_k,\mathbf{y}}^T \mathbf{y} + \mathbf{a}_{c_k,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{c_k,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{c_k} \leq 0\,, \forall\, k = 1, \ldots, q\right\}.$$

Then a ***deep hinge-loss Markov random field*** *defines the conditional probability density:*

$$P(\mathbf{y}|\mathbf{x}_{sy}, \mathbf{x}_{nn}) := \begin{cases} \frac{\exp(-E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}))}{\int_{\hat{\mathbf{y}}} \exp(-E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}))d\hat{\mathbf{y}}} & \mathbf{y} \in \mathbf{\Omega}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ 0 & o.w. \end{cases} \tag{5.5}$$

□

NeuPSL models are NeSy-EBMs with an extended-value deep HL-MRF energy

function capturing the constraints that define the feasible set. In other words, the symbolic component of NeuPSL is infinity if the targets are outside of the deep HL-MRF feasible set, else it is equal to the deep HL-MRF energy function:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = \begin{cases} \mathbf{w}_{sy}^T \mathbf{\Phi}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \mathbf{y} \in \mathbf{\Omega}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ \infty & o.w. \end{cases} \tag{5.6}$$

Further, NeuPSL prediction is finding the MAP state of the deep HL-MRF conditional distribution. Note that in deep HL-MRFs, the partition function is constant over the target variables. Moreover, as the exponential function is monotonically increasing, prediction is equivalent to finding the minimizer of the negative log probability of the deep HL-MRF joint distribution. This reduces to minimizing the deep HL-MRF energy function constrained to the feasible set. Therefore, deep HL-MRF MAP inference is equivalent to minimizing the NeuPSL symbolic component in (5.6):

$$\begin{aligned} \arg\max_{\mathbf{y} \in \mathbb{R}^{n_\mathbf{y}}} P(\mathbf{y}|\mathbf{x}_{sy}, \mathbf{x}_{nn}) &\equiv \arg\min_{\mathbf{y} \in \mathbb{R}^{n_\mathbf{y}}} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) && (5.7) \\ &\equiv \arg\min_{\mathbf{y} \in \mathbb{R}^{n_\mathbf{y}}} \mathbf{w}_{sy}^T \mathbf{\Phi}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &\quad \text{s.t.} \quad \mathbf{y} \in \mathbf{\Omega}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) && (5.8) \end{aligned}$$

Deep HL-MRF potentials are non-smooth and convex. Thus, as Deep HL-MRF energy functions are non-negative weighted sums of the potentials, they are also non-smooth and convex. Moreover, Deep HL-MRFs feasible sets are, by definition, convex polyhedrons. Therefore, Deep HL-MRF inference, as defined above in (5.8), is a non-smooth convex linearly constrained program. A natural extension of the definition above that is often used in practice adds support for integer constraints on the target variables. This change is useful in discrete problems and for leveraging hard logic semantics. However, adding integer constraints breaks the convexity property of MAP inference. Nevertheless, for many problems of practical scale, global minimizers or high-quality approximations of the MAP inference problem with

integer constraints can be quickly found with modern solvers.

5.3 A Smooth Formulation of Deep HL-MRF Inference

In this section I introduce a primal and dual formulation of Deep HL-MRF MAP inference as a linearly constrained convex quadratic program (LCQP) (see Appendix B for details). The primal and dual LCQP formulation has theoretical and practical advantages. Theoretically, the new formulation will be utilized to prove continuity and curvature properties of the Deep HL-MRF energy minimizer and value-function. Practically, LCQP solvers (e.g. Gurobi [Gurobi Optimization, 2024]) can be employed to achieve highly efficient MAP inference. Moreover, features of modern solvers, including support for integer constraints, can be leveraged to improve predictions.

In summary, m slack variables with lower bounds and $2 \cdot n_{\mathbf{y}} + m$ linear constraints are defined to represent the target variable bounds and deep hinge-loss potentials. All $2 \cdot n_{\mathbf{y}} + m$ variable bounds, m potentials, and $q \geq 0$ constraints are collected into a $(2 \cdot n_{\mathbf{y}} + q + 2 \cdot m) \times (n_{\mathbf{y}} + m)$ dimensional matrix $\mathbf{A}$ and a vector of $(2 \cdot n_{\mathbf{y}} + q + 2 \cdot m)$ elements that is an affine function of the neural predictions and symbolic inputs $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. Moreover, the slack variables and a $(n_{\mathbf{y}} + m) \times (n_{\mathbf{y}} + m)$ positive semi-definite diagonal matrix, $\mathbf{D}(\mathbf{w}_{sy})$, and a $(n_{\mathbf{y}} + m)$ dimensional vector, $\mathbf{c}(\mathbf{w}_{sy})$, are created using the symbolic weights to define a quadratic objective. Further, the original target variables and the slack variables are gathered into a vector $\nu \in \mathbb{R}^{n_{\mathbf{y}}+m}$. Altogether, the regularized convex LCQP reformulation of Deep HL-MRF MAP inference is:

$$V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) := \tag{5.9}$$
$$\min_{\nu \in \mathbb{R}^{n_{\mathbf{y}}+m}} \nu^T (\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I}) \nu + \mathbf{c}(\mathbf{w}_{sy})^T \nu \quad \text{s.t.} \quad \mathbf{A}\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0,$$

where $\epsilon \geq 0$ is a scalar regularization parameter added to the diagonal of $\mathbf{D}$ to ensure strong convexity. The function $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ in (5.9) is the optimal value-function of the LCQP formulation of NeuPSL inference referred to in the previous chapter.

By Slater's constraint qualification, strong duality holds when there is a feasible solution to (5.9) Boyd and Vandenberghe [2004]. In this case, an optimal solution to the dual problem yields an optimal solution to the primal problem. The Lagrange dual problem of (5.9) is:

$$\min_{\mu \in \mathbb{R}^{2\cdot(n_{\mathbf{y}}+m)+q}_{\geq 0}} \quad h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \tag{5.10}$$
$$:= \frac{1}{4}\mu^T \mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{A}^T\mu + \frac{1}{2}(\mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{c}(\mathbf{w}_{sy}) - 2\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))^T\mu,$$

where μ is the vector of dual variables and $h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is the LCQP dual objective function. As $(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})$ is diagonal, it is easy to invert, and thus it is practical to work in the dual space and map dual to primal variables. The dual-to-primal variable mapping is:

$$\nu \leftarrow -\frac{1}{2}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}(\mathbf{A}^T\mu + \mathbf{c}(\mathbf{w}_{sy})). \tag{5.11}$$

On the other hand, the primal-to-dual mapping is more computationally expensive and requires calculating a pseudo-inverse of the constraint matrix $\mathbf{A}$.

I use the LCQP formulation in (5.9) to establish continuity and curvature properties of the NeuPSL energy minimizer and the optimal value-function provided in the following theorem:

Theorem 10

Suppose for any setting of $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$ *there is a feasible solution to NeuPSL inference* (5.9). *Further, suppose* $\epsilon > 0$, $\mathbf{w}_{sy} \in \mathbb{R}^r_+$, *and* $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$. *Then:*

- *The minimizer of* (5.9), $\mathbf{y}^*(\mathbf{w}_{sy}, \mathbf{w}_{nn})$, *is a* $O(1/\epsilon)$ *Lipschitz continuous function of* $\mathbf{w}_{sy}$.
- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$, *is concave over* $\mathbf{w}_{sy}$.
- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is convex over* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$.
- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is differentiable with respect to* $\mathbf{w}_{sy}$. *Moreover,*

$$\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) = \mathbf{\Phi}(\mathbf{y}^*(\mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})).$$

Furthermore, $\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is Lipschitz continuous over* $\mathbf{w}_{sy}$.

- *If there is a feasible point* ν *strictly satisfying the* $i'th$ *inequality constraint of* (5.9)*, i.e.,* $\mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] < 0$*, then* $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is subdifferentiable with respect to the* $i'th$ *constraint constant* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i]$*. Moreover,*

$$\partial_{\mathbf{b}[i]} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) = \{\mu^*[i] \,|\, \mu^* \in \underset{\mu \in \mathbb{R}^{2\cdot(n_{\mathbf{y}}+m)+q}_{\geq 0}}{\arg\min} h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))\}.$$

Furthermore, if $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ *is a smooth function of* $\mathbf{w}_{nn}$*, then* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ *is a smooth function of* $\mathbf{w}_{nn}$*. Additionally, the set of regular subgradients of* $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is:*

$$\begin{aligned} &\hat{\partial}_{\mathbf{w}_{nn}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\ &\quad \supset \nabla_{\mathbf{w}_{nn}} \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))^T \partial_{\mathbf{b}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))). \end{aligned} \tag{5.12}$$

Proof. See Appendix B.2. □

Theorem 10 establishes the continuity properties of the NeuPSL value-function, complementing the result in Theorem 6. Further, it provides a simple explicit form of the value-function gradient with respect to the symbolic weights and regular subgradient with respect to the neural weights (equivalent to those suggested by Theorem 6). Thus, Theorem 10 supports the principled application of the end-to-end learning algorithms presented in Section 4.3 for training both the symbolic and neural weights of a NeuPSL model.

5.4 Dual block coordinate descent

In this section, I introduce a block coordinate descent (BCD) [Wright, 2015] algorithm for working directly with the dual LCQP formulation of inference in (5.10). The dual BCD algorithm is the first method specialized for the regularized NeuPSL dual LCQP inference formulation. It is, therefore, also the first to produce optimal dual

variables that directly yield both optimal primal variables and principled gradients for learning.

At a high-level, the dual BCD algorithm proceeds by successively minimizing the objective along the subgradient of a block of dual variables. For this reason, dual BCD guarantees descent at every iteration, partially explaining its effectiveness at leveraging warm-starts and improving learning runtimes. The algorithm is stopped when the primal-dual gap drops below a threshold $\delta > 0$.

In this section, I omit the symbolic and neural weights from the function arguments to simplify notation. Further, define U_i, $i = 1, 2, \ldots, p$ to be a cover of the dual variable components $\{1, 2, \ldots, n_{\mathbf{y}} + m + q\}$. In practice, blocks are defined as a single dual variable corresponding to a constraint from the feasible set or a deep hinge-loss function, along with the dual variables corresponding to the bounds of the primal variables in the constraint or hinge-loss.

I work with a slightly more general objective,

$$h(\mu) := \frac{1}{2}\mu^T \mathbf{A}\tilde{\mathbf{D}}\mathbf{A}^T\mu + \tilde{\mathbf{c}}^T\mu, \tag{5.13}$$

from which one can recover (B.12) by replacing $\tilde{\mathbf{D}} \leftarrow (\mathbf{D} + \epsilon\mathbf{I})^{-1}$ and $\tilde{\mathbf{c}} \leftarrow \mathbf{A}(\mathbf{D} + \epsilon\mathbf{I})^{-1}\mathbf{c} - 2\mathbf{b}$.

I use the superscript $\cdot^{(l)}$ to denote values in the l-th iteration and subscript $\cdot_{[i]}$ for the values corresponding to the block U_i. The row submatrix of $\mathbf{A}$ that corresponds to block i is denote by $\mathbf{A}_{[i]}$.

At each iteration l, one block $i \in \{1, 2, \ldots, p\}$ is chosen at random and the subvector of $\nabla h(\mu^{[l]})$ that corresponds to this block is computed,

$$\mathbf{d}^{(l)}_{[i]} := \nabla_{[i]} h(\mu^{(l)}) = (\mathbf{A}\tilde{\mathbf{D}}\mathbf{A}^T\mu^{(l)} + \tilde{\mathbf{c}})_{[i]}. \tag{5.14}$$

Defining $\mathbf{d}^{(l)}$ to be the vector in $\mathbb{R}^N$ whose ith block is $\mathbf{d}^{(l)}_{[i]}$ with zeros elsewhere, I

perform a line search along the negative of this direction. Note that

$$h(\mu^{(l)} - \alpha \mathbf{d}^{(l)}) = \frac{1}{2}\alpha^2 \mathbf{d}^{(l)T} \mathbf{A}\tilde{\mathbf{D}}\mathbf{A}^T \mathbf{d}^{(l)} - \alpha \mathbf{d}^{(l)T}(\mathbf{A}\tilde{\mathbf{D}}\mathbf{A}^T \mu^{(l)} + \tilde{\mathbf{c}}) + \textbf{constant} \quad (5.15)$$

$$= \frac{1}{2}\alpha^2 \mathbf{d}_{[i]}^{(l)T} \mathbf{A}_{[i]}\tilde{\mathbf{D}}\mathbf{A}_{[i]}^T \mathbf{d}_{[i]}^{(l)} - \alpha \mathbf{d}_{[i]}^{(l)T} \mathbf{d}_{[i]}^{(l)} + \textbf{constant}. \quad (5.16)$$

The unconstrained minimizer of this expression is

$$\alpha_l^* = \frac{\mathbf{d}_{[i]}^{(l)T} \mathbf{d}_{[i]}^{(l)}}{\mathbf{d}_{[i]}^{(l)T} \mathbf{A}_{[i]}\tilde{\mathbf{D}}\mathbf{A}_{[i]}^T \mathbf{d}_{[i]}^{(l)}}. \quad (5.17)$$

Given the nonnegativity constraints, I also need to ensure that $\mu_{[i]}^{(l)} - \alpha \mathbf{d}_{[i]}^{(l)} \geq 0$. Therefore, my choice of steplength is

$$\alpha_l = \min \left\{ \alpha_l^*, \min_{j \in U_i : \mathbf{d}_j^{(l)} > 0} \frac{\mu_j^{(l)}}{\mathbf{d}_j^{(l)}} \right\}. \quad (5.18)$$

To save some computation, I introduce intermediate variables $\mathbf{f}^{(l)} := \mathbf{A}^T \mathbf{d}^{(l)} = \mathbf{A}_{[i]}^T \mathbf{d}_{[i]}^{(l)}$, and $\mathbf{m}^{(l)} := \mathbf{A}^T \mu^{(l)}$. With the intermediate variables, the updates of the BCD algorithm are:

$$\mathbf{d}_{[i]}^{(l)} \leftarrow \mathbf{A}_{[i]}\tilde{\mathbf{D}}\mathbf{m}^{(l)} + \tilde{\mathbf{c}}_{[i]},\ \mathbf{f}^{(l)} \leftarrow \mathbf{A}_{[i]}^T \mathbf{d}_{[i]}^{(l)} \quad (5.19)$$

$$\mathbf{m}^{(l+1)} \leftarrow \mathbf{A}^T(\mu^{(l)} - \alpha_l \mathbf{d}^{(l)}) = \mathbf{m}^{(l)} - \alpha_l \mathbf{f}^{(l)}. \quad (5.20)$$

With the steplength suggested by (5.18), descent is guaranteed at each iteration. This property is partially why the dual BCD algorithm is effective at leveraging warmstarts which is valuable for improving the runtime of learning algorithms, as is demonstrated in Section 6.3.1.

As strong duality holds for the LCQP formulation of deep HL-MRF inference, stopping when the primal-dual gap is below a given threshold $\delta > 0$, is a principled stopping criterion. Formally, at any iteration Algorithm 5 applied to (B.11), I recover an estimate of the primal variable $\mathbf{v}$ from (B.13) and terminate when the gap between the primal and the dual objective falls below δ. The stopping criterion is checked

Algorithm 5 Dual LCQP Block Coordinate Descent

Set $l = 0$ and compute an initial feasible point $\mu^{(0)}$;
Compute $\mathbf{m}^{(0)} = \mathbf{A}^T \mu^{(0)}$;
while Stopping Criterion Not Satisfied **do**
  $S_k \leftarrow \text{Permutation}([1, 2, \ldots, p])$;
  for all $i \in S_k$ (in order) **do**
    Compute $\mathbf{d}_{[i]}^{(l)} \leftarrow \mathbf{A}_{[i]} \tilde{\mathbf{D}} \mathbf{m}^{(l)} + \tilde{\mathbf{c}}_{[i]}$; $\mathbf{f}^{(l)} \leftarrow \mathbf{A}_{[i]}^T \mathbf{d}_{[i]}^{(l)}$;
    Compute $\alpha_l \leftarrow \min \left\{ \frac{\mathbf{d}_{[i]}^{(l)T} \mathbf{d}_{[i]}^{(l)}}{\mathbf{f}^{(l)T} \tilde{\mathbf{D}} \mathbf{f}^{(l)}}, \min_{j \in U_i : \mathbf{d}_j^{(l)} > 0} \frac{\mu_j^{(l)}}{\mathbf{d}_j^{(l)}} \right\}$
    $\mu_{[i]}^{(l+1)} \leftarrow \mu_{[i]}^{(l)} - \alpha_l \mathbf{d}_{[i]}^{(l)}$; $\mu_{[j]}^{(l+1)} \leftarrow \mu_{[j]}^{(l)}$ for all $j \neq i$;
    $\mathbf{m}^{(l+1)} \leftarrow \mathbf{m}^{(l)} - \alpha_l \mathbf{f}^{(l)}$;
    $l \leftarrow l + 1$;
  $k \leftarrow k + 1$;

after every permutation block has been completely iterated over.

Connected Component Parallel D-BCD Oftentimes, the NeuPSL dual inference objective is additively separable over partitions of the variables. In this case, the dual BCD algorithm is parallelizable over the variable partitions. I propose an efficient method for identifying the separable components via the primal objective and constraints. More formally, prior to the primal problem instantiation, a disjoint-set data structure [Cormen et al., 2009] is initialized such that every primal variable belongs to a single unique disjoint set. Then, during instantiation, the disjoint-set data structure is maintained to preserve the property that two primal variables exist in the same set if and only if they occur together with a non-zero coefficient in a constraint or a potential. This is achieved by merging the sets of variables in every generated constraint or potential. This process is made extremely efficient with a path compression strategy implemented to optimize finding set representatives. This parallelization strategy is empirically studied in Chapter 6 where I refer to it as Connected Component Parallel D-BCD (CC D-BCD).

Lock Free Parallel D-BCD For sparse factor graphs with few connected components (e.g., a chain), the CC variant of D-BCD is ineffective as the updates cannot be distributed to maximize CPU utilization. One solution to overcome this issue and preserve the guaranteed descent property is to lock access and updates to dual

variables. In other words, processes checkout locks on the dual variables to access and update its value and corresponding statistics. Unfortunately, in practice there is too much overlap in the blocks for this form of synchronization to see runtime improvements. For this reason, I additionally propose a method that sacrifices the theoretical guaranteed descent property of the dual BCD algorithm for significant runtime improvements. My approach is inspired by lock free parallelization strategies in optimization literature [Bertsekas and N. Tsitsiklis, 1989, Recht et al., 2011, Liu et al., 2015]. Specifically, rather than having processes checkout locks on dual variables for the entire iteration, I only assume dual and intermediate variable updates are atomic. This assumption ensures the dual variables and intermediate variables are synchronized across processes. However, the steplength subproblem solution and the gradient may be incorrect. Despite this, in Section 6.2.1, I show this distributed variant of the dual BCD algorithm consistently finds a solution satisfying the stopping criterion and realizes significant runtime improvements over the CC D-BCD algorithm in some datasets.

