UNIVERSITY OF CALIFORNIA
SANTA CRUZ

BUILDING PRACTICAL STATISTICAL RELATIONAL LEARNING SYSTEMS

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

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Eriq Augustine

March 2023

The Dissertation of Eriq Augustine
is approved:

Lise Getoor, Chair

Alexander Dekhtyar

Peter Alvaro

Adam Smith

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures vi

List of Tables ix

Abstract xii

Dedication xiv

Acknowledgments xv

1 Introduction 1
- 1.1 The Four Pillars of Practical SRL Systems . . . 4
- 1.2 Contributions . . . 6

2 Background 9
- 2.1 Statistical Relational Learning and Template Graphical Models . . . 9
- 2.2 Structure in SRL . . . 14
- 2.3 Grounding in SRL . . . 16
  - 2.3.1 Independent Grounding . . . 19
- 2.4 Probabilistic Soft Logic . . . 20
  - 2.4.1 Weight Learning in PSL . . . 21
- 2.5 Markov Logic Networks . . . 23
  - 2.5.1 Diagonal Newton Method for Weight Learning (DN) . . . 25

3 Scalability 26
- 3.1 Tandem Inference . . . 26
  - 3.1.1 Tandem Inference Methodology . . . 29
  - 3.1.2 Empirical Evaluation . . . 39
- 3.2 Collective Grounding . . . 49

3.2.1 Collective Grounding Methodology . . . 52
3.2.2 Empirical Evaluation . . . 63
3.3 Conclusions and Future Work . . . 76

4 Expressivity **79**
4.1 NeuPSL . . . 79
4.1.1 Related Work . . . 81
4.1.2 Neuro-Symbolic Energy-Based Models . . . 82
4.1.3 Neural Probabilistic Soft Logic . . . 84
4.1.4 NeuPSL Inference and Learning . . . 87
4.1.5 Joint Reasoning in NeSy-EBMs . . . 91
4.1.6 Empirical Evaluation . . . 93
4.2 Negative Weights . . . 99
4.2.1 Related Work . . . 102
4.2.2 Methodology . . . 103
4.2.3 Empirical Evaluation . . . 113
4.3 Conclusions and Future Work . . . 119

5 Model Adaptability **120**
5.1 Weight Learning . . . 120
5.1.1 Black-box Optimization . . . 123
5.1.2 Search-Based Approaches for Weight Learning . . . 126
5.1.3 Efficient Space to Search for Weights . . . 142
5.1.4 Accommodating Negative Weights in Markov Logic Networks . . . 151
5.1.5 Empirical Evaluation . . . 152
5.2 Online Inference . . . 164
5.2.1 Related Work . . . 167
5.2.2 Online Collective Inference . . . 168
5.2.3 Model Instantiation . . . 172
5.2.4 Stability and Regret of Online Inference . . . 174
5.2.5 Empirical Evaluation . . . 181
5.3 Conclusions and Future Work . . . 186

6 Usability **188**
6.1 Standard Relational Language Interface . . . 189
6.1.1 The Components of SRL . . . 190
6.1.2 Recreating Existing SRL Frameworks . . . 195
6.1.3 Implementing Custom SRL Engines . . . 196
6.1.4 A Complete SRLi Model . . . 198
6.1.5 Empirical Evaluation . . . 199
6.2 VMI-PSL: Visual Model Inspector for Probabilistic Soft Logic . . . 202

6.2.1 System Overview . . . 205
6.2.2 Example Workflows . . . 206
6.3 Conclusions and Future Work . . . 208

7 Conclusions and Future Work **209**

A Appendix **259**
A.1 Collective Grounding . . . 259
A.1.1 Grounding Pseudocode . . . 259
A.1.2 Full LASTFM Model . . . 262
A.1.3 Full DDI Model . . . 262
A.2 Weight Learning Sampling . . . 264
A.2.1 Surface of a Unit Hypersphere as Search Space . . . 264

List of Figures

1.1 Modern data comes from many different sources and takes on varying degrees of complexity. Data may be simple and come from flat files, but can also come from intricate networks, databases, wired and wireless devices, and social networks. . . 2
1.2 The four pillars of practical SRL along with my primary contributions to each pillar. 6