5.5 Deep HL-MRF Learning

Deep HL-MRF learning follows the NeSy-EBM learning framework presented in Chapter 3. Additionally, symbolic parameters are constrained to lie on the unit simplex, $\Delta^r = \{\mathbf{w}_{sy} \in \mathbb{R}^r_+ \,|\, \|\mathbf{w}\|_1 = 1\}$, when the deep HL-MRF is exclusively used to obtain MAP inference predictions. This is because MAP inference in HL-MRFs is invariant to scalar multiplications of the weights, as stated in the following lemma.

Lemma 11

Consider a Deep-HL-MRF with r symbolic parameters $\mathbf{w}_{sy} \in \mathbb{R}^r_+$. For all symbolic weight configurations, $\mathbf{w}_{sy} \in \mathbb{R}^r_+$, and scalars, $\tilde{c} \in \mathbb{R}_+$, the solution of MAP inference with symbolic parameters $\mathbf{w}_{sy}$ and $\tilde{c} \cdot \mathbf{w}_{sy}$ are the same, i.e.,

$$\begin{aligned} &\operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \\ = &\operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \tilde{c} \cdot \mathbf{w}_{sy}, \mathbf{w}_{nn}). \end{aligned}$$

Proof. Scaling the symbolic parameters by a positive scalar $\tilde{c} \in \mathbb{R}_+$ is equivalent to a positive scaling of the MAP inference objective:

$$\begin{aligned}
&\operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \tilde{c} \cdot \mathbf{w}_{sy}, \mathbf{w}_{nn}) \\
&= \operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} \tilde{c} \cdot \mathbf{w}_{sy}^T \boldsymbol{\Phi}(\mathbf{y}, \mathbf{x_{sy}}, \mathbf{g}(\mathbf{x_{nn}}, \mathbf{w_{nn}}))) \\
&= \operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} \mathbf{w}_{sy}^T \boldsymbol{\Phi}(\mathbf{y}, \mathbf{x_{sy}}, \mathbf{g}(\mathbf{x_{nn}}, \mathbf{w_{nn}}))) \\
&= \operatorname*{arg\,min}_{\mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}).
\end{aligned} \tag{5.21}$$

□

The energy function $E(\cdot, \cdot, \cdot, \mathbf{w}_{sy} = \mathbf{0}, \mathbf{w}_{nn})$ cannot be captured by any symbolic parameter in the unit simplex. However, this is a benefit of adding the simplex constraint. The zero-weight configuration is a degenerate solution and should be avoided. Formally, $\mathbf{w}_{sy} = \mathbf{0}$ results in a *collapsed energy function*: a function that assigns all points to the same energy. Collapsed energy functions have no predictive power since finding the lowest energy state of the variables is trivial and uninformative.

With the simplex constraint on the symbolic parameters, deep HL-MRF parameter learning is:

$$\begin{aligned}
&\operatorname*{arg\,min}_{\mathbf{w}_{sy} \in \Delta^r, \mathbf{w}_{nn}} \mathcal{L}(\tilde{E}(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \\
&= \operatorname*{arg\,min}_{\mathbf{w}_{sy} \in \Delta^r, \mathbf{w}_{nn}} \frac{1}{P} \sum_{i=1}^{P} L^i(\tilde{E}(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}),
\end{aligned} \tag{5.22}$$

where $\tilde{E}$ is the extended value extension of the deep HL-MRF energy function capturing the constraints imposed by Ω:

$$\tilde{E}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) := \begin{cases} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) & \mathbf{y} \in \boldsymbol{\Omega}(\mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ \infty & \text{o.w.} \end{cases}.$$

5.5.1 Symbolic Parameter Regularizers

Generally, any convex loss function over the symbolic weights can be used to encourage solutions that are not at a corner or edge of the simplex. I suggest using the negative logarithm function as a symbolic weight regularizer:

$$\mathcal{R}_{Log}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) := -\sum_{i=1}^{r} \log_b(\mathbf{w}_{sy}[i]).$$

The negative logarithm function gives configurations with symbolic weights equal to 0 an infinite loss. A related regularization is the negative entropy function:

$$\mathcal{R}_{Entropy}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) := \sum_{i=1}^{r} \mathbf{w}_{sy}[i] \log_b(\mathbf{w}_{sy}[i]).$$

This regularization does not give infinite loss for configurations with symbolic weights equal to 0 like the negative logarithm function, However, it does bias the loss towards symbolic weights at the center of the simplex.

5.5.2 Gradient-Based Symbolic Weight Learning

See Wright and Recht (2022) for a complete introduction to gradient-based methods for constrained optimization. In this work, I consider the projected gradient descent and mirror descent algorithms for minimizing the learning loss over the symbolic weights of a deep HL-MRF. I give details on the updates for both algorithms in the following two sections.

When fitting only symbolic parameters, the gradient-based learning algorithms are stopped when the gradient of the objective with respect to the current iterate is nearly normal to the simplex. This stopping criterion is motivated by the following definition and theorem. The definition is the standard formal description of the normal cone of a set at a point. Then the theorem (see Wright and Recht (2022) for more details) states that at a local solution of the objective, the gradient of the objective is in the normal cone of the feasible set.

Definition 12

Let $\mathbf{\Omega} \subset \mathbb{R}^n$ *be a closed convex set. For a point* $\mathbf{x} \in \Omega$*, the* normal cone *is:*

$$N_\Omega(\mathbf{x}) = \{\mathbf{d} \in \mathbb{R}^n : \mathbf{d}^T(\mathbf{x}' - \mathbf{x}) \leq 0 \quad \forall \mathbf{x}' \in \Omega\}.$$

Theorem 13

Consider the problem

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}),$$

where $\mathbf{\Omega} \subset \mathbb{R}^n$ *is a closed convex set and* $\mathbf{f}$ *is continuously differentiable. If* $\mathbf{x}^* \in \Omega$ *is a local solution, then the gradient of the function at* $\mathbf{x}$ *is in the normal cone of the feasible set at* $\mathbf{x}$*:* $-\nabla f(\mathbf{x}^*) \in N_\Omega(\mathbf{x}^*)$*.*

The distance to the normal cone for both methods is measured using the KL-divergence between the gradient passed through a softmax function and the discrete r dimensional uniform distribution. Formally, when learning only symbolic parameters, the first-order learning methods presented in the following subsections are stopped when:

$$\mathrm{KL}(\mathrm{SoftMax}(\nabla_{\mathbf{w}_{sy}} \mathcal{L}(\tilde{E}(\cdot, \cdot, \cdot, \mathbf{w}_{sy}^{(k)}, \mathbf{w}_{nn}^{(k)}), S)) \| \mathrm{Unif}(\{1, \cdots, r\})) \leq \delta,$$

where $\delta > 0$ is a user-specified tolerance.

Projected Gradient Descent

The projected gradient algorithm is a standard approach to optimizing over a set of constraints. When the symbolic parameters, $\mathbf{w}_{sy}$, are constrained to Δ^r, and given an initial step size parameter $\eta^{(0)} > 0$ and feasible starting point $(\mathbf{w}_{sy}^0, \mathbf{w}_{nn}^0) \in \Delta^r \times \mathcal{W}_{nn}$, the projected gradient algorithm for Deep HL-MRF symbolic weight learning is defined

by the process:

$$\begin{aligned}
\mathbf{w}_{sy}^{(k+1)} &\leftarrow P_{\Delta^r}(\mathbf{w}_{sy}^{(k)} - \eta^{(k)} \nabla_{\mathbf{w}_{sy}} \mathcal{L}(\tilde{E}(\cdot,\cdot,\cdot,\mathbf{w}_{sy}^{(k)},\mathbf{w}_{nn}^{(k)}),S)) \\
\mathbf{w}_{nn}^{(k+1)} &\leftarrow \mathbf{w}_{nn}^{(k)} - \eta^{(k)} \nabla_{\mathbf{w}_{nn}} \mathcal{L}(\tilde{E}(\cdot,\cdot,\cdot,\mathbf{w}_{sy}^{(k)},\mathbf{w}_{nn}^{(k)}),S) \\
\eta^{(k+1)} &\leftarrow \frac{\eta^{(0)}}{k+1},
\end{aligned}$$

where $P_{\Delta^r}(\cdot)$ is the euclidean projection operator for the feasible set Δ^r.

Mirror Descent

Alternatively, minimizing the learning objective with the symbolic weights constrained to the unit simplex is achieved via mirror descent with the softmax link function [Kivinen and Warmuth, 1997, Shalev-Shwartz, 2012]. Given an initial step size parameter $\eta^{(0)} > 0$ and feasible starting point $(\mathbf{w}_{sy}^0, \mathbf{w}_{nn}^0) \in \Delta^r \times \mathcal{W}_{nn}$, the mirror descent updates are

$$\begin{aligned}
\mathbf{w}_{sy}^{(k+1)}[i] &\leftarrow \frac{\mathbf{w}_{sy}^{(k)}[i] \exp\{-\eta^{(k)} \frac{\partial \mathcal{L}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy}^{(k)},\mathbf{w}_{nn}^{(k)}),S)}{\partial \mathbf{w}_{sy}^{(k)}[i]}\}}{\sum_{j=1}^r \mathbf{w}_{sy}^{(k)}[j] \exp\{-\eta^{(k)} \frac{\partial \mathcal{L}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy}^{(k)},\mathbf{w}_{nn}^{(k)}),S)}{\partial \mathbf{w}_{sy}^{(k)}[j]}\}}, \quad \forall i = 1, \cdots, r \\
\mathbf{w}_{nn}^{(k+1)} &\leftarrow \mathbf{w}_{nn}^{(k)} - \eta^{(k)} \nabla_{\mathbf{w}_{nn}} \mathcal{L}(E(\cdot,\cdot,\cdot,\mathbf{w}_{sy}^{(k)},\mathbf{w}_{nn}^{(k)}),S) \\
\eta^{(k+1)} &\leftarrow \frac{\eta^{(0)}}{k+1}.
\end{aligned}$$

The symbolic parameters are guaranteed to satisfy the simplex constraints at every iteration.

Chapter 6

Empirical Analysis

In this chapter, I perform an empirical analysis of the NeSy-EBM modeling paradigms and learning algorithms presented in this work using the NeuPSL system introduced in Chapter 5. My experiments are designed to investigate the four following research questions:

- RQ1: What are the empirical runtime tradeoffs among the presented NeuPSL inference approaches?
- RQ2: Can the NeSy-EBM framework enhance the accuracy and reasoning capabilities of deep learning models?
- RQ3: Can the value-function gradients provided in Theorem 6 be used as a reliable descent direction for value-based learning losses?
- RQ4: Can symbolic constraints be used to train a deep learning model with partially labeled data?
- RQ5: What are the prediction performance and runtime tradeoffs among the presented modular, gradient descent, bilevel, and stochastic policy learning techniques?

The empirical analysis is organized into three subsections. First, in Section 6.1, I introduce the neural-symbolic datasets and models used in the experiments. In Section 6.2, I examine the runtime and the prediction performance of NeSy-EBMs

for constraint satisfaction, few-shot, and zero-shot reasoning of the NeuPSL inference algorithms presented in Chapter 5. In Section 6.3, I evaluate the runtime and performance of the learning techniques presented in Chapter 4 for the applications of fine-tuning, few-shot learning, and semi-supervision. All code and data for reproducing the empirical analysis are available at `https://github.com/linqs/dickens-arxiv24`.

6.1 Datasets and Models

This section introduces the NeSy datasets and models, which will be utilized throughout the empirical analysis. A summary of the datasets studied in the the empirical analysis is provided in Table 6.1. Any modifications to the datasets and models made to answer specific research questions will be described in the following sections. Details and implementations of the model architectures of both the neural and symbolic components are available at `https://github.com/linqs/dickens-arxiv24`.

Table 6.1: Datasets with data source citation and the NeSy-EBM model task. Every dataset task is a structured prediction problem.

Dataset	**Citation**	**Task**	**Perf. Metric**
MNIST-Add-k	Manhaeve et al. (2021a)	Image Classification	Accuracy
Visual-Sudoku	Wang et al. (2019)	Puzzle Solving	Accuracy
Pathfinding	Vlastelica et al. (2020)	Min Cost Path.	Accuracy
Debate	Hasan and Ng (2013)	Stance Classification	AUROC
4Forums	Walker et al. (2012)	Stance Classification	AUROC
Epinions	Richardson et al. (2003)	Link Prediction	AUROC
DDI	Wishart et al. (2006)	Link Prediction	AUROC
Yelp	Kouki et al. (2015)	Regression	MAE
Citeseer	Sen et al. (2008)	Text Classification	Accuracy
Cora	Sen et al. (2008)	Text Classification	Accuracy
RoadR	Singh et al. (2021)	Scene Understanding	IoU & F1
Logical-Deduction	Srivastava et al. (2022)	Question Answering	Accuracy

- **MNIST-Add-k Dataset:** MNIST-Add-k is a canonical NeSy dataset introduced by Manhaeve et al. (2021a) where models must determine the sum of each pair of digits from two lists of MNIST images. An MNIST-Add$k$ equation consists of two lists of $k > 0$ MNIST images. For instance, $[3] + [5] = \mathbf{8}$ is an MNIST-Add1 equation, and $[0, 4] + [3, 7] = \mathbf{41}$ is an MNIST-Add2

equation.

Evaluation: For all experiments, I evaluate models over 5 splits of the low-data setting proposed by Manhaeve et al. (2021a) with 600 total images for training and $1,000$ images each for validation and test. Prediction performance in this setting is measured by the accuracy of the image classifications and the inferred sums. Constraint satisfaction consistency in this setting is the proportion of predictions that satisfy the semantics of addition.

Baseline Architecture: The baseline neural architecture for all MNIST-Add$k$ datasets is a ResNet18 convolutional neural network backbone [He et al., 2016] with a 2-layer multi-layer perceptron (MLP) prediction head. The baseline is trained and applied as a digit classifier. Further, to allow the baseline to leverage the unlabeled training data in the semi-supervised settings, the digit classifier backbone is pre-trained using the SimCLR self-supervised learning framework [Chen et al., 2020]. Augmentations are used to obtain positive pairs for the contrastive pre-training process.

NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline digit classifier and a symbolic component created with NeuPSL that encodes the semantics of addition. The target variables of the symbolic component are the labels of the MNIST digits and their sum. The neural classification is used as a prior for the digit labels.

- **Visual-Sudoku Dataset:** Visual-Sudoku, first introduced by Wang et al. (2019), is a dataset containing a collection of 9×9 Sudoku puzzles constructed from MNIST images. In each puzzle, 30 cells are filled with MNIST images and are referred to as *clues*. The remaining cells are empty. The task is to correctly classify all clues and fill in the empty cells with digits that satisfy the rules of Sudoku: no repeated digits in any row, column, or box.

  Evaluation: For all experiments, results are reported across 5 splits with 20 puzzles for training and 100 puzzles each for validation and test. There is an equal number of MNIST images (600) in the training datasets for Visual-Sudoku

and MNIST-Add-k. Prediction performance in this setting is measured by the accuracy of the image classifications. Constraint satisfaction consistency in this setting is the proportion of predictions that satisfy the rules of Sudoku.

Baseline Architecture: The baseline neural architecture for Visual-Sudoku is the same as that of the MNIST-Add$k$.

NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline digit classifier and a symbolic component created with NeuPSL that encodes the rules of Sudoku. The target variables of the symbolic component are the labels of the clues and the empty cells. The neural classification is used as a prior for the clues.

- **Pathfinding Dataset:** Pathfinding is a NeSy dataset introduced by Vlastelica et al. (2020) consisting of 12000 randomly generated images of terrain maps from the Warcraft II tileset. The images are partitioned into 12×12 grids where each vertex represents a terrain with a cost. The task is to find the lowest cost path from the top left to the bottom right corner of each image.

  Evaluation: For all experiments, results are reported over 5 splits generated by partitioning the images into sets of $10,000$ for training, $1,000$ for validation, and $1,000$ for testing. Prediction performance in this setting is measured by the proportion of valid predicted paths, i.e., continuous, and that have a minimum cost. Constraint satisfaction continuity in this setting is measured by the proportion of predictions with a continuous predicted path.

  Baseline Architecture: The baseline neural architecture for the Pathfinding dataset is a ResNet18 convolutional neural network. The input of the ResNet18 path-finder baseline is the full Warcraft II map, and the output is the predicted shortest path. The model is trained using the labeled paths from the training data set.

  NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline path-finder and a symbolic component created with NeuPSL that encodes end-points and continuity constraints, i.e., the path from the top

left corner of the map to the bottom right corner must be continuous. The target variables of the symbolic component are variables indicating whether a vertex of the map grid is on the path. The neural classification is used as a prior for the path, and the symbolic component finds a valid path near the neural prediction.

- **4Forums and CreateDebate**: Stance-4Forums and Stance-CreateDebate are two datasets containing dialogues from online debate sites: `4forums.com` and `createdebate.com`, respectively. In this paper, I study stance classification, i.e., the task of identifying the stance of a speaker in a debate as being for or against. The evaluation protocol and models follow Sridhar et al. (2015).

  Evaluation: For all experiments, I evaluate models over 5 splits. Prediction performance is measured by the AUROC of the stance classifications.

  Baseline Architecture: The baseline neural architecture for Stance-4Forums and Stance-CreateDebate is a logistic regression classifier using features including n-grams, lexical category counts, and text lengths.

  NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline logistic regression model and a symbolic component created with NeuPSL that encodes domain knowledge of agreement and disagreement using user interactions.

- **Epinions**: Epinions is a trust network with $2,000$ individuals connected by $8,675$ directed edges representing whether they know each other and whether they trust each other Richardson et al. [2003]. I study link prediction, which in this setting is whether two individuals trust each other. The evaluation protocol and models follow Bach et al. (2017).

  Evaluation: For all experiments, I evaluate models over 5 splits. Prediction performance is measured by the AUROC of the link predictions. In each of the 5 data splits, the entire network is available, and the prediction performance is measured on $\frac{1}{8}$ of the trust labels. The remaining set of labels are available for training.

Baseline Architecture: There is no neural component in this model.

NeSy-EBM Architecture: The symbolic component encodes multiple rules relating trust to the users an individual knows.

- **DDI**: Drug-drug interaction (DDI) is a network of 315 drugs and $4,293$ interactions derived from the DrugBank database [Wishart et al., 2006]. The edges in the drug network represent interactions and seven different similarity metrics. In this paper, we perform link prediction, which in this setting infering unknown drug-drug interactions. The evaluation protocol and models follow Sridhar et al. (2016).