2.1 Example rules that encode the intuition that user may purchase items that taste or look similar to items they have already purchased. 10
2.2 Example data used in SRL organized by logical predicate. 11
2.3 An example mapping of variables to constants that results from grounding. 12
2.4 Example ground rules created when variable-constant mappings are applied to template rules. 12
2.5 An example probabilistic graphical model that is derived from representing ground rules as cliques. 13
2.6 An overview of the grounding process for logical rules. Weighted logical rules ((t_1, w_1) and (t_2, w_2)) are provided by the user and combined with the provided ground data to produce all possible mappings of the variables used in each template to constants from the provided data. The mappings are used to instantiate the templates to produce the ground rules: $\{\phi_1, \phi_2, \phi_3, \phi_4\}$. The ground rules are then used to construct a graphical model (represented in this example as a factor graph). 15

3.1 Block diagram showing the TI system architecture. 29
3.2 Network sequence diagram for iteration 1 of TI. 30
3.3 Network sequence diagram for iterations 2 through T of TI. 31
3.4 Comparison of the runtimes of TI, ADMM, and SGD on 10 datasets. 39
3.5 Maximum memory usage for TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages. 44

3.6 Number of I/O operations per optimization iteration of TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages. 45
3.7 Runtime per optimization iteration of TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages. 46
3.8 The effect of different optimizers on convergence. 48
3.9 An example of how the collective grounding process converts relational templates (t_1, t_2, t_3) into a grounding plan. Colors are used to represent the rule that queries and candidates satisfy, color gradients indicate that multiple templates are satisfied. 53
3.10 A base query, Q_2, and candidate query, C, both being used to ground the same rule template, t_2. The candidate query is simpler, but produces more results. The extraneous results (in red) from the candidate query can be filtered out by invoking the SRL framework's validation function, V. 55
3.11 A fully materialized candidate search tree for rule t_1 from Figure 2.6. Duplicate nodes are excluded and nodes that do not contain the required variables (and are therefore invalid) are shaded. 56
3.12 A fully expanded candidate search tree created from a single base query (the root). Each child represents a candidate that can be formed by dropping one relation from each parent. Each candidate has two colors: the top color represents the pessimistic cost, while the bottom color represents the optimistic cost. Colors closer to blue represent a lower cost while colors closer to red represent a higher cost. Duplicate nodes are omitted. 59
3.13 Runtime results for independent grounding (IG) and collective grounding (CG) using both the PER-DATASET and OVERALL hyperparameters. All results are aggregated over ten iterations, run on the same machine, and had database and disk caches cleared between runs. 69

4.1 NeuPSL inference and learning pipeline. 84
4.2 Example of non-joint and joint energy functions. 92
4.3 Example of overlapping MNIST images in **MNIST-Add1**. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros. 96
4.4 Average test set accuracy and standard deviation on **MNIST-Add** datasets with varying amounts of overlap. 97
4.5 Average test set accuracy and standard deviation on **Visual-Sudoku-Classification** dataset with varying amounts of overlap. 99
4.6 Example non-convex MAP inference objective function for a negative weight PSL model. The negative weight is interpreted using the approach described in Section 4.2.2. The weights for the model in Equation 4.8 are set to $w_1 = 1$ and $w_2 = -10$. 105
4.7 Surface plots and heat maps of the potential values instantiated by the example negated rule (4.12) using the three negation-based approaches. 109

4.8 Bar plot of the mean RMSE of all five methods for supporting the negative weights in the experiment model for ten folds of each of the three variations of the synthetic datatset. One standard deviation is shown with error bars. 116

5.1 Heat map for AUROC and log-likelihood for a model with two rules parameterized by their respective weights w_1 and w_2. The lighter color indicates higher values and higher values are desired for both metrics. 127
5.2 Analyzing the scalability of different approaches on the number of rules and groundings. With number of iterations fixed, search-based approaches scale better with both the number of rules and groundings. 161
5.3 Comparison of normalized MAP inference objective (top), domain-specific evaluation metrics (middle), and runtime (bottom) of the three systems. 183
5.4 Comparison of warm-start and cold-start variable movements plotted with various theoretical and empirically estimated bounds. 184

6.1 The three different context layers in VMI-PSL, along with a module from each layer. Here, the ground atom RATING('8', '1769') is the context for the `ground atom` layer and the ground rule BPMF_RATING('8', '1769') $\rightarrow$ RATING('8', '1769') is the context for the `ground rule` layer. 205
6.2 Three different workflows showcasing different use cases of VMI-PSL: debugging an erroneous model, analyzing the performance of components of a working model, and explaining specific predictions. 207

A.1 The full LASTFM model used in Kouki et al. [2015]. 263
A.2 The full DDI model used in Sridhar et al. [2016]. 264
A.3 Visualization of Dirichlet distribution with different A of a model with three rules. Visualization shown both in OS and SS. 265

List of Tables

3.1 Details of models used and their memory consumption for non-streaming inference. Memory usage for FRIENDSHIP-500M and FRIENDSHIP-1B are estimates. The machine has only 400GB of memory and was unable to fully run FRIENDSHIP-500M and FRIENDSHIP-1B, so I had to estimate the memory usage. 41

3.2 Details of datasets and models used for evaluation. The generic task performed on each dataset include collective classification (CC), link prediction (LP), and recommendation (REC). "Minimum Queries" shows the minimum possible queries that the model can be grounded with (determined by manual inspection). 65

3.3 The best performing hyperparameters on the validation split for each dataset. The row labeled OVERALL contains the hyperparameters that on average performed the best over all datasets. 66

3.4 The runtime (in milliseconds) and standard deviation for independent grounding (IG) and collective grounding (CG) using both the PER-DATASET and OVERALL hyperparameters averaged over 10 splits. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold. 68

3.5 The effect of search budget on the search time and overall runtime. All times are shown in milliseconds, aggregated over ten runs, and include their standard deviation. 71

3.6 The number of queries run in collective grounding (using the OVERALL hyperparameters) compared to the number of rules in each dataset. 72

3.7 The performance of base queries compared against simpler queries chosen by collective grounding for the DDI dataset. Note that the simpler queries used by CG results in significantly faster runtimes while only returning a few additional query results (which are subsequently filtered out by the SRL framework). The OVERALL hyperparameters are used for CG. 75

3.8 The performance of independent grounding and collective grounding against forcing a grounding plan that uses only two queries (see Equation 3.12). CG runs used OVERALL hyperparameters. 76

3.9 A timeline of the grounding milestones in the SRL community. The size column refers to the number of ground rules in the model. All systems except FoxPSL were run on the same machine. All PSL rows are run on the IMDB-ER dataset (discussed in Section 3.2.2). Time spent grounding does not apply directly to TI, since grounding and inference occur at the same time. 77

4.1 Inference times (milliseconds) on test sets constructed from $10,000$ MNIST images. 95
4.2 Test set accuracy and standard deviation on **MNIST-Add**. 96
4.3 Test set accuracy and standard deviation on **Visual-Sudoku-Classification**. 98
4.4 A truth table showing the value of the potential function of a conjunction and the three disjunctions that can represent it. 112
4.5 The mean and standard deviation of the number of epochs required to reach convergence of ADAM SGD optimization for each method and dataset. 118

5.1 Performance of PSL weight learning methods across datasets; the best scoring methods (with $p < 0.05$) are shown in bold. Note: the metric values are rounded to three point precision making some numbers the same, but the significance test was performed on values with six point precision. 156
5.2 Performance of MLN weight learning methods across datasets; the best scoring methods (with $p < 0.05$) are shown in bold. 158
5.3 The table shows the mean (std) of the metrics obtained by running search-based approaches with varied initialization. The performance of BOWL is least affected by both initialization. 162
5.4 Performance of different search-based approaches by varying the A parameter in the Dirichlet distribution. The best metric value in every row is shown in bold. . . . 164
5.5 The table shows the effect of metrics obtained by using different acquisition function with BOWL. The performance of BOWL is unaffected by both acquisition function. 165

6.1 SRL engines provided by SRLi. 199
6.2 Details of datasets and models used for evaluation. The generic task performed on each dataset include collective classification (CC), link prediction (LP), and recommendation (REC). 200
6.3 The result of running multiple SRLi engines. A check mark (✓) means that the engine ran without issue, TIMEOUT indicates that the engine was killed after two hours, and MEMORY indicates that the engine ran out of memory. 201
6.4 The mean quality metric and standard deviation of different SRLi methods run over several established relational datasets averaged over 5 splits. The presented metric is the metric used in the original publication. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold. 203

6.5 The mean runtime and standard deviation of different SRLi methods run over several established relational datasets averaged over 5 splits. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold. 203