  Evaluation: For all experiments, I evaluate models over 5 splits. Prediction performance is measured by the AUROC of the link predictions.

  Baseline Architecture: The baseline neural architecture for DDI is a collection of similarity-based models.

  NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline similarity based models and a symbolic component created with NeuPSL that combines the baseline predictions and utilizes known interactions.

- **Yelp**: Yelp is a network of $34,454$ users and $3,605$ items connected by $99,049$ edges representing ratings. The task is to predict missing ratings, in this setting we model this as a regression task. The evaluation protocol and models follow Kouki et al. (2015).

  Evaluation: For all experiments, I evaluate models over 5 splits. Prediction performance is measured by the MAE of the predicted ratings.

  Baseline Architecture: The baseline neural architecture for DDI is a collection of matrix factorization and similarity models.

  NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline models and a symbolic component created with NeuPSL that combines the baseline predictions and utilizes social relationships between users.

- **Citeseer and Cora Dataset:** Citeseer and Cora are two widely studied

citation network node classification datasets first introduced by Sen et al. (2008). Citeseer consists of $3,327$ scientific publications classified into one of 6 topics, while Cora contains $2,708$ scientific publications classified into one of 7 topics.

Evaluation: For all experiments, I evaluate models over 5 randomly sampled splits using 20 examples of each topic for training, 200 of the nodes for validation, and 1000 nodes for testing. Prediction performance in this setting is measured by the categorical accuracy of a paper label.

Baseline Architecture: The baseline neural architecture for the Citation network settings is a Simple Graph Convolutional Network (SGC) [Wu et al., 2019]. SGCs are graph convolutional networks with linear activations in the hidden layers to reduce computational complexity. The SGC neural baseline uses bag-of-words feature vectors associated with each paper as node features and citations as bi-directional edges. Then, a MLP is trained to predict the topic label given the SGC-transformed features.

NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline SGC and a symbolic component created with NeuPSL that encodes the homophilic structure of the citation network, i.e., two papers connected in the network are more likely to have the same label. Target variables indicate the degree to which a paper has a particular topic. The neural classification is used as a prior for the labels of the nodes, and the symbolic component propagates this knowledge to its neighbors.

- **RoadR Dataset:** RoadR is an extension of the ROAD (Road event Awareness Dataset) dataset, initially introduced by Singh et al. (2021). The ROAD dataset was developed to evaluate the situational awareness of autonomous vehicles in various road environments, weather conditions, and times of day by. It contains 22 videos, $122k$ labeled frames, $560k$ bounding boxes, and a total of $1.7M$ labels, which include $560k$ agents, $640k$ actions, and $499k$ locations. RoadR builds upon this by adding 243 logical requirements that must be satisfied, further enhancing its utility for testing autonomous vehicles [Giunchiglia et al.,

2023]. For instance, a traffic light should never be simultaneously predicted as red and green.

Evaluation: For all experiments, I evaluate models with 15 videos for training and 3 videos for testing. Prediction performance in this setting is measured by the matching boxes using Intersection over Union (IoU) and then multi-class f1. Constraint satisfaction consistency in this setting is the proportion of frame predictions with no constraint violations.

Baseline Architecture: The baseline neural architecture for the RoadR dataset is a DEtection TRansformer (DETR) model with a ResNet50 backbone [Carion et al., 2020]. The baseline is trained and applied to detect objects in a frame, along with a multi-label classification for its class labels (e.g., car, red, traffic light, etc.).

NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline object detector and classifier and a symbolic component created with NeuPSL that encodes the logical requirements. The target variables are the classification labels of a bounding box. The neural classification is used as both the bounding box creation and a prior on the labels that the symbolic component uses as a starting point to find a valid solution to the constraints.

- **Logical-Deduction** is a multiple-choice question-answering dataset introduced by Srivastava et al. [2022]. These questions require deducing the order of a sequence of objects given a natural language description and then answering a multiple-choice question about that ordering.

  Evaluation: I report results for a single test set of 300 deduction problems, with a prompt containing two examples. Prediction performance in this setting is measured by the accuracy of the predicted multiple-choice answer.

  Baseline Architecture: The baseline neural architecture for the Logical-Deduction dataset is the models presented in Pan et al. (2023) on GPT-3.5-turbo and GPT-4 OpenAI [2024]. Each model is run using *Standard* and *Chain-of-Thought (CoT)* [Wei et al., 2022] prompting.

NeSy-EBM Architecture: The NeSy-EBM architecture is a composition of the baseline LLM that is being prompted to create the constraints within the symbolic program. Symbolic inference is then performed, and the output is returned to the LLM for final evaluation. In this sense, the NeSy-EBM writes a program to perform reasoning rather than depending on the language model to reason independently.

6.2 NeSy-EBM Inference

In this section I investigate research questions RQ1 and RQ2. First, I examine the runtime of the dual BCD NeuPSL inference algorithms presented in Section 5.4. Then I study the prediction performance of NeSy-EBMs for constraint satisfaction and few-shot reasoning. For all experiments in this section, I employ a modular training approach to obtain neural and symbolic component weights. Specifically, neural components undergo training using the complete training dataset for supervision, and symbolic weights are either fixed or trained using a random grid search. A *DSPot* NeSy-EBM is used for Logical Deduction and a *DSPar* NeSy-EBM is used for all other datasets.

6.2.1 Runtime

To answer research question RQ1, I investigate the runtime of NeuPSL inference using the new inference algorithms: connected component parallel dual BCD (CC D-BCD) and lock-free parallel dual BCD (LF D-BCD), and the alternating direction method of multipliers (ADMM) Boyd et al. [2010], the current state-of-the-art inference algorithm. I also evaluate the performance of Gurobi, a leading off-the-shelf optimizer, and subgradient descent (GD) in Appendix C.2 and show ADMM and dual BCD consistently match or outperform GD and the proprietary solver. All inference algorithms have access to the same computing resources detailed in Section C.1. Moreover, for each algorithm, I run a hyperparameter search, detailed in Appendix C.2. The hyperparameter configuration yielding a prediction performance that is within a standard deviation of the best and completed with the lowest runtime is

reported. All algorithms are stopped when the L_∞ norm of the primal variable change between iterates is less than 0.001.

Table 6.2: Time in seconds for inference using ADMM and my proposed CC D-BCD and LF D-BCD algorithms on nine datasets.

	ADMM	**CC D-BCD**	**LF D-BCD**
Debate	9.98 ± 1.13	$\mathbf{0.05 \pm 0.02}$	$\mathbf{0.05 \pm 0.03}$
4Forums	15.17 ± 0.74	0.11 ± 0.02	$\mathbf{0.05 \pm 0.01}$
Epinions	0.36 ± 0.041	1.84 ± 0.4	$\mathbf{0.26 \pm 0.04}$
Citeseer	0.63 ± 0.07	1.36 ± 0.24	$\mathbf{0.49 \pm 0.08}$
Cora	$\mathbf{0.71 \pm 0.07}$	6.46 ± 3.5	0.79 ± 0.19
DDI	7.85 ± 0.28	31.47 ± 0.17	$\mathbf{1.76 \pm 0.17}$
Yelp	$\mathbf{6.37 \pm 1.19}$	48.44 ± 3.82	7.58 ± 0.48
MNIST-Add1	11.45 ± 1.32	$\mathbf{10.23 \pm 1.04}$	115 ± 45
MNIST-Add2	285 ± 66	$\mathbf{29.09 \pm 8.00}$	$1,189 \pm 16$

The total average inference runtime in seconds for each algorithm and model is provided in Table 6.2. Surprisingly, despite the potential for an inexact solution to the BCD steplength subproblem, LF D-BCD is faster than CC D-BCD in the first 7 datasets and demonstrates up to 6× speedup over CC D-BCD in Yelp. However, in MNIST-Add datasets, CC D-BCD is up to 10× faster than LF D-BCD as there is a high number of tightly connected components, one for each addition instance. This behavior highlights the complementary strengths of the two parallelization strategies. LF D-BCD should be applied to problems with larger factor graph representations that are connected, while CC D-BCD is effective when there are many similarly sized connected components.

Dual BCD and Regularization

The regularization parameter added to the LCQP formulation of NeuPSL inference in (5.9) ensures strong convexity of the optimal value of the energy function. However, adding regularization makes the new formulation an approximation. Here, the runtime and prediction performance of the D-BCD inference algorithm is evaluated at varying levels of regularization to understand its effect on NeuPSL inference. The regularization parameter varies in the range $\epsilon \in \{100, 10, 1, 0.1, 0.01\}$. The D-BCD algorithm is stopped when the primal-dual gap drops below $\delta = 0.1$ Inference time is

provided in seconds, and the performance metric is consistent with Table 6.1. Results are provided in Table 6.3.

Table 6.3: D-BCD Inference time in seconds and prediction performance with varying values for the LCQP regularization parameter ϵ.

Dataset	ϵ	**Time (sec)**	**Perf.**
CreateDebate	100	0.02 ± 0.01	64.77 ± 10.61
	10	0.02 ± 0.01	64.83 ± 10.53
	1	0.02 ± 0.01	64.74 ± 10.67
	0.1	0.05 ± 0.02	65.39 ± 9.07
	0.01	0.42 ± 0.51	66.01 ± 9.35
4Forums	100	0.11 ± 0.02	61.31 ± 6.17
	10	0.10 ± 0.03	61.26 ± 6.16
	1	0.09 ± 0.01	61.12 ± 6.18
	0.1	0.43 ± 0.11	62.73 ± 5.46
	0.01	7.11 ± 3.05	62.31 ± 5.47
Epinions	100	0.33 ± 0.05	72.59 ± 2.27
	10	0.28 ± 0.04	72.69 ± 2.21
	1	0.33 ± 0.05	74.24 ± 1.95
	0.1	1.08 ± 0.16	77.05 ± 1.06
	0.01	5.21 ± 0.37	77.45 ± 0.70
Citeseer	100	0.95 ± 0.14	71.28 ± 1.31
	10	1.00 ± 0.12	71.28 ± 1.30
	1	1.48 ± 0.29	71.59 ± 1.01
	0.1	7.01 ± 1.57	71.75 ± 1.10
	0.01	62.41 ± 14.67	71.92 ± 1.09
Cora	100	4.53 ± 2.20	81.31 ± 1.73
	10	4.56 ± 2.39	81.57 ± 1.83
	1	7.36 ± 4.19	81.48 ± 1.70
	0.1	42.24 ± 25.06	81.88 ± 1.82
	0.01	269.45 ± 49.50	81.79 ± 1.72
DDI	100	24.56 ± 0.25	94.85 ± 0.00
	10	29.23 ± 0.59	94.85 ± 0.00
	1	47.15 ± 0.95	94.82 ± 0.00
	0.1	280.62 ± 5.19	94.80 ± 0.00
	0.01	266.07 ± 42.68	94.81 ± 0.00
Yelp	100	105.60 ± 5.03	0.23 ± 0.01
	10	$3,239 \pm 81$	0.22 ± 0.01
	1	$3,227 \pm 54$	0.19 ± 0.01
	0.1	421 ± 202	0.18 ± 0.00
	0.01	$2,472 \pm 297$	0.18 ± 0.00

Table 6.3 shows there is a consistent correlation between the LCQP regularization parameter and the runtime and performance of inference. As ϵ increases, there is a significant decrease in the runtime as the D-BCD algorithm can find a solution with a gradient meeting the stopping criterion in fewer iterations. Notably, for the Citeseer inference problem, the D-BCD algorithm realizes a roughly $45\times$ speedup. On the other hand, while the runtime performance improves with increasing ϵ, the prediction performance can sometimes decay. There is a tradeoff between runtime

and prediction performance when setting the ϵ regularization parameter.

6.2.2 Prediction Performance

Next, to answer research question RQ2, I examine the prediction performance of NeSy-EBMs for the motivating applications of constraint satisfaction and few-shot reasoning.

Constraint Satisfaction and Joint Reasoning

To investigate constraint satisfaction and joint reasoning, I use the dataset settings outlined in Section 6.1 for Visual-Sudoku, Pathfinding, RoadR, Citeseer, and Cora. Additionally, I introduce the following variant of the MNIST-Add-k dataset.

- **MNIST-Add**k: The $k = 1, 2, 4$ MNIST-Add$k$ datasets with the sums of the MNIST-Add-k equations available as observations during inference. Prediction performance is measured by the accuracy of the image classifications.

The MNIST-Add-k modification allows the NeSy-EBM to use the semantics of addition and the sum observation to form constraints to correct the neural component predictions. For instance, consider the MNIST-Add-1 equation $[\,3\,] + [\,5\,] = \mathbf{8}$. If the neural component incorrectly classifies the first MNIST image, 3, as an 8 with low confidence but correctly classifies the second MNIST image, 5, as a 5 with high confidence, then it can use the sum label, 8, to correct the first digit label.

Table 6.4: Digit accuracy and constraint satisfaction consistency of the ResNet18 and NeuPSL models on the MNIST-Add-k and Visual-Sudoku datasets.

	ResNet18		**NeuPSL**	
	Digit Acc.	**Consistency**	**Digit Acc.**	**Consistency**
MNIST-Add1	97.60 ± 0.55	93.04 ± 1.33	$\mathbf{99.80 \pm 0.14}$	$\mathbf{100.0 \pm 0.00}$
MNIST-Add2		86.56 ± 2.72	$\mathbf{99.68 \pm 0.22}$	$\mathbf{100.0 \pm 0.00}$
MNIST-Add4		75.04 ± 4.81	$\mathbf{99.72 \pm 0.29}$	$\mathbf{100.0 \pm 0.00}$
Visual-Sudoku		70.20 ± 2.17	$\mathbf{99.37 \pm 0.11}$	$\mathbf{100.0 \pm 0.00}$

Tables 6.4 to 6.6 report the prediction performance and constraint satisfaction consistency of a neural baseline and NeuPSL model on the MNIST-Add$k$, Visual-Sudoku, Pathfinding, and RoadR datasets, respectively. Across all settings, the

Table 6.5: Accuracy of finding a minimum cost path (Min. Cost Acc.) and consistency in satisfying continuity constraints (Continuity) of the ResNet18 and NeuPSL models on the Pathfinding dataset.

	ResNet18		NeuPSL	
	Min. Cost Acc.	**Continuity**	**Min. Cost Acc.**	**Continuity**
Pathfinding	80.12 ± 22.44	84.80 ± 17.11	$\mathbf{90.02 \pm 11.70}$	$\mathbf{100.0 \pm 0.00}$

Table 6.6: Object detection F1 and constraint satisfaction consistency of the DETR and NeuPSL models on the RoadR dataset.

	DETR		NeuPSL	
	F1	**Consistency**	**F1**	**Consistency**
RoadR	0.457	27.5	**0.461**	**100.0**

baseline neural models frequently violate constraints within the test dataset. Further, the frequency of these violations increases with the complexity of the constraints. This behavior is best illustrated in the MNIST-Add$k$ datasets, where consistency decreases as the number of digits, k, increases. This decline can be attributed to the baseline ResNet18 model treating each digit prediction independently and thus failing to account for the dependencies from the sum relation. Moreover, in the RoadR experiment, the DETR baseline adheres to road event constraints only 27.5% of the time. On the other hand, NeuPSL always satisfies the problem constraints in the MNIST-Add$k$, Visual-Sudoku, Pathfinding, and RoadR datasets, achieving 100% consistency. This is because the DSPar NeSy-EBM models used in these experiments can enforce constraints on all target variables. This allows the NeSy-EBM models to leverage the structural relations inherent in the constraints to infer target variables and jointly improve prediction accuracy. Prediction performance gains from constraint satisfaction and joint reasoning are possible when the neural component accurately quantifies its confidence. The symbolic component uses the confidence of the neural component and the constraints together to correct the neural model's erroneous predictions. This observation motivates an exciting avenue of future research: exploring whether calibrating the confidence of the neural component can further improve the structured prediction and joint reasoning capabilities of NeSy-EBMs.

Table 6.7: Node classification accuracy of the SGC and NeuPSL models on the Citeseer and Cora datasets.

	SGC	NeuPSL
Citeseer	65.14 ± 2.96	$\mathbf{66.52 \pm 3.26}$
Cora	80.90 ± 1.54	$\mathbf{81.82 \pm 1.73}$

Unlike MNIST-Add$k$, Visual-Sudoku, Pathfinding, and RoadR, which have hard constraints on the target variables, the citation network datasets showcase the capacity of NeSy-EBMs to perform joint reasoning with constraints and dependencies that are not strictly adhered to. For Citeseer and Cora, NeuPSL enhances prediction accuracy by leveraging the homophilic structure of the citation networks, i.e., papers that are linked tend to share topic labels.

Table 6.7 reports the baseline and NeuPSL NeSy-EBM prediction performance on the citation network node classification datasets. In all instances, NeuPSL outperforms the baseline. The performance gain from NeuPSL in the citation network experiments is verified to be statistically significant with a paired t-test and p-value less than 0.05.

Few-Shot Reasoning

To investigate the few-shot reasoning capabilities of NeSy-EBMs, I use the Logic Deduction dataset and DSPot NeSy-EBM model outlined in Section 6.1. Similarly, in the question-answering logical deduction problem, NeuPSL uses an LLM to generate rules representing the dependencies described in natural language. Although the LLM may sometimes fail to generate accurate rules, NeuPSL will consistently use the rules for logical reasoning.

Table 6.8: Comparison of accuracy in answering logical deduction questions using two large language models, GPT-3.5-turbo and GPT-4 OpenAI [2024], across three methods: Standard few-shot prompting, Chain of Thought (CoT) few-shot prompting, and NeuPSL.

	LLM	Standard	CoT	NeuPSL
Logical Deduction	*GPT-3.5-turbo*	40.00	42.33	**70.33**
	GPT-4	71.33	75.25	**90.67**

Table 6.8 reports the baseline and NeuPSL NeSy-EBM prediction performance on logical deduction datasets. NeuPSL obtains a 15% improvement over the GPT-3.5-turbo and GPT-4 LLMs with standard and chain of thought few-shot prompting. This performance gain is achieved despite the fact that the LLM neural component in NeuPSL could produce invalid syntax or an infeasible set of logical constraints. The LLM was able to produce valid programs 89.0% and 98.7% of the time with GPT-3.5-turbo and GPT-4, respectively. This observation motivates a promising avenue of future research in employing self-refinement approaches similar to that of Pan et al. [2023] to correct the infeasible programs and further improve LLM reasoning capabilities.