Abstract

Building Practical Statistical Relational Learning Systems

by

Eriq Augustine

In our increasingly connected world, data comes from many different sources, in many different forms, and is noisy, complex, and structured. To confront modern data, we need to embrace the structure inherent in the data and in the predictions. Statistical relational learning (SRL) is a subfield of machine learning that provides an effective means of approaching this problem of structured prediction. SRL frameworks use weighted logical and arithmetic expressions to easily create probabilistic graphical models (PGMs) to jointly reason over interdependent data. However, despite being well suited for modern, interconnected data, SRL faces several challenges that keep it from becoming practical and widely used in the machine learning community. In this dissertation, I address four pillars of practicality for SRL systems: *scalability*, *expressivity*, *model adaptability*, and *usability*. My work in this dissertation uses and extends Probabilistic Soft Logic (PSL), a state-of-the-art open-source SRL framework.

Scalability in SRL systems is essential for using large datasets and complex models. Because of the complex nature of interconnected data, models can easily outgrow available memory spaces. To address scalability for SRL, I developed methods that more efficiently and intelligently instantiate PGMs from templates and data. I also developed fixed-memory inference methods that can perform inference on very large models without requiring a proportional amount of memory.

Expressivity allows SRL systems to represent many different problems and data patterns. Because SRL uses logical and arithmetic expressions to represent structured dependencies, SRL frameworks need to be able to express more than just what is represented by feature vectors. To address expressivity for SRL, I created a system to incorporate neural models with structured SRL inference, and expanded the interpretation of PSL weight hyperparameters to include additional types of distributions.

Model adaptability is the ability of SRL frameworks to handle models that change. A changing model can be as simple as a model that has its hyperparameters updated, or as complex as a model that changes its structure over time. To address model adaptability for SRL, I developed new weight learning approaches for PSL, and created a system for generalized online inference in PSL.

Usability make SRL frameworks easy for people to use. Because of the need to model structural dependencies, SRL frameworks are often harder to use when compared to more common machine learning libraries. To address usability for SRL, I have created a new SRL framework that removes the tight coupling between the different components of the SRL pipeline that is seen in other SRL frameworks and allows the the recreation of exiting SRL frameworks along with the creation of new SRL frameworks using the same common runtime. Additionally, I developed a visual model inspector for analyzing and debugging PSL models.

To all those seeking a better world where people and technology live in peace with nature.

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Lise Getoor. She has supported me from day one and has seen me through some truly difficult times. Without her guidance and support, I would not be where I am today.

A thank you to all my lab members. Research is not done in a vacuum, and I could not have asked for better lab members. Specifically I am grateful to: Jay Pujara and Pigi Kouki for mentoring me when I was a junior student, Sriram Srinivasan for always being ready for new ideas with a positive attitude, Varun Embar for looking at problems in a different way from me, Connor Pryor for always being there to work through difficult problems with a spirited debate, and Charles Dickens for always be ready to test out new ideas.

A special thanks to my family, particularly my siblings. They are a constant support that I can always turn towards.

Thanks to my committee who has put up with my long drafts. A special thanks to Dr. Alexander Dekhtyar who also served as my Master's advisor. Without Alex, I may have never pursued graduate school or met Lise.

Portions of this dissertation were supported by National Science Foundation grants CCF-1740850, CCF-2023495, IIS-1703331, and IFDS DMS-2023495; AFRL; the Defense Advanced Research Projects Agency; 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

In our increasingly connected world, data comes from many different sources, in many different forms, and is noisy, complex, and structured. To confront modern data, we need to reject past tendencies to flatten data into a simple homogeneous data table and embrace the structure inherent in the data, e.g. graphs, and in the predictions, e.g. constraints. The task of performing predictions in the presence of structured inputs and outputs is referred to as *structured prediction* Baki et al. [2007]. Considering the structure of the problem often leads to dramatic increases in predictive performance, but also creates a substantially more complex inference problem. Over the years, many approaches, such as relational models, structured support vector machines, and conditional random fields, have been proposed to perform structured prediction Friedman et al. [1999], Taskar et al. [2004], Baki et al. [2007], Joachims et al. [2009].