6.3 NeSy-EBM Learning

Next, I investigate NeSy-EBM learning to answer research questions RQ3, RQ4, and RQ5. The analysis is divided into two parts. First, I study the runtime of the learning techniques, then I examine the prediction performance. I compare the results of two value-based losses, Energy and Structure Perceptron (SP), and two minimizer-based losses, Mean Square Error (MSE), and Binary Cross Entropy (BCE). In all but the semi-supervised prediction performance experiments, models within this section use the *DSPar* modeling paradigm. Moreover, in this section I study the following modular and end-to-end NeSy-EBM learning algorithms.

- **RGS**: The modular random grid search (RGS) algorithm optimizing the dataset evaluation metric.
- **CRS**: The modular continuous random search (CRS) algorithm optimizing the dataset evaluation metric.
- **BOWL**: The modular Bayesian optimization for weight learning algorithm (BOWL) optimizing dataset evaluation metric.
- **Energy**: Direct gradient descent minimizing the value-based energy loss.

- **SP**: Direct gradient descent minimizing the value-based structured-perceptron (SP) loss.
- **MSE**: Bilevel value-function optimization for NeSy-EBM learning algorithm minimizing the mean-squared error (MSE) and energy loss.
- **BCE**: Bilevel value-function optimization for NeSy-EBM learning algorithm minimizing the binary cross entropy (BCE) and energy loss.
- **IndeCateR**: Stochastic policy optimization algorithm optimizing the dataset evaluation metric.

Theorem 6 in Section 4.2 is used to compute the learning gradients with respect to the neural output and symbolic weights for the Energy, SP, and Bilevel algorithms. Similarly, the Independent Categorical REINFORCE (IndeCateR) gradient estimator [De Smet et al., 2023] estimate is used to compute the learning gradients with respect to the neural output and symbolic weights for stochastic policy optimization. Then, gradients with respect to the neural parameters are found via backpropagation for all methods. The neural parameters are updated via AdamW [Loshchilov and Hutter, 2019], and the symbolic parameters are updated using gradient descent with a fixed step size. Additional details on the hardware and hyperparameters settings of the learning algorithms are provided in Appendix C.

6.3.1 Runtime

Inference and Learning

I study how the algorithms applied to solve inference affect the learning runtime with the SP and MSE losses. Specifically, I examine the cumulative time required for ADMM and D-BCD inference to complete 500 epochs. Hyperparameters used for SP and MSE learning are reported in Appendix C.2.1. For inference, I apply the same hyperparameters used in the previous section and the fastest parallelization method for D-BCD.

Table 6.9 shows that the D-BCD algorithm consistently results in the lowest total inference runtime, validating its ability to leverage warm starts to improve learning

Table 6.9: Cumulative inference time in seconds for ADMM and D-BCD during learning with gradient-descent on SP and bilevel optimization on MSE.

	Gradient Descent - SP		**Bilevel - MSE**	
	ADMM	**D-BCD**	**ADMM**	**D-BCD**
Debate	10.68 ± 8.63	$\mathbf{0.34 \pm 0.36}$	49.00 ± 31.23	$\mathbf{0.62 \pm 0.09}$
4Forums	11.87 ± 12.81	$\mathbf{0.65 \pm 0.05}$	67.09 ± 13.79	$\mathbf{1.11 \pm 0.16}$
Epinions	12.54 ± 0.37	$\mathbf{1.33 \pm 0.06}$	17.48 ± 0.62	$\mathbf{2.27 \pm 0.98}$
Citeseer	167 ± 37	$\mathbf{41.57 \pm 6.39}$	225 ± 32	$\mathbf{70.01 \pm 5.86}$
Cora	183 ± 26	$\mathbf{48.16 \pm 5.82}$	241 ± 37	$\mathbf{79.62 \pm 13.77}$
DDI	$4,554 \pm 13$	$\mathbf{19.65 \pm 0.30}$	$7,652 \pm 218$	$\mathbf{52.78 \pm 4.23}$
Yelp	$1,835 \pm 47$	$\mathbf{114 \pm 4}$	$2,250 \pm 100$	$\mathbf{170 \pm 12}$
MNIST-Add1	$1,624 \pm 34$	$\mathbf{232 \pm 44}$	$2,942 \pm 109$	$\mathbf{2,738 \pm 93}$
MNIST-Add2	TIME-OUT	$\mathbf{804 \pm 106}$	TIME-OUT	$\mathbf{4,291 \pm 114}$

runtimes. Notably, on the DDI dataset, D-BCD achieves roughly a 100× speedup over ADMM. Moreover, on MNIST-Add2, ADMM timed out with over 6 hours of inference time for SP and MSE learning, while D-BCD accumulated less than 0.5 and 1.2 hours of inference runtime on average for SP and MSE, respectively.

Gradient Descent Symbolic Weight Learning

Here, I evaluate the practical convergence rate of two gradient descent methods proposed for learning symbolic weights of a NeuPSL model. Both algorithms use the dual BCD method for their inference subproblems using the regularization parameter $\lambda = 0.001$. Furthermore, for both algorithms, the negative log regularization parameter is set to 0.001 (Section 5.5.1). The initial stepsize hyperparameter is searched over the range $\eta \in \{1.0, 0.1, 0.01\}$. Both algorithms are stopped when the KL-divergence stopping criteria discussed in Section 5.5.2 is satisfied with $\delta = 0.01$ or after 1000 iterations.

Table 6.10: Number of learning iterations needed to converge on the structured perceptron learning loss.

	Mirror Descent	**Projected GD**
Epinions	38	59
Citeseer	70	597
Cora	57	398
Yelp	128	1000+

The results of the first-order learning convergence experiments for the structured perceptron learning loss are shown in Table 6.10. The mirror descent algorithms consistently finds a solution satisfying the stopping criteria in fewer iterations than projected gradient descent. This indicates the mirror descent algorithm more efficiently optimizes the learning objective over the unit simplex. For the remainder of the first-order learning experiments, the mirror descent algorithm is used on the symbolic parameters.

6.3.2 Prediction Performance

Impact of Dirichlet Parameter on Black-Box Performance

Here I evaluate the impact of the Dirichlet concentration parameter, A, on the performance of CRS and BOWL. I chose four different values for $A = \{10, 1, 0.1, 0.01\}$ and evaluate the methods on one discrete dataset: Citeseer, and one continuous dataset: Jester. Large values of the concentration hyperparameter, A, generate samples concentrated towards the center (near equally valued weights), while low values of the parameter bias the distribution towards the edges and vertices of the simplex.

Table 6.11: The mean and standard deviation of the performance of different search-based approaches with varying the A parameter in the Dirichlet distribution.

Datasets	**Methods**	**A = 10**	**A = 1**	**A = 0.1**	**A = 0.01**
Citeseer	*CRS*	84.40 ± 0.03	84.40 ± 0.02	84.30 ± 0.02	84.40 ± 0.03
	BOWL	84.40 ± 0.02	84.40 ± 0.03	84.50 ± 0.02	84.30 ± 0.02
Jester	*CRS*	$0.064 \pm 1.8e\text{-}3$	$0.054 \pm 5.6e\text{-}4$	$0.053 \pm 9.4e\text{-}4$	$0.056 \pm 2.2e\text{-}3$
	BOWL	$0.053 \pm 2.3e\text{-}4$	$0.053 \pm 4.0e\text{-}4$	$0.053 \pm 7.3e\text{-}4$	$0.053 \pm 2.3e\text{-}4$

Table 6.11 reports the metrics obtained for different values of A on Citeseer and Jester with CRS and BOWL. The effect of A on Citeseer is minimal in both methods. This is likely because the accuracy function with respect to the weights is nearly uniform, and small weight changes have minimal impact. Therefore as long as there is at least one sampled point in a region, it is sufficient to get a good solution. Next, considering the Jester dataset, the parameter A has a more significant impact on CRS. CRS performs the best for a value of $A = 0.1$; then, as it is increased to 10,

it produces the worst MSE value. This is because the samples generated by the Dirichlet distribution using the relatively large parameter values do not include many high-performing configurations. Finally, BOWL is shown to be more robust to the parameter A as it uses a large initial sampling from the Dirichlet distribution and an acquisition function to choose the next point.

Symbolic Weight Learning

In this subsection I focus on symbolic weight learning performance, in other words, the neural components are first trained using supervised neural losses and are then frozen. I use the following dataset variations.

- **Citeseer and Cora** The NeSy-EBM models for symbolic weight learning performance experiments are extended versions from Bach et al. (2017) Bach et al. [2017]. Specifically, a copy of each rule that is specialized for the topic is made. Moreover, topic propagation across citation links is considered for papers with differing topics. For instance, the possibility of a citation from a paper with topic `'A'` could imply a paper is more or less likely to be topic `'B'`. The extended models are available at `https://github.com/linqs/dickens-arxiv24/tree/main/modular_learning/psl-extended-examples`. The models for learning prediction performance experiments are from Pryor et al. (2023a). The data and models are available at: `https://github.com/linqs/dickens-arxiv24/tree/main/citation/models/symbolic`.

Table 6.12: Prediction performance of HL-MRF models with symbolic weights trained with direct modular learning, gradient descent, and bilevel techniques.

	Modular		**Gradient Descent**		**Bilevel**	
	CRS	**BOWL**	**Energy**	**SP**	**MSE**	**BCE**
Debate	65.87 ± 10.09	$\mathbf{66.87 \pm 9.54}$	64.76 ± 9.54	64.68 ± 11.05	65.33 ± 11.98	64.83 ± 9.70
4Forums	62.63 ± 6.49	62.29 ± 5.34	62.96 ± 6.11	63.15 ± 6.40	64.22 ± 6.41	$\mathbf{64.85 \pm 6.01}$
Epinions	79.35 ± 1.52	79.49 ± 1.60	78.96 ± 2.29	79.85 ± 1.62	$\mathbf{81.18 \pm 2.21}$	80.89 ± 2.32
Citeseer	63.45 ± 2.02	63.31 ± 2.52	70.29 ± 1.54	70.92 ± 1.33	71.22 ± 1.56	$\mathbf{71.94 \pm 1.17}$
Cora	61.52 ± 1.70	62.81 ± 0.31	54.30 ± 1.74	74.16 ± 2.32	81.05 ± 1.41	$\mathbf{81.07 \pm 1.31}$
DDI	94.90 ± 0.27	95.00 ± 0.13	94.54 ± 0.00	94.61 ± 0.00	94.70 ± 0.00	$\mathbf{95.08 \pm 0.00}$
Yelp	18.08 ± 1.74	18.68 ± 0.62	18.11 ± 0.34	18.57 ± 0.66	18.14 ± 0.36	$\mathbf{17.93 \pm 0.50}$

Table 6.12 reports the prediction performance achieved by each of the four learning

techniques across the seven modular datasets. Models trained with bilevel-based losses consistently achieve better average predictive performance than those trained with value-based losses. Notably, on the Cora dataset, the NeuPSL model trained with the BCE loss achieved a remarkable improvement of over six percentage points compared to the SP loss, which was the better-performing value-based loss. The models trained with the Energy and SP loss suffered from a collapsed solution, i.e., symbolic parameters giving nearly equal energy to all settings of the target variables.

End-to-End Learning

In this section I analyze the performance and empirical convergence properties of the three end-to-end gradient-based NeSy-EBM learning techniques: Energy, Bilevel, and IndeCateR. To investigate the performance of the NeSy-EBM learning techniques, I use the dataset settings outlined in Section 6.1 for Citeseer and Cora. Additionally, I introduce the following variants of the MNIST-Add$k$, Visual-Sudoku, and Pathfinding datasets:

- **MNIST-Add$k$**: The $k = 1, 2$ MNIST-Add$k$ datasets with no digit supervision, i.e., parameters are learned only from the addition relations.
- **Visual-Sudoku**: A few-shot setting with 5 labeled examples of each of the 9 possible classes available for training. The remaining images in the training data are unlabeled, and the model must primarily rely on the Sudoku rules for learning.
- **Pathfinding**: A limited supervision setting where only 10% of the training data is labeled, and the remaining training data is unlabeled. Specifically, only 5% of the map vertices are observed to be on or off the labeled minimum cost path. In other words, supervision is distributed across maps, and the minimum cost paths for a map are only partially observed.

Table 6.13 presents the average and standard deviation of the prediction performance for the symbolic component of the NeuPSL NeSy-EBM model across the six datasets examined in this section. In five of the six datasets, the Bilevel learning

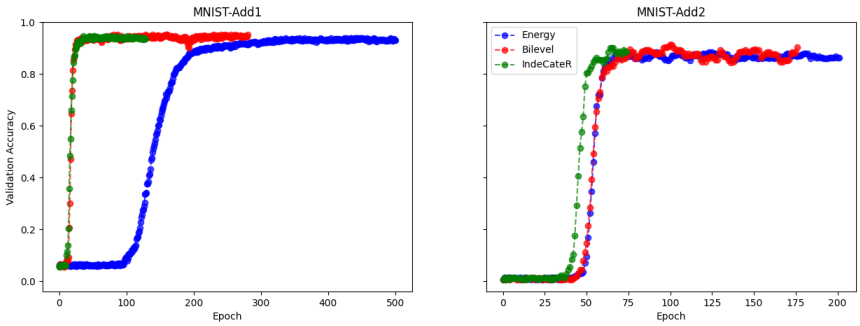
Table 6.13: The average and standard deviation of the prediction performance of NeuPSL NeSy-EBMs trained using end-to-end learning algorithms on 6 datasets.

	NeuPSL		
	Energy	**IndeCateR**	**Bilevel**
MNIST-Add1	93.80 ± 1.12	94.52 ± 0.99	$\mathbf{94.92 \pm 1.40}$
MNIST-Add2	87.92 ± 1.63	86.88 ± 1.82	$\mathbf{89.36 \pm 1.54}$
Visual-Sudoku	$\mathbf{98.12 \pm 0.37}$	TIMEOUT	98.10 ± 0.19
Path-Finding	22.53 ± 0.75	TIMEOUT	$\mathbf{22.85 \pm 1.33}$
Citeseer	67.04 ± 1.82	TIMEOUT	$\mathbf{67.96 \pm 1.11}$
Cora	80.40 ± 0.74	TIMEOUT	$\mathbf{81.88 \pm 0.65}$

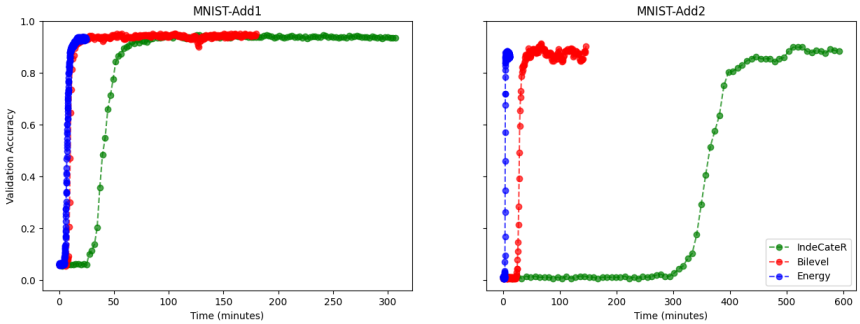
algorithm achieves the best results. Notably, in MNIST-Add1, IndeCateR's performance was comparable to Bilevel's. However, as the complexity of the target variable constraints increased, IndeCateR's performance deteriorated, exemplified by poor results in MNIST-Add2 and failures to find viable solutions within the allotted time in the other datasets.

While Energy generally underperformed compared to Bilevel across most settings, it was the fastest in execution time. For instance, Fig. 6.1 plots the validation image classification accuracy of the MNIST-Add1 and MNIST-Add2 NeuPSL NeSy-EBMs trained with the Energy, IndeCater, and Bilevel learning algorithms versus the training epoch and wall-clock time for a single fold. The Bilevel and IndeCateR algorithms reach higher validation performance levels than the Energy algorithm on both MNIST-Add$k$ datasets for the reported fold. This pattern is consistent with the average prediction performance results reported in Table 6.13 for MNIST-Add1. For the MNIST-Add2 dataset, on the other hand, the IndeCateR algorithm was timed out after 10 hours of training rather than allowing it to fully converge, which explains the drop in the relatively lower average test performance results in Table 6.13. Surprisingly, the IndeCateR algorithm has the best empirical rate of improvement with respect to training epochs on both datasets; the next best is Bilevel, and finally, Energy. However, the IndeCateR algorithm's per-iteration cost counteracts its advantage, and it has a significantly slower rate of improvement with respect to wall-clock time. On the other end of the spectrum, Energy has the slowest rate of prediction performance improvement, but its per iteration cost is low enough that it

Figure 6.1: Validation image classification accuracy versus training (a) epoch and (b) time in minutes for NeuPSL models trained with the Energy, IndeCateR, and Bilevel NeSy-EBM learning algorithms.



(a)



(b)

converges the fastest with respect to wall-clock time. The Bilevel algorithm balances the strengths of the two algorithms. It has a lower per-iteration cost because it only uses value-function gradients and optimizes a minimizer-based loss. The convergence results in Fig. 6.1 motivate future work on training pipelines that pre-train with a value-based loss, eithe and fine-tune with a more expensive minimizer-based loss to achieve the fastest training time and best final prediction performance.

Semi-Supervision

In this set of experiments, I investigate the effectiveness of the NeSy-EBM framework in training a deep learning model in a semi-supervised setting. This experiment aims to further investigate research questions RQ4 and RQ5. Specifically, I compare the prediction performance of a neural baseline trained solely with a supervised neural loss to that of a NeuPSL model's neural component (with the same architecture)

Table 6.14: Digit accuracy of the ResNet18 models trained with varying levels of supervision.

		ResNet18	
	Labeled	**Supervised**	**NeuPSL Semi-Supervised**
MNIST-Add1	1.00	$\mathbf{97.84 \pm 0.23}$	97.40 ± 0.51
	0.50	$\mathbf{97.42 \pm 0.30}$	97.02 ± 0.65
	0.10	93.05 ± 0.69	$\mathbf{96.78 \pm 0.80}$
	0.05	75.35 ± 0.33	$\mathbf{96.82 \pm 0.72}$
MNIST-Add2	1.00	$\mathbf{97.84 \pm 0.23}$	97.22 ± 0.19
	0.50	$\mathbf{97.42 \pm 0.30}$	96.84 ± 0.42
	0.10	93.05 ± 0.69	$\mathbf{95.14 \pm 1.21}$
	0.05	75.35 ± 0.33	$\mathbf{95.90 \pm 0.43}$
Visual-Sudoku	1.00	97.84 ± 0.23	$\mathbf{97.89 \pm 0.15}$
	0.50	$\mathbf{97.42 \pm 0.30}$	97.26 ± 0.70
	0.10	93.05 ± 0.69	$\mathbf{96.82 \pm 0.32}$
	0.05	75.35 ± 0.33	$\mathbf{96.49 \pm 0.67}$

trained using an end-to-end NeSy-EBM learning technique. In both cases, only a subset of the training set labels is available to the neural component. To enhance neural performance with a structured loss, the MNIST-Addk and Visual-Sudoku models in this section employ the *DSVar* modeling paradigm due to its simplicity and speed, while Pathfinding, Citeseer, and Cora models use the *DSPar* modeling paradigm. I use the following variants of four datasets for the experiments.