One of these structured predictions methods is statistical relational learning (SRL) Getoor

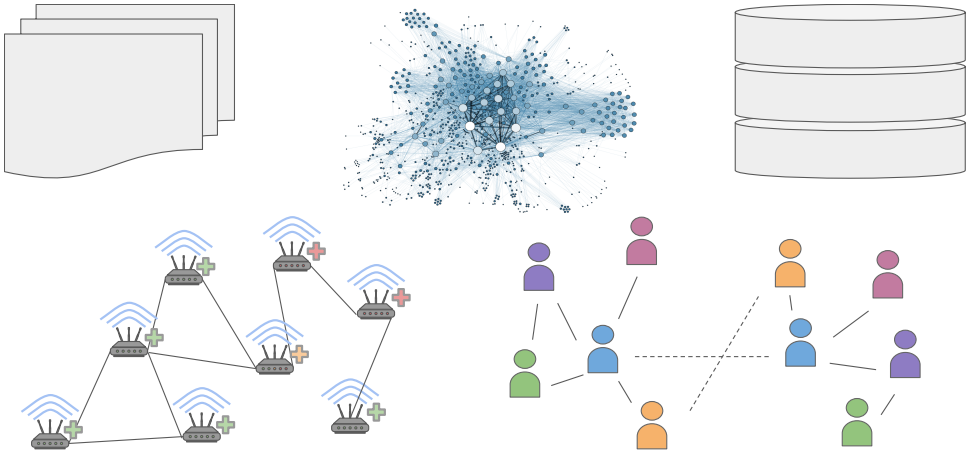


Figure 1.1: Modern data comes from many different sources and takes on varying degrees of complexity. Data may be simple and come from flat files, but can also come from intricate networks, databases, wired and wireless devices, and social networks.

and Taskar [2007]. SRL Richardson and Domingos [2006], Getoor and Taskar [2007], Raedt and Kersting [2011] is an effective method of combining weighted first-order logic with probabilistic inference to make high-quality, structured predictions. Prior to SRL, there have long been attempts at performing inference over logical expression Robinson [1965] as well as probabilistic inference over structured data Cavallo and Pittarelli [1987]. A characteristic trait of SRL is the ability to generate large probabilistic graphical models (PGMs) from a small set of logical templates (rules). Several different SRL frameworks have been proposed, each making use of different types of graphical models, inference algorithms, or hyperparameter learning methods Richardson and Domingos [2006], Bach et al. [2017], Poole and Mackworth [2017], Raedt et al. [2007], Riedel [2008], Kok et al. [2009], Niu et al. [2011], Noessner et al. [2013], Magliacane et al. [2015], Alberti et al. [2017]. SRL methods have achieved state-of-the-art results in a varied set of domains such as image classification Gridach et al. [2017], Aditya et al. [2018a], scene understanding Aditya et al.

[2018b], activity recognition London et al. [2013b], natural language processing Ebrahimi et al. [2016], Johnson et al. [2017], Beltagy et al. [2014], Deng and Wiebe [2015], Khot et al. [2015a], Ebrahimi et al. [2016], Wang and Ku [2016b], Johnson and Goldwasser [2016a], Johnson et al. [2017], bioinformatics Sridhar et al. [2016], recommender systems Kouki et al. [2015], Lalithsena et al. [2017a], diagnosis of physical systems Chen et al. [2014], and knowledge bases Pujara et al. [2013], Pujara and Getoor [2016], Embar et al. [2018], and information retrieval Alshukaili et al. [2016], Platanios et al. [2017a], Srinivasan et al. [2019b].

The deep learning community has created several very large datasets which are able to achieve impressive results, such as the LAION-5b Schuhmann et al. [2022] consisting of billions of images and text captions scraped from the public internet. These datasets, however, make limited use of the structure found in the source data. For example, image-text datasets like LAION-5b includes images with a text caption. However an image with caption may be published as part of a larger article (that contains relevant text) on a blog (that specializes on specific topics) by an author (that is an expert in specific fields) on a specific date. All of this contextual information is useful and adds structure to the data. While there are some neural architectures that attempt to encode limited forms of structure, many of the deep neural approaches ignore structure and instead rely on large amounts of data to make up for the lack of context. As a result, these neural models struggle when presented that data that is less plentiful, noisy, and interconnected. Later in this dissertation I will show how relevant methods from deep learning can be integrated into larger SRL systems to gain the benefits of both learning paradigms.

However, despite being well suited for modern, interconnected data, SRL has several

challenges that keep it from becoming practical and widely used in the machine learning community. These challenges include the steep complexity that accompanies processing and predicting interconnected data, the difficulty of capturing the complexity of realworld structure in a model, the need for models to update with new and changing data, and building frameworks that are accessible to users of all skill levels. In this dissertation, I categorize the challenges of building SRL systems into four pillars of practicality: *scalability*, *expressivity*, *model adaptability*, and *usability*. My work in this dissertation demonstrates how all four of these pillars can be addressed to make practical SRL frameworks.

1.1 The Four Pillars of Practical SRL Systems

Scalability in SRL systems allows for the use of large datasets and complex models. Because of the complex nature of interconnected data, models can easily outgrow available memory spaces. Much of the scalability issues in SRL stems from the rejection of the independent and identically distributed (i.i.d.) assumption, which states that each data point is independently drawn from a fixed distribution Bishop [2006]. Using the i.i.d. assumption allows models to ignore the structure in the data, and collect sufficient statistics to estimate parameters by just summing over the data. Setting aside this assumption and embracing the structure in the data allows SRL methods to use relational information that would otherwise not be used in i.i.d. models. However, ignoring the i.i.d. assumption also imposes a large computational burden on SRL methods. SRL methods must create a collective model over all connected variables and make predictions jointly.

Expressivity allows SRL systems to represent many different problems and data patterns. Traditionally, SRL uses logical and arithmetic expressions to represent structured dependencies. This representation has proven to be effective at representing symbolic information. However, SRL frameworks need to be able to express more than just what can be represented at the symbolic level Raedt et al. [2020]. One specific class of models that SRL has historically struggled to support are models that deal with high-dimensional, low-level feature spaces.

Model adaptability is the ability of SRL frameworks to handle models that change. A model may change because new or updated data is encountered, or core assumptions about the model have changed and need to be reworked. In the simple case, working with an evolving model can mean tuning the hyperparameters of a model. In a more complex setting, working with an evolving model may mean using a model whose distribution may change mid-inference, as in the online setting.

Usability focuses on making SRL frameworks easy for people to use, and focuses on three different types of users: beginners new to machine learning, experts well-versed with other machine learning frameworks, and domain experts that specialize in domain modeling over machine learning. Building usable systems is particularly difficult for SRL because there are many different ways structure can be incorporated into a model, and there is no consensus or common language for "structure".

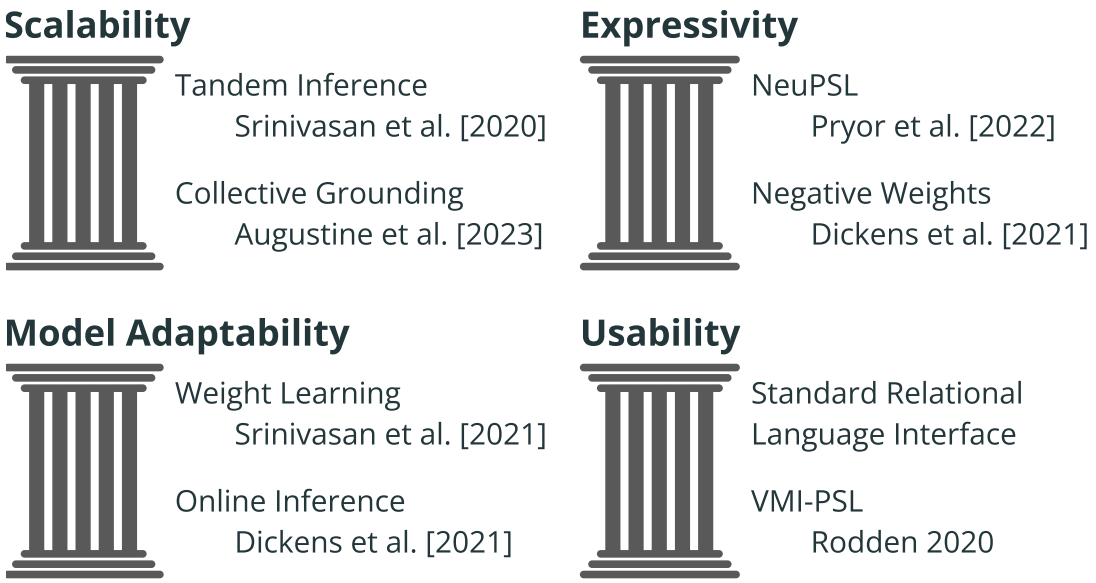


Figure 1.2: The four pillars of practical SRL along with my primary contributions to each pillar.

1.2 Contributions

The key contributions of this dissertation all contribute to the common goal all building practical SRL systems. More specific contributions are listed in later chapters, and here I discuss the high-level contributions to each pillar of practical SRL, as shown in Figure 1.2.