- **MNIST-Add**k: The $k = 1, 2$ MNIST-Add$k$ datasets with the proportion of image class labels available in the training data varying over $\{1.0, 0.5, 0.1, 0.05\}$. Prediction performance in this section is measured by the accuracy of the image classifications.

- **Visual-Sudoku**: The proportion of image class labels available in the training data varies over $\{1.0, 0.5, 0.1, 0.05\}$.

- **Pathfinding**: Supervision is distributed across all training maps, so the shortest paths in the training data are only partially observed. The proportion of vertex labels available in the training data varies over $\{1.0, 0.5, 0.1\}$.

- **Citeseer and Cora**: The proportion of paper topic labels available in the

Table 6.15: Topic accuracy of the trained SGC models with varying levels of supervision.

		SGC	
	Labeled	**Supervised**	**NeuPSL Semi-Supervised**
Citeseer	1.00	$\mathbf{76.12 \pm 1.71}$	75.92 ± 2.23
	0.50	$\mathbf{74.70 \pm 1.68}$	74.38 ± 1.82
	0.10	68.64 ± 1.06	$\mathbf{69.66 \pm 0.16}$
	0.05	64.56 ± 1.68	$\mathbf{66.12 \pm 1.22}$
Cora	1.00	$\mathbf{87.62 \pm 0.97}$	87.18 ± 1.08
	0.50	85.82 ± 0.50	$\mathbf{86.74 \pm 0.54}$
	0.10	80.88 ± 2.00	$\mathbf{81.96 \pm 2.62}$
	0.05	74.98 ± 3.32	$\mathbf{78.88 \pm 2.85}$

Table 6.16: Accuracy of finding a minimum cost path (Min. Cost Acc.) and consistency in satisfying continuity constraints (Continuity) of the ResNet18 models with varying levels of supervision.

		ResNet18			
		Supervised		**NeuPSL Semi-Supervised**	
	Labeled	**Min. Cost Acc.**	**Continuity**	**Min. Cost Acc.**	**Continuity**
Pathfinding	1.00	80.12 ± 22.44	$\mathbf{84.80 \pm 17.11}$	$\mathbf{80.90 \pm 21.93}$	83.02 ± 20.09
	0.50	52.06 ± 14.77	61.86 ± 14.28	$\mathbf{59.84 \pm 16.51}$	$\mathbf{67.94 \pm 14.25}$
	0.10	2.60 ± 1.04	9.02 ± 1.90	$\mathbf{4.26 \pm 1.40}$	$\mathbf{35.18 \pm 3.40}$

training data varies over $\{1.0, 0.5, 0.1\}$.

The Bilevel learning algorithm is applied to train the NeSy-EBM neural components for the MNIST-Add$k$, Citeseer, and Cora datasets. The Energy learning algorithm is applied to train the NeSy-EBM neural components for the Visual-Sudoku and Pathfinding datasets.

Tables 6.14 to 6.16 report the average and standard deviation of the prediction performance of the supervised neural baseline and the semi-supervised neural component on the MNIST-Add$k$, Visual-Sudoku, Citeseer, Cora, and Pathfinding datasets. Across all datasets, as the proportion of unlabeled data increases, the semi-supervised neural component begins to outperform the supervised baseline. This behavior indicates that NeSy-EBMs are able to leverage the unlabeled training data by using the knowledge encoded in the NeuPSL rules. The benefit of utilizing symbolic knowledge is most evident in the lowest supervision settings, with the

NeuPSL semi-supervised ResNet18 model achieving over 20 percentage points of improvement when there is only 5% percent of the training labels in the MNIST-Add$k$ and Visual-Sudoku datasets. Surprisingly, this outcome is repeated in the Citeseer and Cora datasets, where the NeuPSL rules are not always adhered to. In other words, leveraging domain knowledge becomes more valuable for improving prediction performance as the amount of supervision decreases, even if the domain knowledge is not strictly accurate.

The Pathfinding results in Table 6.16 show there is not only a prediction performance gain achievable by making use of the symbolic component but also a reliability improvement. The reported Continuity metric measuring the consistency of the ResNet18 model in satisfying path continuity constraints is significantly improved when there is limited supervision and the model is trained with a NeSy-EBM loss. The NeuPSL semi-supervised ResNet18 model attains an over 25% improvement in path continuity consistency when only 10% of training labels are available. These results show NeSy-EBMs are valuable for aligning neural networks with desirable properties beyond accuracy.

Chapter 7

Limitations

In this chapter, I discuss the limitations of the NeSy-EBM framework, NeuPSL, and the empirical analysis. The NeSy-EBM framework is a high-level and general paradigm for NeSy. The value of the framework is that it provides a unifying theory for NeSy and a foundation for creating widely applicable modeling paradigms and learning algorithms. Progress on developing highly efficient NeSy inference algorithms, on the other hand, benefits from a perspective that considers the specific structure of the energy function and inference task. For instance, I show prediction in NeuPSL is a quadratic program, a property that is leveraged to create the dual BCD inference algorithm tailored for leveraging warm starts to realize learning runtime improvements. Similarly, the inference task of density estimation for NeSy systems such as semantic probabilistic layers [Ahmed et al., 2022a] is made highly efficient by levering constraints on the design of the energy function.

The taxonomy of NeSy modeling paradigms introduced in Section 3.2 is not exhaustive. For instance, it omits NeSy systems that integrate symbolic knowledge extraction from deep neural networks [Tran and d'Avila Garcez, 2018]. Moreover, I do not discuss DSVar, DSPar, and DSPot model combinations. I leave the exploration of utilizing multiple NeSy modeling paradigms to fuse neural components operating over multiple modalities for future work.

The four learning techniques proposed in this manuscript are presented with necessary assumptions on the energy function. For instance, direct gradient descent

can only be principally applied to minimize a NeSy-EBM loss at points where the energy function is twice differentiable with respect to the neural output and symbolic weights. Similarly, the bilevel technique is principled at points where the optimal value-function is differentiable with respect to the neural output and symbolic weights. I do not explore methods for extending the gradient descent and bilevel learning techniques to support NeSy-EBMs that do not satisfy all assumptions. One approach is to substitute the inference program with an approximation. The modular and stochastic policy optimization learning techniques require significantly fewer assumptions on the form of the energy function. However, these two techniques have their own limitations, which I discuss in their respective subsections.

The NeuPSL modeling framework, while expressive, does not currently support the implementation of every NeSy-EBM energy function and inference task. Specifically, NeuPSL can create energy functions defined as a weighted sum of potentials derived via arithmetic, logic, and Lukasiewicz real-logic semantics, as described in Chapter 5. NeuPSL does not support potentials constructed from other real-logic semantics. Further, NeuPSL is currently only designed to perform non-probabilistic inference tasks (e.g., prediction, ranking, and detection). This is due to the complexities of computing marginal distributions with the Gibbs partition function defined from the energy.

The empirical evaluations do not encompass every NeSy application, for instance, reasoning with noisy data. Furthermore, although my research advances the incorporation of commonsense reasoning and domain knowledge into LLMs for question answering, I have not extended the investigation to more complex reasoning tasks like summarization or explanation.

Chapter 8

Conclusion and Future Work

In this dissertation, I contributed to reaching four milestones that are necessary to achieve general and practical NeSy systems: *a mathematical framework*, *modeling paradigms*, *learning techniques*, and *a practical implementation*. I introduced a unifying mathematical framework for neural-symbolic (NeSy) reasoning with Neural-Symbolic Energy-Based Models (NeSy-EBMs). The NeSy-EBM framework is a foundation for NeSy and a bridge for adapting techniques from the broader machine learning literature to solve challenges in NeSy. I utilized the framework to create a principled taxonomy of NeSy modeling paradigms based on reasoning capabilities and a suite of learning losses and algorithms. Additionally, I introduced Neural Probabilistic Soft Logic (NeuPSL), an open-source and highly expressive implementation of NeSy-EBMs. NeuPSL supports the primary modeling paradigms and continuity properties required for efficient end-to-end neural and symbolic parameter learning.

I demonstrated that NeSy-EBMs provide a unifying view of NeSy by building a taxonomy of fundamental modeling paradigms. The modeling paradigms organize the strengths and limitations of NeSy systems and clarify architecture requirements for applications. NeSy-EBMs and the paradigms are valuable mechanisms for practitioners and researchers to understand the growing NeSy literature and design effective systems.

Further, NeSy-EBMs illuminate connections between NeSy and the broader machine learning community. For instance, I formalized a general categorization of

NeSy learning losses and the necessary assumptions for supporting direct gradient descent. Moreover, I leverage methods from reinforcement learning and bilevel optimization literature to work around the assumptions and design more practical and general algorithms.

The insights gained from creating the mathematical framework, the taxonomy of modeling paradigms, and the suite of learning techniques shaped the development of the NeuPSL NeSy modeling library. NeuPSL is built to support every modeling paradigm and learning technique I discuss. Moreover, I developed a new inference algorithm for NeuPSL that greatly improves learning runtimes. I demonstrated the effectiveness of NeuPSL in an empirical analysis. Specifically, I explored four real-world use cases of NeSy. I showed compelling results confirming NeSy-EBMs enhance neural network predictions, enforce constraints, improve label and data efficiency, and empower LLMs with consistent reasoning.

Several promising avenues for future research have emerged. A more extensive exploration into techniques for leveraging symbolic knowledge to fine-tune and adapt foundation models is a promising direction. The NeSy-EBM framework and my proposed learning techniques are a solid basis for building pipelines to fine-tune foundation models. Moreover, stochastic policy optimization for end-to-end NeSy learning has great potential due to its general applicability. Finally, contributing to the active area of research on overcoming the challenge of high-variance gradient estimates would be highly beneficial for improving NeSy learning.

Looking ahead, I foresee symbolic knowledge and reasoning remaining necessary for practical machine learning and AI systems. This conclusion is drawn from a number of observations. First, interpretable and or explainable decision making, which is inherently symbolic, is a requirement for many applications [Burkart and Huber, 2021]. Similarly, guardrails on AI systems are critical to ensure operation within a set of parameters and neural-symbolic integrations are an effective method for reliably controlling model behaviors [Dong et al., 2024]. Finally, there is a limit to human-generated training data Villalobos et al. [2024]. We therefore need to begin generating synthetic data or improving the data efficiency of our models and

algorithms, both approaches can leverage symbolic knowledge and reasoning. For instance, training on synthetic data that can be verified with symbolic systems, such as mathematics, logic, programming, and games, has shown great promise [Silver et al., 2017, Haluptzok et al., 2023, Yang et al., 2024, Trinh et al., 2024]. It is time we embrace the value of symbolic knowledge and reasoning to create practical AI.

Appendix A

Introduction

The appendix includes the following chapters: Extended Neural Probabilistic Soft Logic (Appendix B), and Extended Empirical Analysis (Appendix C).

Appendix B

Extended Neural Probabilistic Soft Logic

In this section, I expand on the smooth formulation of NeuPSL inference and provide proofs for the continuity results presented in Section 5.3.

B.1 Extended Smooth Formulation of Inference

Recall the primal formulation of NeuPSL inference restated below:

$$\arg\min_{\mathbf{y}\in\mathbb{R}^{n_\mathbf{y}}} \mathbf{w}_{sy}^T \mathbf{\Phi}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \quad \text{s.t. } \mathbf{y} \in \mathbf{\Omega}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \tag{B.1}$$

Importantly, note the structure of the deep hinge-loss potentials defining $\mathbf{\Phi}$:

$$\begin{aligned} &\phi_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &:= (\max\{\mathbf{a}_{\phi_k,y}^T \mathbf{y} + \mathbf{a}_{\phi_k,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_k,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_k}, 0\})^{p_k}. \end{aligned} \tag{B.2}$$

The LCQP NeuPSL inference formulation is defined using ordered index sets: $\mathbf{I}_S$ for the partitions of squared hinge potentials (indices k which for all $j \in t_k$ the exponent term $p_j = 2$) and $\mathbf{I}_L$ for the partitions of linear hinge potentials (indices k which for

all $j \in t_k$ the exponent term $p_j = 1$). With the index sets, define

$$\mathbf{W}_S := \begin{bmatrix} w_{\mathbf{I}_S[1]}\mathbf{I} & 0 & \cdots & 0 \\ 0 & w_{\mathbf{I}_S[2]}\mathbf{I} & & \\ \vdots & & \ddots & \end{bmatrix} \quad \text{and} \quad \mathbf{w}_L := \begin{bmatrix} w_{\mathbf{I}_L[1]}\mathbf{1} \\ w_{\mathbf{I}_L[2]}\mathbf{1} \\ \vdots \end{bmatrix} \tag{B.3}$$

Let $m_S := |\cup_{\mathbf{I}_S} t_k|$ and $m_L := |\cup_{\mathbf{I}_L} t_k|$, be the total number of squared and linear hinge potentials, respectively, and define slack variables $\mathbf{s}_S := [s_j]_{j=1}^{m_S}$ and $\mathbf{s}_L := [s_j]_{j=1}^{m_L}$ for each of the squared and linear hinge potentials, respectively. NeuPSL inference is equivalent to the following LCQP:

$$\min_{\mathbf{y} \in [0,1]^{n_y}, \mathbf{s}_S \in \mathbb{R}^{m_S}, \mathbf{s}_\mathbf{H} \in \mathbb{R}_+^{m_L}} \mathbf{s}_S^T \mathbf{W}_S \mathbf{s}_S + \mathbf{w}_L^T \mathbf{s}_L \tag{B.4a}$$

$$\text{s.t.} \quad \mathbf{a}_{c_i,\mathbf{y}}^T \mathbf{y} + \mathbf{a}_{c_i,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{c_i,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{c_i} \leq 0 \quad \forall\, i = 1, \ldots, q, \tag{B.4b}$$

$$\mathbf{a}_{\phi_j,\mathbf{y}}^T \mathbf{y} + \mathbf{a}_{\phi_j,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_j,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_j} - s_j \leq 0 \quad \forall j \in I_S \cup I_L. \tag{B.4c}$$

I ensure strong convexity by adding a square regularization with parameter ϵ to the objective. Let the bound constraints on $\mathbf{y}$ and $\mathbf{s}_L$ and linear inequalities in the LCQP be captured by the $(2 \cdot n_y + q + m_S + 2 \cdot m_L) \times (n_y + m_S + m_L)$ matrix $\mathbf{A}$ and $(2 \cdot n_y + q + m_S + 2 \cdot m_L)$ dimensional vector $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. More formally, $\mathbf{A} := [a_{ij}]$ where a_{ij} is the coefficient of a decision variable in the implicit and explicit

constraints in the formulation above:

$$a_{i,j} := \begin{cases} 0 & (i \leq q) \wedge (j \leq m_S + m_L) \\ \mathbf{a}_{c_i,\mathbf{y}}[j - (m_S + m_L)] & (i \leq q) \wedge (j > m_S + m_L) \\ 0 & (q < i \leq q + m_S + m_L) \wedge (j \leq m_S + m_L) \wedge (j \neq i - q) \\ -1 & (q < i \leq q + m_S + m_L) \wedge (j \leq m_S + m_L) \wedge (j = i - q) \\ \mathbf{a}_{\phi_{i-q},\mathbf{y}}[j - (m_S + m_L)] & (q < i \leq q + m_S + m_L) \wedge (j > m_S + m_L) \\ 0 & (q + m_S + m_L < i \leq q + m_S + 2 \cdot m_L + n_y) \\ & \wedge (j \neq i - (q + m_L)) \\ -1 & (q + m_S + m_L < i \leq q + m_S + 2 \cdot m_L + n_y) \\ & \wedge (j = i - (q + m_L)) \\ 0 & (q + m_S + 2 \cdot m_L + n_y < i \leq q + m_S + 2 \cdot m_L + 2 \cdot n_y) \\ & \wedge (j \neq i - (q + m_S + m_L)) \\ 1 & (q + m_S + 2 \cdot m_L + n_y < i \leq q + m_S + 2 \cdot m_L + 2 \cdot n_y) \\ & \wedge (j = i - (q + m_S + m_L)) \end{cases} . \tag{B.5}$$

Furthermore, $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = [b_i(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))]$ is the vector of constants corresponding to each constraint in the formulation above:

$$b_i(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \tag{B.6}$$

$$:= \begin{cases} \mathbf{a}_{c_i,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{c_i,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{c_i} & i \leq q \\ \mathbf{a}_{\phi_{i-q},\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_{i-q},\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_{i-q}} & q < i \leq q + m_S + m_L \\ 0 & q + m_S + m_L < i \\ & \quad \leq q + m_S + 2 \cdot m_L + n_y \\ -1 & q + m_S + 2 \cdot m_L + n_y < i \\ & \quad \leq q + m_S + 2 \cdot m_L + 2 \cdot n_y \end{cases}. \tag{B.7}$$

Note that $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ is a linear function of the neural network outputs, hence, if $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ is a smooth function of the neural parameters, then $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ is also smooth.

With this notation, the regularized inference problem is:

$$V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) := \min_{\mathbf{y}, \mathbf{s_S}, \mathbf{s_H}} \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix}^T \begin{bmatrix} \mathbf{W}_S + \epsilon I & 0 & 0 \\ 0 & \epsilon I & 0 \\ 0 & 0 & \epsilon I \end{bmatrix} \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{w}_L \\ 0 \end{bmatrix}^T \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix}$$
$$\text{s.t.} \quad \mathbf{A} \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix} + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0. \tag{B.8}$$

For ease of notation, let

$$D(\mathbf{w}_{sy}) := \begin{bmatrix} \mathbf{W}_S & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{c}(\mathbf{w}_{sy}) := \begin{bmatrix} 0 \\ \mathbf{w}_L \\ 0 \end{bmatrix}, \nu := \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix}. \tag{B.9}$$

Then the regularized primal LCQP MAP inference problem is concisely expressed as

$$\min_{\nu \in \mathbb{R}^{n_{\mathbf{y}}+m_S+m_L}} \nu^T(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}(\mathbf{w}_{sy})^T\nu \tag{B.10}$$
$$\text{s.t.} \quad \mathbf{A}\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0.$$

By Slater's constraint qualification, strong-duality holds when there is a feasible solution. In this case, an optimal solution to the dual problem yields an optimal solution to the primal problem. The Lagrange dual problem of (B.10) is

$$\begin{aligned}
&\arg\max_{\mu \geq 0} \min_{\nu \in \mathbb{R}^{n_{\mathbf{y}}+m_S+m_L}} \nu^T(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}(\mathbf{w}_{sy})^T\nu + \mu^T(\mathbf{A}\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\
&= \arg\max_{\mu \geq 0} -\frac{1}{4}\mu^T\mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{A}^T\mu \\
&\qquad - \frac{1}{2}(\mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{c}(\mathbf{w}_{sy}) - 2\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))^T\mu
\end{aligned} \tag{B.11}$$

where $\mu = [\mu_i]_{i=1}^{n_\mu}$ are the Lagrange dual variables. For later reference, denote the negative of the Lagrange dual function of MAP inference as:

$$\begin{aligned}
&h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\
&:= \frac{1}{4}\mu^T\mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{A}^T\mu + \frac{1}{2}(\mathbf{A}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}\mathbf{c}(\mathbf{w}_{sy}) - 2\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))^T\mu.
\end{aligned} \tag{B.12}$$

The dual LCQP has more decision variables but is only over non-negativity constraints rather than the complex polyhedron feasible set. The dual-to-primal variable translation is:

$$\nu = -\frac{1}{2}(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})^{-1}(\mathbf{A}^T\mu + \mathbf{c}(\mathbf{w}_{sy})) \tag{B.13}$$

As $(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})$ is diagonal, it is easy to invert and hence it is practical to work in the dual space to obtain a solution to the primal problem.

B.2 Extended Continuity of Inference

I now provide background on sensitivity analysis that I then apply in my proofs on the continuity properties of NeuPSL inference.

B.2.1 Background

Theorem 14 (Boyd and Vandenberghe [2004] p. 81)

If for each $\mathbf{y} \in \mathcal{A}$, $f(\mathbf{x}, \mathbf{y})$ *is convex in* $\mathbf{x}$ *then the function*

$$g(\mathbf{x}) := \sup_{\mathbf{y} \in \mathcal{A}} f(\mathbf{x}, \mathbf{y}) \tag{B.14}$$

is convex in $\mathbf{x}$.

Theorem 15 (Boyd and Vandenberghe [2004] p. 81)

If for each $\mathbf{y} \in \mathcal{A}$, $f(\mathbf{x}, \mathbf{y})$ *is concave in* $\mathbf{x}$ *then the function*

$$g(\mathbf{x}) := \inf_{\mathbf{y} \in \mathcal{A}} f(\mathbf{x}, \mathbf{y}) \tag{B.15}$$

is concave in $\mathbf{x}$.

Definition 16 (Convex Subgradient: Boyd and Vandenberghe [2004] and Shalev-Shwartz [2012])

Consider a convex function $f : \mathbb{R}^n \to [-\infty, \infty]$ *and a point* $\overline{\mathbf{x}}$ *with* $f(\overline{\mathbf{x}})$ *finite. For a vector* $\mathbf{v} \in \mathbf{R}^n$, *one says that* $\mathbf{v}$ *is a (convex) subgradient of* f *at* $\overline{\mathbf{x}}$, *written* $\mathbf{v} \in \partial f(\overline{\mathbf{x}})$, *iff*

$$f(\mathbf{x}) \geq f(\overline{\mathbf{x}}) + < \mathbf{v}, \mathbf{x} - \overline{\mathbf{x}} >, \quad \forall \mathbf{x} \in \mathbf{R}^n. \tag{B.16}$$

Definition 17 (Closedness: Bertsekas (2009))

If the epigraph of a function $f : \mathbb{R}^n \to [-\infty, \infty]$ *is a closed set, then* f *is a closed function.*

Definition 18 (Lower Semicontinuity: Bertsekas [2009])

The function $f : \mathbb{R}^n \to [-\infty, \infty]$ is lower semicontinuous (lsc) at a point $\overline{\mathbf{x}} \in \mathbb{R}^n$ if

$$f(\overline{\mathbf{x}}) \leq \liminf_{k \to \infty} f(\mathbf{x}_k), \tag{B.17}$$

for every sequence $\{\mathbf{x}_k\} \subset \mathbb{R}^n$ with $\mathbf{x}_k \to \overline{\mathbf{x}}$. f is lsc if it is lsc at each $\overline{\mathbf{x}}$ in its domain.

Theorem 19 (Closedness and Semicontinuity: Bertsekas [2009] Proposition 1.1.2.) *For a function $f : \mathbb{R}^n \to [-\infty, \infty]$, the following are equivalent:*

1. *The level set $V_\gamma = \{\mathbf{x} \,|\, f(\mathbf{x}) \leq \gamma\}$ is closed for every scalar γ.*

2. *f is lsc.*

3. *f is closed.*

The following definition and theorem are from Rockafellar and Wets [1997] and they generalize the notion of subgradients to non-convex functions and the chain rule of differentiation, respectively. For complete statements see Rockafellar and Wets [1997] Rockafellar and Wets [1997].

Definition 20 (Regular Subgradient: Rockafellar and Wets [1997] Definition 8.3) *Consider a function $f : \mathbb{R}^n \to [-\infty, \infty]$ and a point $\overline{\mathbf{x}}$ with $f(\overline{\mathbf{x}})$ finite. For a vector $\mathbf{v} \in \mathbf{R}^n$, one says that $\mathbf{v}$ is a regular subgradient of f at $\overline{\mathbf{x}}$, written $\mathbf{v} \in \hat{\partial} f(\overline{\mathbf{x}})$, iff*

$$f(\mathbf{x}) \geq f(\overline{\mathbf{x}}) + \langle \mathbf{v}, \mathbf{x} - \overline{\mathbf{x}} \rangle + o(\mathbf{x} - \overline{\mathbf{x}}), \quad \forall \mathbf{x} \in \mathbf{R}^n, \tag{B.18}$$

where the $o(t)$ notation indicates a term with the property that

$$\lim_{t \to 0} \frac{o(t)}{t} = 0. \tag{B.19}$$

The relation of the regular subgradient defined above and the more familiar convex subgradient is the addition of the $o(\mathbf{x} - \overline{\mathbf{x}})$ term. Evidently, a convex subgradient is a regular subgradient.

Theorem 21 (Chain Rule for Regular Subgradients: Rockafellar and Wets [1997] Theorem 10.6)

Suppose $f(\mathbf{x}) = g(F(\mathbf{x}))$ *for a proper, lsc function* $g : \mathbb{R}^m \to [-\infty, \infty]$ *and a smooth mapping* $F : \mathbb{R}^n \to \mathbb{R}^m$. *Then at any point* $\overline{\mathbf{x}} \in dom\, f = F^{-1}(dom\, g)$ *one has*

$$\hat{\partial} f(\overline{\mathbf{x}}) \supset \nabla F(\overline{\mathbf{x}})^T \hat{\partial} g(F(\overline{\mathbf{x}})), \tag{B.20}$$

where $\nabla F(\overline{\mathbf{x}})^T$ *is the Jacobian of* F *at* $\overline{\mathbf{x}}$.

Theorem 22 (Danskin's Theorem: Danskin [1966] and Bertsekas [1971] Proposition A.22)

Suppose $\mathcal{Z} \subseteq \mathbb{R}^m$ *is a compact set and* $g(\mathbf{x}, \mathbf{z}) : \mathbb{R}^n \times \mathcal{Z} \to (-\infty, \infty]$ *is a function. Suppose* $g(\cdot, \mathbf{z}) : \mathbb{R}^n \to \mathbb{R}$ *is closed proper convex function for every* $\mathbf{z} \in \mathcal{Z}$. *Further, define the function* $f : \mathbb{R}^n \to \mathbb{R}$ *such that*

$$f(\mathbf{x}) := \max_{\mathbf{z} \in \mathcal{Z}} g(\mathbf{x}, \mathbf{z}).$$

Suppose f *is finite somewhere. Moreover, let* $\mathcal{X} := int(domf)$, *i.e., the interior of the set of points in* $\mathbb{R}^n$ *such that* f *is finite. Suppose* g *is continuous on* $\mathcal{X} \times \mathcal{Z}$. *Further, define the set of maximizing points of* $g(\mathbf{x}, \cdot)$ *for each* $\mathbf{x}$

$$Z(\mathbf{x}) = \arg\max_{\mathbf{z} \in \mathcal{Z}} g(\mathbf{x}, \mathbf{z}).$$

Then the following properties of f *hold.*

1. *The function* $f(\mathbf{x})$ *is a closed proper convex function.*
2. *For every* $\mathbf{x} \in \mathcal{X}$,

$$\partial f(\mathbf{x}) = conv\left\{\partial_{\mathbf{x}} g(\mathbf{x}, \mathbf{z}) \,|\, \mathbf{z} \in Z(\mathbf{x})\right\}. \tag{B.21}$$

Corollary 23

Assume the conditions for Danskin's Theorem above hold. For every $\mathbf{x} \in \mathcal{X}$, *if* $Z(\mathbf{x})$ *consists of a unique point, call it* $\mathbf{z}^*$, *and* $g(\cdot, \mathbf{z}^*)$ *is differentiable at* $\mathbf{x}$, *then* $f(\cdot)$ *is*

differentiable at $\mathbf{x}$*, and*

$$\nabla f(\mathbf{x}) := \nabla_{\mathbf{x}} g(\mathbf{x}, \mathbf{z}^*). \tag{B.22}$$

Theorem 24 (Bonnans and Shapiro [1998] Theorem 4.2, Rockafellar [1974] p. 41) *Let* $\mathbf{X}$ *and* $\mathbf{U}$ *be Banach spaces. Let* $\mathbf{K}$ *be a closed convex cone in the Banach space* $\mathbf{U}$*. Let* $G : \mathbf{X} \to \mathbf{U}$ *be a convex mapping with respect to the cone* $\mathbf{C} := -\mathbf{K}$ *and* $f : \mathbf{X} \to (-\infty, \infty]$ *be a (possibly infinite-valued) convex function. Consider the following convex program and its optimal value function:*

$$\begin{aligned} v_P(\mathbf{u}) := \min_{\mathbf{x} \in \mathbf{X}} \ & f(\mathbf{x}) && (P) \\ s.t. \quad & G(\mathbf{x}) + \mathbf{u} \in \mathbf{K}. \end{aligned}$$

Moreover, consider the (Lagrangian) dual of the program:

$$v_D(\mathbf{u}) := \max_{\lambda \in \mathbf{K}^-} \min_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}) + \lambda^T (G(\mathbf{x}) + \mathbf{u}) \qquad (D)$$

Suppose $v_P(\mathbf{0})$ *is finite. Further, suppose the feasible set of the program is nonempty for all* $\mathbf{u}$ *in a neighborhood of* $\mathbf{0}$*, i.e.,*

$$\mathbf{0} \in int\{G(\mathbf{X}) - \mathbf{K}\}. \tag{B.23}$$

Then,

1. *There is no primal dual gap at* $u = 0$*, i.e.,* $v_P(0) = v_D(0)$*.*
2. *The set,* Λ_0*, of optimal solutions to the dual problem with* $\mathbf{u} = 0$ *is non-empty and bounded.*
3. *The optimal value function* $v_P(\mathbf{u})$ *is continuous at* $\mathbf{u} = 0$ *and* $\partial v_P(\mathbf{0}) = \Lambda_0$*.*

Theorem 25 (Bonnans and Shapiro [2000] Proposition 4.3.2)

Consider two optimization problems over a non-empty feasible set $\mathbf{\Omega}$*:*

$$\min_{\mathbf{x}\in\mathbf{\Omega}} f_1(\mathbf{x}) \qquad \textit{and} \qquad \min_{\mathbf{x}\in\mathbf{\Omega}} f_2(\mathbf{x}) \tag{B.24}$$

where $f_1, f_2 : \mathcal{X} \to \mathbb{R}$*. Suppose* f_1 *has a non-empty set* $\mathbf{S}$ *of optimal solutions over* $\mathbf{\Omega}$*. Suppose the second order growth condition holds for* $\mathbf{S}$*, i.e., there exists a neighborhood* $\mathcal{N}$ *of* $\mathbf{S}$ *and a constant* $\alpha > 0$ *such that*

$$f_1(\mathbf{x}) \geq f_1(\mathbf{S}) + \alpha(dist(\mathbf{x}, \mathbf{S}))^2, \qquad \forall \mathbf{x} \in \mathbf{\Omega} \cap \mathcal{N}, \tag{B.25}$$

where $f_1(\mathbf{S}) := inf_{\mathbf{x}\in\mathbf{\Omega}} f_1(\mathbf{x})$*. Define the difference function:*

$$\Delta(\mathbf{x}) := f_2(\mathbf{x}) - f_1(\mathbf{x}). \tag{B.26}$$

Suppose $\Delta(\mathbf{x})$ *is L-Lipschitz continuous on* $\mathbf{\Omega} \cap \mathcal{N}$*. Let* $\mathbf{x}^* \in \mathcal{N}$ *be an* δ*-solution to the problem of minimizing* $f_2(\mathbf{x})$ *over* $\mathbf{\Omega}$*. Then*

$$dist(\mathbf{x}^*, \mathbf{S}) \leq \frac{L}{\alpha} + \sqrt{\frac{\delta}{\alpha}}. \tag{B.27}$$

B.2.2 Proofs

I provide proofs of theorems presented in the main paper and restate them here for completeness.

Theorem 10

Suppose for any setting of $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$ *there is a feasible solution to NeuPSL inference* (5.9)*. Further, suppose* $\epsilon > 0$*,* $\mathbf{w}_{sy} \in \mathbb{R}^r_+$*, and* $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$*. Then:*

- *The minimizer of* (5.9)*,* $\mathbf{y}^*(\mathbf{w}_{sy}, \mathbf{w}_{nn})$*, is a* $O(1/\epsilon)$ *Lipschitz continuous function of* $\mathbf{w}_{sy}$*.*
- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$*, is concave over* $\mathbf{w}_{sy}$*.*
- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is convex over* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$*.*

- $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is differentiable with respect to* $\mathbf{w}_{sy}$*. Moreover,*

$$\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy},\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) = \mathbf{\Phi}(\mathbf{y}^*(\mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})).$$

  Furthermore, $\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is Lipschitz continuous over* $\mathbf{w}_{sy}$*.*

- *If there is a feasible point* ν *strictly satisfying the* $i'th$ *inequality constraint of* (5.9)*, i.e.,* $\mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] < 0$*, then* $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is subdifferentiable with respect to the* $i'th$ *constraint constant* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i]$*. Moreover,*

$$\begin{aligned}&\partial_{\mathbf{b}[i]} V(\mathbf{w}_{sy},\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\ &\qquad = \{\mu^*[i] \,|\, \mu^* \in \underset{\mu \in \mathbb{R}^{2\cdot(n_{\mathbf{y}}+m)+q}_{\geq 0}}{\arg\min} h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))\}.\end{aligned}$$

  Furthermore, if $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ *is a smooth function of* $\mathbf{w}_{nn}$*, then* $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ *is a smooth function of* $\mathbf{w}_{nn}$*. Additionally, the set of regular subgradients of* $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ *is:*

$$\begin{aligned}&\hat{\partial}_{\mathbf{w}_{nn}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\ &\qquad \supset \nabla_{\mathbf{w}_{nn}} \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))^T \partial_{\mathbf{b}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))).\end{aligned} \tag{B.28}$$

Proof of Theorem 10*.* I first show the minimizer of the LCQP formulation of NeuPSL inference, ν^*, with $\epsilon > 0$, $\mathbf{w}_{sy} \in \mathbb{R}^r_+$, and $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$ is a Lipschitz continuous function of $\mathbf{w}_{sy}$. Suppose $\epsilon > 0$ and $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$ is given. To show continuity over $\mathbf{w}_{sy} \in \mathbb{R}^r_+$, first note the matrix $(\mathbf{D} + \epsilon\mathbf{I})$ is positive definite and the primal inference problem (5.10) is an ϵ-strongly convex LCQP with a unique minimizer denoted by $\nu^*(\mathbf{w}_{sy}, \mathbf{w}_{nn})$. I leverage the Lipschitz stability result for optimal values of constrained problems from Bonnans and Shapiro [2000] and presented here in Theorem 25. Define the primal objective as an explicit function of the weights:

$$f(\nu, \mathbf{w}_{sy}, \mathbf{w}_{nn}) := \nu^T(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}^T(\mathbf{w}_{sy})\nu \tag{B.29}$$

Note that the solution $\nu^* = \begin{bmatrix} \mathbf{s}_S^* \\ \mathbf{s}_L^* \\ \mathbf{y}^* \end{bmatrix}$ will always be bounded, since from (B.4c), for all $j \in I_S \cup I_L$,

$$0 \leq s_j^* = \max(\mathbf{a}_{\phi_k,y}^T \mathbf{y}^* + \mathbf{a}_{\phi_k,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_k,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_k}, 0) \tag{B.30}$$

$$\leq \|\mathbf{a}_{\phi_k,y}\| + |\mathbf{a}_{\phi_k,\mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_k,\mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_k}|. \tag{B.31}$$

Thus, setting these trivial upper bounds for s_j will not change the solution of the problem. I can henceforth consider the problem in a bounded domain $\|\nu\| \leq C$ where C does not depend on $\mathbf{w}$'s.

Let $\mathbf{w}_{1,sy}, \mathbf{w}_{2,sy} \in \mathbb{R}_+^r$ and $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ be arbitrary. As $\epsilon > 0$, $f(\nu, \mathbf{w}_{1,sy}, \mathbf{w}_{nn})$ is strongly convex in ν and it therefore satisfies the second-order growth condition in ν. Define the difference function:

$$\Delta_{\mathbf{w}_{sy}}(\nu) := f(\nu, \mathbf{w}_{2,sy}, \mathbf{w}_{nn}) - f(\nu, \mathbf{w}_{1,sy}, \mathbf{w}_{nn}) \tag{B.32}$$

$$= \nu^T(\mathbf{D}(\mathbf{w}_{2,sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}^T(\mathbf{w}_{2,sy})\nu - \left(\nu^T(\mathbf{D}(\mathbf{w}_{1,sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}^T(\mathbf{w}_{1,sy})\nu\right) \tag{B.33}$$

$$= \nu^T(\mathbf{D}(\mathbf{w}_{2,sy}) - \mathbf{D}(\mathbf{w}_{1,sy}))\nu + (\mathbf{c}(\mathbf{w}_{2,sy}) - \mathbf{c}(\mathbf{w}_{1,sy}))^T\nu. \tag{B.34}$$

The difference function $\Delta_{\mathbf{w}_{sy}}(\nu)$ over $\mathcal{N}$ has a finitely bounded gradient:

$$\|\nabla\Delta_{\mathbf{w}_{sy}}(\nu)\|_2 = \left\|2(\mathbf{D}(\mathbf{w}_{2,sy}) - \mathbf{D}(\mathbf{w}_{1,sy}))\nu + \mathbf{c}(\mathbf{w}_{2,sy}) - \mathbf{c}(\mathbf{w}_{1,sy})\right\|_2 \tag{B.35}$$

$$\leq \|\mathbf{c}(\mathbf{w}_{2,sy}) - \mathbf{c}(\mathbf{w}_{1,sy})\|_2 + 2\|(\mathbf{D}(\mathbf{w}_{2,sy}) - \mathbf{D}(\mathbf{w}_{1,sy}))\nu\|_2 \tag{B.36}$$

$$\leq \|\mathbf{w}_{2,sy} - \mathbf{w}_{1,sy}\|_2 + 2\|\mathbf{w}_{2,sy} - \mathbf{w}_{1,sy}\|_2 \, \|\nu\|_2 \tag{B.37}$$

$$\leq \|\mathbf{w}_{2,sy} - \mathbf{w}_{1,sy}\|_2(1 + 2C) =: L_{\mathcal{N}}(\mathbf{w}_{1,sy}, \mathbf{w}_{2,sy}). \tag{B.38}$$

Thus, the distance function, $\Delta_{\mathbf{w}_{sy}}(\nu)$ is $L_{\mathcal{N}}(\mathbf{w}_{1,sy}, \mathbf{w}_{2,sy})$-Lipschitz continuous over $\mathcal{N}$. Therefore, by Bonnans and Shapiro [2000] (Theorem 25), the distance between

$\nu^*(\mathbf{w}_{1,sy}, \mathbf{w}_{nn})$ and $\nu^*(\mathbf{w}_{2,sy}, \mathbf{w}_{nn})$ is bounded above:

$$\|\nu^*(\mathbf{w}_{2,sy}, \mathbf{w}_{nn}) - \nu^*(\mathbf{w}_{1,sy}, \mathbf{w}_{nn})\|_2 \leq \frac{L_{\mathcal{N}}(\mathbf{w}_{1,sy}, \mathbf{w}_{2,sy})}{\epsilon} = \frac{(1+2C)}{\epsilon}\|\mathbf{w}_{2,sy} - \mathbf{w}_{1,sy}\|_2. \tag{B.39}$$

Therefore, the function $\nu^*(\mathbf{w}_{sy}, \mathbf{w}_{nn})$ is $O(1/\epsilon)$-Lipschitz continuous in $\mathbf{w}_{sy}$ for any $\mathbf{w}_{nn}$.

Next, I prove curvature properties of the value-function with respect to the weights. Observe NeuPSL inference is an infimum over a set of functions that are concave (affine) in $\mathbf{w}_{sy}$. Therefore, by Theorem 15, $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is concave in $\mathbf{w}_{sy}$.

I use a similar argument to show $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is convex in the constraint constants, $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. Assuming for any setting of the neural weights, $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$, there is a feasible solution to the NeuPSL inference problem, then (5.9) satisfies the conditions for Slater's constraint qualification. Therefore, strong duality holds, i.e., $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is equal to the optimal value of the dual inference problem (B.11). Observe that the dual NeuPSL inference problem is a supremum over a set of functions convex (affine) in $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. Therefore, by Theorem 14, $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is convex in $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$.

I can additionally prove convexity in $\mathbf{b}$ from first principles. For simplicity, fix other parameters, and write the objective and the value function as $Q(\nu)$ and $V(\mathbf{b})$, respectively. Let us first consider the domain where the optimization is bounded and the optimal solution exists. Given $\mathbf{b}_1$ and $\mathbf{b}_2$, let the corresponding optimal solutions of (B.10) parameterized by $\mathbf{b}_1$ and $\mathbf{b}_2$ be ν_1 and ν_2. Take any $\alpha \in [0, 1]$, note that $\alpha\nu_1 + (1-\alpha)\nu_2$ is feasible for the optimization problem parameterized by $\mathbf{b} = \alpha\mathbf{b}_1 + (1-\alpha)\mathbf{b}_2$. Because I take the inf over all νs, the optimal ν for this $\mathbf{b}$ might be even smaller. Thus, (for convex quadratic objective Q)

$$\begin{aligned} V(\alpha\mathbf{b}_1 + (1-\alpha)\mathbf{b}_2) &\leq Q(\alpha\nu_1 + (1-\alpha)\nu_2) \\ &\leq \alpha Q(\nu_1) + (1-\alpha)Q(\nu_2) \\ &= \alpha V(\mathbf{b}_1) + (1-\alpha)V(\mathbf{b}_2), \end{aligned} \tag{B.40}$$

which shows that V is convex in $\mathbf{b}$. To establish the convexity when $V(\mathbf{b})$ takes extended real-values ($\mathbb{R} \cup \{-\infty\}$) to allow for unbounded optimization problems, it suffices to consider sequences $\{\nu_i^k\}_{k=1}^{\infty}$ for $\mathbf{b}_i$ ($i = 1, 2$, $\mathbf{b}_1 \neq \mathbf{b}_2$) as follows:

(1) If $V(\mathbf{b}_i)$ is finite, let $\nu_i^k = \nu_i$ for all k, where ν_i is the optimal solution.

(2) If $V(\mathbf{b}_i) = -\infty$, there exists sequence $\{\nu_i^k\}_{k=1}^{\infty}$ such that $Q(\nu_i^k) \to -\infty$ as $k \to \infty$.

Now, for any $0 < \alpha < 1$, observe:

Case 1: Both $V(\mathbf{b}_1)$ and $V(\mathbf{b}_2)$ are finite. I can reuse the argument above.

Case 2: At least one of $V(\mathbf{b}_1)$ and $V(\mathbf{b}_2)$ is $-\infty$. By convexity of Q, $Q(\alpha\nu_1^k + (1-\alpha)\nu_2^k) \leq \alpha Q(\nu_1^k) + (1-\alpha)Q(\nu_2^k)$. Therefore, I have $Q(\alpha\nu_1^k + (1-\alpha)\nu_2^k) \to -\infty$ as $k \to \infty$ when $0 < \alpha < 1$. Note that for all k, $\alpha\nu_1^k + (1-\alpha)\nu_2^k$ is feasible for the optimization problem parameterized by $\mathbf{b} = \alpha\mathbf{b}_1 + (1-\alpha)\mathbf{b}_2$. It follows that $V(\alpha\mathbf{b}_1 + (1-\alpha)\mathbf{b}_2) = -\infty$.

Therefore, convexity holds when $V(\mathbf{b})$ takes extended real-values ($\mathbb{R} \cup \{-\infty\}$).

Next, I prove (sub)differentiability properties of the value-function. Suppose $\epsilon > 0$. First, I show the optimal value function, $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$, is differentiable with respect to the symbolic weights. Then, I show subdifferentiability properties of the optimal value function with respect to the constraint constants. Finally, I apply the Lipschitz continuity of the minimzer result to show the gradient of the optimal value function is Lipschitz continuous with respect to $\mathbf{w}_{sy}$.

Starting with differentiability with respect to the symbolic weights, $\mathbf{w}_{sy}$, note, the optimal value function of the regularized LCQP formulation of NeuPSL inference, (5.9), is equivalently expressed as the following maximization over a continuous function in the primal target variables, $\mathbf{y}$, the slack variables, $\mathbf{s}_S$ and $\mathbf{s}_L$, and the

symbolic weights, $\mathbf{w}_{sy}$:

$$
\begin{aligned}
& V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \qquad \text{(B.41)} \\
&= -\left(\max_{\mathbf{y}, \mathbf{s_H}, \mathbf{s_L}} -\Big(\begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix}^T \begin{bmatrix} \mathbf{W}_S + \epsilon I & 0 & 0 \\ 0 & \epsilon I & 0 \\ 0 & 0 & \epsilon I \end{bmatrix} \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{w}_L \\ 0 \end{bmatrix}^T \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix} \Big) \right) \\
&\quad \text{s.t.} \quad \mathbf{A} \begin{bmatrix} \mathbf{s}_S \\ \mathbf{s}_L \\ \mathbf{y} \end{bmatrix} + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) \leq 0,
\end{aligned}
$$

where the matrix $\mathbf{W}_s$ and vector $\mathbf{w}_L$ are functions of the symbolic parameters $\mathbf{w}_{sy}$ as defined in (B.3). Moreover, the objective above is and convex (affine) in $\mathbf{w}_{sy}$. Additionally, note that the decision variables can be constrained to a compact domain without breaking the equivalence of the formulation. Specifically, the target variables are constrained to the box $[0, 1]^{\mathbf{n}_y}$, while the slack variables are nonnegative and have a trivial upper bound derived from (B.4c):,

$$
\begin{aligned}
0 \leq s_j^* &= \max(\mathbf{a}_{\phi_k, y}^T \mathbf{y}^* + \mathbf{a}_{\phi_k, \mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_k, \mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_k}, 0) \\
&\leq \|\mathbf{a}_{\phi_k, y}\| + |\mathbf{a}_{\phi_k, \mathbf{x}_{sy}}^T \mathbf{x}_{sy} + \mathbf{a}_{\phi_k, \mathbf{g}_{nn}}^T \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) + b_{\phi_k}|, \qquad \text{(B.42)}
\end{aligned}
$$

for all $j \in I_S \cup I_L$. Therefore, the negative optimal value function satisfies the conditions for Danskin's theorem Danskin [1966] (stated in Appendix B.2.1). Moreover, as there is a single unique solution to the inference problem when $\epsilon > 0$, and the quadratic objective in (5.9) is differentiable for all $\mathbf{w}_{sy} \in \mathbb{R}_+^r$, I can apply Corollary 23. The optimal value function is therefore concave and differentiable with respect to the symbolic weights with

$$
\nabla_{\mathbf{w}_{sy}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = \mathbf{\Phi}(\mathbf{y}^*, \mathbf{x_{sy}}, \mathbf{g_{nn}}(\mathbf{x_{nn}}, \mathbf{w_{nn}})). \qquad \text{(B.43)}
$$

Next, I show subdifferentiability of the optimal value-function with respect to the constraint constants, $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. Suppose at a setting of the neural

weights $\mathbf{w}_{nn} \in \mathbb{R}^{n_g}$ there is a feasible point ν for the NeuPSL inference problem. Moreover, suppose ν strictly satisfies the $i'th$ inequality constraint of (5.9), i.e., $\mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] < 0$. Observe that the following strongly convex conic program is equivalent to the LCQP formulation of NeuPSL inference, (5.9):

$$\begin{aligned} \min_{\nu \in \mathbb{R}^{n_{\mathbf{y}}+m_S+m_L}} \; & \nu^T(\mathbf{D}(\mathbf{w}_{sy}) + \epsilon\mathbf{I})\nu + \mathbf{c}(\mathbf{w}_{sy})^T\nu + P_{\Omega\setminus i}(\nu) \\ \text{s.t.} \quad & \mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] \in \mathbb{R}_{\leq 0}, \end{aligned} \tag{B.44}$$

where $P_{\Omega\setminus i}(\nu) : \mathbb{R}^{n_{\mathbf{y}}+m_S+m_L} \to \{0, \infty\}$ is the indicator function identifying feasibility w.r.t. all the constraints of the LCQP formulation of NeuPSL inference in (5.9) except the $i'th$ constraint: $\mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] \leq 0$. In other words, in the conic formulation above only the $i'th$ constraint is explicit. Note that $\mathbb{R}_{\leq 0}$ is a closed convex cone in $\mathbb{R}$. Moreover, both the objective in the program and the mapping $G(\nu) := \mathbf{A}[i]\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i]$ are convex. Lastly, note the constraint qualification (B.23) is similar to Slater's condition in the case of (B.44) which is satisfied by the supposition there exists a feasible ν that strictly satisfies the $i'th$ inequality constraint of (5.9). Therefore, (B.44) satisfies the conditions of Theorem 24. Thus, the value function is continuous in the constraint constant $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i]$ at $\mathbf{w}_{nn}$ and

$$\begin{aligned} &\partial_{\mathbf{b}[i]} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\ &\quad = \{\mu^*[i] \,|\, \mu^* \in \operatorname*{arg\,min}_{\mu \in \mathbb{R}^{2 \cdot n_{\mathbf{y}}+m+q}_{\geq 0}} h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))\}. \end{aligned} \tag{B.45}$$

Moreover, when $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ is a smooth function of the neural weights $\mathbf{w}_{nn}$, then I can apply the chain rule for regular subgradients, Theorem 21, to get

$$\begin{aligned} &\hat{\partial}_{\mathbf{w}_{nn}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &\quad \supset \nabla \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})^T \partial_{\mathbf{b}} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \end{aligned} \tag{B.46}$$

To prove the optimal value function is Lipschitz smooth over $\mathbf{w}_{sy}$, it is equivalent to show it is continuously differentiable and that all gradients have bounded magnitude.

To show the value function is continuously differentiable, I first apply the result asserting the minimizer is unique and a continuous function of the symbolic parameters $\mathbf{w}_{sy}$. Therefore, the optimal value function gradient is a composition of continuous functions, hence continuous in $\mathbf{w}_{sy}$. The fact that the value function has a bounded gradient magnitude follows from the fact that the decision variables $\mathbf{y}$ have a compact domain over which the gradient is finite; hence a trivial and finite upper bound exists on the gradient magnitude. □

Appendix C

Extended Empirical Analysis

In this section, I provide additional information on the empirical analysis. The implementation of models and training processes can be found at `https://github.com/linqs/dickens-arxiv24`.

C.1 Hardware

All timing experiments were performed on an Ubuntu 22.04.1 Linux machine with Intel Xeon Processor E5-2630 v4 at 3.10GHz and 128 GB of RAM.

C.2 Extended Inference Runtime

This section details the hyperparameter settings and search process for the inference runtime experiments in Section 6.2.1. The GD, ADMM, and D-BCD algorithms are stopped when the L_∞ norm of the primal variable change between iterates is less than 0.001. For the D-BCD algorithms, the regularization parameter resulting in the fastest runtime and yielding a prediction performance within a standard error of the best is used. The default Gurobi optimizer hyperparameters are used. Table C.1 reports the range of hyperparameters searched over and the final values.

Table C.2 reports the average and standard deviation of the inference runtime for Gurobi, GD, ADMM, and D-BCD algorithms on 4 of the datasets from Table 6.1. As in the main paper, the D-BCD algorithms are competitive with ADMM, the

Table C.1: Hyperparameter ranges and final values for the inference runtime experiments.

Dataset	Parameter	Range	Final Value
CreateDebate	**ADMM Step Length**	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	0.1
4Forums	**ADMM Step Length**	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	0.1
Epinions	**GD Step Length**	$\{10.0, 1.0, 0.1, 0.01, 0.001\}$	0.01
	ADMM Step Length	$\{10.0, 1.0, 0.1, 0.01\}$	0.1
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	0.1
Citeseer	**GD Step Length**	$\{10.0, 1.0, 0.1, 0.01, 0.001\}$	0.1
	ADMM Step Length	$\{10.0, 1.0, 0.1, 0.01\}$	10.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	10.0
Cora	**GD Step Length**	$\{10.0, 1.0, 0.1, 0.01, 0.001\}$	0.1
	ADMM Step Length	$\{10.0, 1.0, 0.1, 0.01\}$	10.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	10.0
DDI	**ADMM Step Length**	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	10.0
Yelp	**GD Step Length**	$\{10.0, 1.0, 0.1, 0.01, 0.001\}$	0.001
	ADMM Step Length	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01\}$	0.1
MNIST-Add1	**ADMM Step Length**	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01, 0.001\}$	0.001
MNIST-Add2	**ADMM Step Length**	$\{10.0, 1.0, 0.1, 0.01\}$	1.0
	LCQP Regularization	$\{100, 10, 1, 0.1, 0.01, 0.001\}$	0.001

Table C.2: Inference time in seconds for each inference optimization technique.

	Gurobi	**GD**	**ADMM**	**CC D-BCD**	**LF D-BCD**
Epinions	0.46 ± 0.01	34.63 ± 0.33	0.36 ± 0.041	1.84 ± 0.4	$\mathbf{0.26 \pm 0.04}$
Citeseer	0.66 ± 0.08	47.17 ± 0.61	0.63 ± 0.07	1.36 ± 0.24	$\mathbf{0.49 \pm 0.08}$
Cora	$\mathbf{0.71 \pm 0.08}$	48.66 ± 1.24	$\mathbf{0.71 \pm 0.07}$	6.46 ± 3.5	0.79 ± 0.19
Yelp	7.38 ± 0.20	$6,961 \pm 46$	$\mathbf{6.37 \pm 1.19}$	48.44 ± 3.82	7.58 ± 0.48

current state of the art optimizer for NeuPSL inference. Moreover, the LF D-BCD algorithm is also competitive with Gurobi for a single round of inference.

C.2.1 Extended Learning Runtime

This section provides details of the hyperparameter settings for the learning runtime experiments in Section 6.3.1. For both the SP and MSE learning losses, a negative log regularization with coefficient $1.0e-3$ on the symbolic weights is added to the learning loss. For ADMM inference, the same steplength from the inference runtime experiment is used. Similarly, for D-BCD inference on both learning losses, the same regularization parameter from the inference runtime experiment is used. For the MNIST-Add experiments, I use the regularization parameter $\epsilon = 1.0e-3$ and ADMM steplength 1.0 as the values were found to achieve the highest final validation prediction performance.

Mirror descent is applied to learn the symbolic weights for both SP and MSE losses. The mirror descent steplength is set to a default value of $1.0e-3$ for all datasets. The Adam steplength for the neural component of the MNIST-Add models is set to a default value of $1.0e-3$.

For the bilevel learning algorithm minimizing the MSE loss, I set the initial squared penalty parameter to a default value of 2.0. Moreover, for all but the MNIST-Add datasets, I set the Moreau parameter to 0.01, the energy loss coefficient to 0.1, and the steplength on the target variables $\mathbf{y}$ to 0.01. For the MNIST-Add datasets, I set the Moreau parameter to $1.0e-3$, the energy loss coefficient to 10.0, and the steplength on the target variables $\mathbf{y}$ to $1.0e-3$.

C.2.2 Extended Learning Prediction Performance

In this section, I provide additional details on the hyperparamter settings for the learning prediction performance experiments. Table C.3 reports the hyperparameter ranges and final values for the modular learning experiments. Table C.4 reports the hyperparameter ranges and final values for the end-to-end learning experiments.

Table C.3: Hyperparameter ranges and final values for the modular learning prediction performance experiments in Table 6.12.

Dataset	Learning Loss	Parameter	Range	Final Value
CreateDebate	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	10
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
4Forums	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-3$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-3$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
Epinions	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
Citeseer	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-3$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
Cora	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
DDI	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-1$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-3$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
Yelp	Energy	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	SP	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	MSE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-1$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	10
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$
	BCE	Mirror Descent Step Length	$\{1.0e-3, 1.0e-2\}$	$1.0e-3$
		y Step Length	$\{1.0e-2, 1.0e-1\}$	$1.0e-2$
		Moreau Parameter	$\{1.0e-3, 1.0e-2, 1.0e-1\}$	$1.0e-2$
		Energy Loss Coefficient	$\{0, 1.0e-1, 1, 10\}$	0.1
		LCQP Regularization	$\{1.0e-3, 1.0e-2\}$	$1.0e-2$

Table C.4: Hyperparameter ranges and final values for the end-to-end learning experiments.

	Algorithm	Parameter	Range	Final Value
MNIST-Add1	**Energy**	**Neural Learning Rate**	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
	Bilevel	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
	Policy	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
MNIST-Add2	**Energy**	**Neural Learning Rate**	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
	Bilevel	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
	Policy	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
Visual-Sudoku	**Energy**	**Neural Learning Rate**	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-4}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1
	Bilevel	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-3}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1
	Policy	**Energy Loss Coefficient**	$\{10^{-1}, 1, 10\}$	10
		Neural Learning Rate	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-3}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1
Path-Finding	**Energy**	**Neural Learning Rate**	$\{10^{-3}, 10^{-4}, 10^{-5}\}$	10^{-3}
	Bilevel	**Energy Loss Coefficient**	$\{10^{-1}, 1\}$	1
		Neural Learning Rate	$\{5^{-4}, 10^{-4}, 10^{-5}\}$	5^{-4}
	Policy	**Energy Loss Coefficient**	$\{10^{-1}, 1\}$	1
		Neural Learning Rate	$\{5^{-4}, 10^{-4}, 10^{-5}\}$	5^{-4}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1
Citeseer	**Energy**	**Neural Learning Rate**	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Step Size	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
	Bilevel	**Energy Loss Coefficient**	$\{0, 10^{-1}, 1, 10\}$	1
		Neural Learning Rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Step Size	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
	Policy	**Energy Loss Coefficient**	$\{0, 10^{-1}, 1, 10\}$	1
		Neural Learning Rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1
Citeseer	**Energy**	**Neural Learning Rate**	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Step Size	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
	Bilevel	**Energy Loss Coefficient**	$\{0, 10^{-1}, 1, 10\}$	1
		Neural Learning Rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Step Size	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
	Policy	**Energy Loss Coefficient**	$\{0, 10^{-1}, 1, 10\}$	1
		Neural Learning Rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
		Alpha	$\{0.1, 0.5, 0.9\}$	0.1

Bibliography

Savitha Sam Abraham, Marjan Alirezaie, and Luc De Raedt. Clevr-poc: Reasoning-intensive visual question answering in partially observable environments. *arXiv*, 2024.

David Ackley, Geoffrey Hinton, and Terrence Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

Ashkay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In *NeurIPS*, 2019a.

Ashkay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2019b.

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *NeurIPS*, 2022a.

Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. In *UAI*, 2022b.

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. Semantic strengthening of neuro-symbolic learning. In *AISTATS*, 2023a.

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. A pseudo-semantic loss for autoregressive models with logical constraints. In *NeurIPS*, 2023b.

Brandom Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017.

Luca Arrotta, Gabriele Civitarese, and Claudio Bettini. Semantic loss: A new neuro-symbolic approach for context-aware human activity recognition. *Proceeding of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(144): 1–29, 2024.

Eriq Augustine, Connor Pryor, Charles Dickens, Jay Pujara, William Yang Wang, and Lise Getoor. Visual sudoku puzzle classification: A suite of collective neuro-symbolic tasks. In *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, 2022.

Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*, 18(1):1–67, 2017.

Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration - A structured survey. *ArXiv*, 2005.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *AI*, 303(4):103649, 2022.

Samy Badreddine, Luciano Serafini, and Michael Spranger. logltn: Differentiable fuzzy logic in the logarithm space. *arXiv*, 2023.

David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structure prediction energy networks. In *ICML*, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13(null):281–305, 2012.

Dimitri Bertsekas. *Control of Uncertain Systems with a Set-Membership Description of Uncertainty*. PhD thesis, MIT, 1971.

Dimitri Bertsekas. *Convex Optimization Theory*. Athena Scientific, 2009.

Dimitri Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *Neuro-Symbolic Artificial Intelligence: The State of the Art*, 2022.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, and et al. On the opportunities and risks of foundation models. *Arxiv*, 2022.

Joseph Bonnans and Alexander Shapiro. Optimization problems with perturbations: A guided tour. *SIAM Review*, 40(2):228–264, 1998.

Joseph Bonnans and Alexander Shapiro. *Perturbation Analysis of Optimization Problems*. Springer, 2000.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *ICML*, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning (FTML)*, 3(1):1–122, 2010.

Jerome Bracken and James T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

Gerhard Brewka, Thomas Eiter, and Miroshlaw Truszczynski. Answer set programming at a glance. *Communication of the ACM*, 54(12):92–103, 2011.

Nadia Burkart and Marco F Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, 2020.

Tommaso Carraro, Alessandro Daniele, Fabio Aiolli, and Luciano Serafini. Logic tensor networks for top-n recommendation. In *International Conference of the Italian Association for Artificial Intelligence (AIxIA)*, 2022.

Ming-Wei Chang, Lev Ratinov, and Dan Roth. Guiding semi-supervision with constraint-driven learning. In *ACL*, 2007.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

Yoojung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. UCLA, 2020.

William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *JAIR*, 67:285–325, 2020.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

Cristina Cornelio, Jan Stuehmer, Shell Xu Hu, and Timothy Hospedales. Learning where and when to reason in neuro-symbolic inference. In *ICLR*, 2023.

Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. The role of foundation models in neuro-symbolic learning and reasoning. *arXiv*, 2024.

John Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1040, 2022.

Artur d'Avila Garcez, Marco Gori, Luís C. Lamb, Luciano Serafini, Michael Spranger, and Son N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4):611–632, 2019.

Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer, 2002.

Artur S. d'Avila Garcez, Luís C. Lamb, and Dov M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Springer, 2009.

Peter Dayan, Geoffrey Hinton, Radford Neal, and Richard Zemel. The helmholtz machine. *Neural Computation*, 7(5):889–904, 1995.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2007.

Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *IJCAI*, 2020.

Lennert De Smet, Emanuele Sansone, and Pedro Zuidberg Dos Martires. Differetiable sample of categorical distributions using the catlog-derivative trick. In *NeurIPS*, 2023.

Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In *EMNLP*, 2016.

Stephan Dempe and Alain Zemkoho. *Bilevel Optimization*. Springer, 2020.

Vincent Derkinderen, Robin Manhaeve, Pedro Zuidberg Dos Martires, and Luc De Raedt. Semirings for probabilistic and neuro-symbolic logic programming. *International Journal of Approximate Reasoning*, page 109130, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Arxiv*, 2019.

Charles Dickens, Changyu Gao, Connor Pryor, Stephen Wright, and Lise Getoor. Convex and bilevel optimization for neuro-symbolic inference and learning. In *ICML*, 2024a.

Charles Dickens, Connor Pryor, Changyu Gao, Alon Albalak, Eriq Augustine, William Wang, Stephen Wright, and Lise Getoor. A mathematical framework, a taxonomy of modeling paradigms, and a suite of learning techniques for neural-symbolic systems. *Arxiv*, 2024b.

Charles Dickens, Connor Pryor, and Lise Getoor. Modeling patterns for neural-symbolic reasoning using energy-based models. In *AAAI Spring Symposium on Empowering Machine Learning and Large Language Models with Domain and Commonsense Knowledge*, 2024c.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *ICMLA*, 2017.

Chuong Do, Chuan-Sheng Foo, and Andrew Ng. Efficient multiple hyperparameter learning for log-linear models. In *NeurIPS*, 2007.

Justin Domke. Generic methods for optimization-based modeling. In *AISTATS*, 2012.

Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *IJCAI*, 2017.

Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie

Ruan, and Xiaowei Huang. Building guardrails for large language models. In *ICML*, 2024.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy-based models. In *NeurIPS*, 2019.

Yilun Du, Conor Durkan, Robin Strudel, Joshua B. Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *ICML*, 2023.

Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning (ML)*, 109:373–440, 2020.

Jonathan F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*. Springer Science & Business Media, 2013.

Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi Sharma, Yelong Shen, Dongyan Zhao, and Weizhu Chen. Language models can be deductive solvers. In *NAACL*, 2024.

Anthony Fiacco and Garth McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, 1968.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

Marc Genton. Classes of kernels for machine learning: A statistics perspective. *JMLR*, 2:299–312, 2001.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *Arxiv*, 2018.

Tommaso Giovannelli, Griffin Kent, and Luis Nune Vicente. Inexact bilevel stochastic gradient methods for constrained and unconstrained lower-level problems. *Arxiv*, 2022.

Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

Eleonora Giunchiglia, Mihaela Catalina Stoian, Salman Khan, and Thomas Lukasiewicz. ROAD-R: The autonomous driving dataset with logical requirements. *Machine Learning*, 112(1):3261–3291, 2023.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Will Grathwohl, KuanChieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy-based model and you should treat it like one. In *ICLR*, 2020.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2024. URL `https://www.gurobi.com`.

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language models can teach themselves to program better. In *ICLR*, 2023.

Kazi Saidul Hasan and Vincent Ng. Stance classification of ideological debates: Data, models, features, and constraints. In *IJCNLP*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

Jonathan Ho, Aajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Aapo Hyvarinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research (JMLR)*, 6:695–709, 2005.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022.

J. J. Ye and D. L. Zhu. Optimality conditions for bilevel programming problems. *Optimization*, 33(1):9–27, 1995.

Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Convergence analysis and enhanced design. In *ICML*, 2021.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv*, 2020.

Prashant Khanduri, Ioannis Tsaknakis, Yihua Zhang, Jia Liu, Sijia Liu, Jiawei Zhang, and Mingyi Hong. Linearly constrained bilevel optimization: A smoothed implicit gradient approach. In *ICML*, 2023.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic - Theory and Applications*. Prentice Hall, 1995.

Pigi Kouki, Shobeir Fakhraei, James Foulds, Magdalini Eirinaki, and Lise Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *ACM Conference on Recommender Systems (RecSys)*, Vienna, Austria, 2015.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *JBE*, 86(1):97–106, 1964.

Jeongyeol Kwon, Dohyun Kwon, Stephen Wright, and Robert Nowak. A fully first-order method for stochastic bilevel optimization. In *ICML*, 2023.

Luís C. Lamb, Artur d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *IJCAI*, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 1(0), 2006.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. BOME! bilevel optimization made easy: A simple first-order approach. In *NeurIPS*, 2022.

Ji Liu, Stephen J. Wright, Christopher Rè, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent aglorithm. *Journal of Machine Learning Research (JMLR)*, 16:285–322, 2015.

Risheng Liu, Xuan Liu, Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. A value-function-based interior-point method for non-convex bi-level optimization. In *ICML*, 2021.

Risheng Liu, Xuan Liu, Shangzhi Zeng, Jin Zhang, and Yixuan Zhang. Value-function-based sequential minimization for bi-level optimization. *Arxiv*, 2023.

Weitang Liu, Xiaoyun Wang, John D. Owens, and Yixuan Li. Energy-based out-of-distribution detection. In *NeurIPS*, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Jaron Maene and Luc De Raedt. Soft-unification in deep probabilistic logic. In *NeurIPS*, 2024.

Jaron Maene, Vincent Derkinderen, and Luc De Raedt. On the hardness of probabilistic neurosymbolic learning. *arXiv*, 2024.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence (AI)*, 298:103504, 2021a.

Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate inference for neural probabilistic logic programming. In *ICPKRR*, 2021b.

Emanuele Marconato, Stefano Teso, Antonio Vergari, and Andrea Passerini. Not all neuro-symbolic concepts are created equal: Analysis and mitigation of reasoning shortcuts. In *NeurIPS*, 2023.

Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Antonio Vergari, Andrea Passerini, and Stefano Teso. Bears make neuro-symbolic models aware of their reasoning shortcuts. *arXiv*, 2024.

Bertil Matérn. *Spatial Variation*. Springer, 1960.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *ACL*, 2020.

Paul Milgrom and Ilya Segal. Envelope theorems for arbitrary choice sets. *Econometrica*, 70(2):583–601, 2002.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. In *TGO*, 1978.

Lia Morra, Alberto Azzari, Letizia Bergamasco, Marco Braga, Luigi Capogrosso, Federico Delrio, Giuseppe Di Giacomo, Simone Eiraudo, Giorgia Ghione, Rocco Giudice, Alkis Koudounas, Luca Piano, Daniele Rege Cambrin, Matteo Risso, Marco Rondina, Alessandro Sebastien Russo, Marco Russo, Francesco Taioli, Lorenzo Vaiani, and Chiara Vercellino. Designing logic tensor networks for visual sudoku puzzle classification. In *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, 2023.

NeSy2005. *Neural-Symbolic Learning and Reasoning Workshop at IJCAI*, 2005.

NeSy2024. *International Conference on Neural-Symbolic Learning and Reasoning*, 2024.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *EMNLP*, 2023.

OpenAI. Gpt-4 technical report. Technical report, OpenAI, 2024.

Diedrik P. Kingma and Yann LeCun. Regularized estimation of image statistics by score matching. In *NeurIPS*, 2010.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *EMNLP*, 2023.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Machine Learning (FTML)*, 3(1):123–231, 2013.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, 2016.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Yang Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. In *IJCAI*, 2023a.

Connor Pryor, Quan Yuan, Jeremiah Zhe Liu, Seyed Mehran Kazemi, Deepak Ramachandran, Tania Bedrax-Weiss, and Lise Getoor. Using domain knowledge to guide dialog structure induction via neural probabilistic soft logic. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Toronto, Canada, 2023b.

Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. SIAM, 2009.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 2021.

Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *NeurIPS*, 2019.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NeurIPS*, 2011.

Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. In *ISWC*, 2003.

Stephen Robinson. Strongly regular generalized equations. *Mathematics of Operations Research*, 5(1):43–62, 1980.

R. Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

R. Tyrrell Rockafellar. Conjugate duality and optimization. In *Regional Conference Series in Applied Mathematics*, 1974.

R. Tyrrell Rockafellar and Roger J-B Wets. *Variational Analysis*. Springer, 1997.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, 2017.

Mrinmaya Sachan, Kumar Avinava Dubey, Tom M Mitchell, Dan Roth, and Eric P Xing. Learning pipelines with limited data and domain knowledge: A study in parsing physics problems. In *NeurIPS*, 2018.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *AISTATS*, 2010.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11, 2017.

Bernard Schölkopf and Alexander Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29 (3):93–106, 2008.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning (FTML)*, 4(2):107–194, 2012.

Karan Sikka, Andrew Silberfarb, John Byrnes, Indranil Sur, Ed Chow, Ajay Divakaran, and Richard Rohwer. Deep adaptive semantic logic (dasl): Compiling declarative knowledge into deep neural networks. Technical report, SRI International, 2020.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel,

et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv*, 2017.

Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Suman Saha, Kossar Jeddi Saravi, Farzad Yousefia, Jacob Culley, Tom Nicholson, Jordan Omokeowa, Salman Khan, Stanislao Grazioso, Andrew Bradley, Giuseppe Di Gironimo, and Fabio Cuzzolin. Road: The road event awareness dataset for autonomous driving. *IEEE TPAMI*, 45:1036–1054, 2021.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NeurIPS*, 2012.

Yang Song and Stefano Ermon. Generative modeling by estimating gradient of the data distribution. In *NeurIPS*, 2019.

Daouda Sow, Kaiyi Ji, Ziwei Guan, and Yingbin Liang. A primal-dual approach to bilevel optimization with multiple inner minima. *Arxiv*, 2022.

Dhanya Sridhar, James Foulds, Marilyn Walker, Bert Huang, and Lise Getoor. Joint models of disagreement and stance in online debate. In *ACL*, 2015.

Dhanya Sridhar, Shobeir Fakhraei, and Lise Getoor. A probabilistic approach for collective similarity-based drug-drug interaction prediction. *Bioinformatics*, 32 (20):3175–3182, 2016.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, 2010.

Sriram Srinivasan, Charles Dickens, Eriq Augustine, Golnoosh Farnadi, and Lise Getoor. A taxonomy of weight learning methods for statistical relational learning. *Machine Learning*, 2021.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power,

Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Annasaheb Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmuller, Andrew M. Dai, Andrew La, Andrew Kyle Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakacs, and et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *ArXiv*, 2022.

Mihaela Cătălina Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. Exploiting t-norms for deep learning in autonomous driving. In *NeSy*, 2023.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*, 2011.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Richard Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 1999.

W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285—294, 1933.

Son N. Tran and Artur S. d'Avila Garcez. Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(2):246–258, 2018.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Jivrí V. Outrata. On the numerical solution of a class of stackelberg problems. *Methods and Models of Operations Research*, 34(4):255–277, 1990.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence (AI)*, 302:103602, 2022.

Emile van Krieken, Thiviyan Thanapalasingam, Jakub M. Tomczak, Frank van Harmelen, and Annette ten Teije. A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. In *NeurIPS*, 2023.

Emile van Krieken, Samy Badreddine, Robin Manhaeve, and Eleonora Giunchiglia. Uller: A unified language for learning and reasoning. *arXiv*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data. In *ICML*, 2024.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *ICLR*, 2020.

Marilyn A Walker, Jean E Fox Tree, Pranav Anand, Rob Abbott, and Joseph King. A corpus for research on deliberation and debate. In *LREC*, 2012.

Zishen Wan, Che-Kai Liu, Hanchen Yang, Chaojian Li, Haoran You, Yonggan Fu, Cheng Wan, Tushar Krishna, Yingyan Lin, and Arijit Raychowdhury. Towards cognitive ai systems: A survey and prospective on neuro-symbolic ai. *arXiv*, 2024.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, 2019.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

Ronald Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *AAAI*, 2022.

David S Wishart, Craig Knox, An Chi Guo, Savita Shrivastava, Murtaza Hassanali, Paul Stothard, Zhan Chang, and Jennifer Woolsey. Drugbank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research (NAR)*, 34:D668–D672, 2006.

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151: 3–34, 2015.

Stephen J. Wright and Benjamin Recht. *Optimization for Data Analysis*. Cambridge University Press, 2022.

Felix Wu, Tianyi Zhang, Amauiri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, 2020.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-symbolic vqa: Disentanlging reasoning from vision and language understanding. In *NeurIPS*, 2019.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. Tractable control for autoregressive language generation. In *International Conference on Machine Learning (ICML)*, 2023.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial networks. In *ICLR*, 2017.

Kaiwen Zhou, Kaizhi Zheng, Connor Pryor, Yilin Shen, Hongxia Jin, Lise Getoor, and Xin Eric Wang. Esc: Exploration with soft commonsense constraints for zero-shot object navigation. In *International Conference on Machine Learning (ICML)*, 2023.