Scalability To create scalable SRL systems, I tackle the two largest computational bottlenecks in SRL: inference and grounding. To address scalable inference, I introduce a novel approach of out-of-core optimization technique for SRL called *tandem inference* (TI) Srinivasan et al. [2020a]. I show how TI can infer over the largest published SRL models to-date using a fixed amount of memory. To address the challenge of scalable grounding, I create a new paradigm for jointly grounding SRL models called *collective grounding*. Collective grounding uses the relational information of an SRL

model along with techniques from database research to ground programs up to five times faster than previous state-of-the-art methods. Based on my research contributions, SRL systems have evolved from processing 2.5 thousand terms per minute in 2009 Kok et al. [2009] to 13 million terms per minute Augustine and Getoor [2018] prior to my work in this dissertation to 65 million terms per minute now.

Expressivity I build more expressive SRL systems by integrating new types of models into SRL frameworks and expanding the representation of weight hyperparameters in SRL models. First, I developed a system called *Neural Probabilistic Soft Logic* (NeuPSL) Pryor et al. [2022], which integrates neural networks with symbolic reasoning. Using NeuPSL, I show how problems using neural perception, previously outside the scope of SRL systems, can be integrated into relational models. Then, I show how the weights for the rules of an SRL framework can be expanded to allow for negative values, allowing for a wider range of models that describe negative associations between variables.

Model Adaptability To make SRL models that are adaptable, I introduce new methods of learning weights for SRL models as well as a method of extending SRL models to handle new and updating information. My work creating new weight learning methods allow models to better tune themselves to specific domains and datasets Srinivasan et al. [2021]. I demonstrate how these methods outperform existing weight learning methods over several realworld datasets. Additionally, I formulate a way to extend SRL existing models through a method called *online collective inference*

Dickens et al. [2021]. I show how online collective inference can not only adapt to new and updated information, but can also change the model itself without fully rebuilding the underlying graphical model.

Usability In my effort towards making usable and accessible SRL systems, I designed and implemented a new general SRL framework as well as a tool to debug SRL systems. My new SRL framework, called the *Standard Relational Language Interface* (SRLi), removes the tight coupling between the different components of the SRL pipeline that is seen in other SRL frameworks. By doing this, SRLi is able to reproduce existing SRL systems, implement custom SRL systems, and construct reusable components that can be shared amongst multiple SRL systems. Additionally, I create a tool called the *Visual Model Inspector* (VMI) Rodden et al. [2020] that allows users to debug and inspect SRL models.

Chapter 2

Background

To lay the foundations for the work discussed in this dissertation, in this chapter I provide an overview of *statistical relational learning* (SRL), templated graphical models, and two popular SRL frameworks used throughout my experiments (Probabilistic Soft Logic (PSL) and Markov Logic Networks (MLNs)).

2.1 Statistical Relational Learning and Template Graphical Models

Statistical Relational Learning (SRL) combines the power of statistical inference with relational data to produce rich models with intricate constraints and dependencies Getoor and Taskar [2007]. Modeling relational data is inherently complicated by the large number of interconnected and overlapping structural dependencies. To make modeling relational data easier, many SRL frameworks use familiar first-order logic as a compact representation of the model Raedt et al.

```
t1 - w1: Bought(User, Item1) ∧ SimilarTaste(Item1, Item2) → Recommend(User, Item2)
t2 - w2:  Bought(User, Item1) ∧ SimilarLook(Item1, Item2) → Recommend(User, Item2)
```

Figure 2.1: Example rules that encode the intuition that user may purchase items that taste or look similar to items they have already purchased.

[2007], Riedel [2008], Kok et al. [2009], Niu et al. [2011], Noessner et al. [2013], Magliacane et al. [2015], Poole and Mackworth [2017], Bach et al. [2017], Alberti et al. [2017]. The logical rules acts as weighted templates that can be instantiated with data to form the factors of a graphical model. Figure 2.1 shows two sample rules, t_1 and t_2 with respective weights w_1 and w_2. These rules encode the intuition that user may purchase items that taste or look similar to items they have already purchased.

I refer to models that use templates to specify the complex structure of a graphical model as *templated graphical models (TGMs)* Koller and Friedman [2009]. Instead of individually specifying each dependency in the graphical model, template factors are used to model the patterns of the structure seen in the model. TGMs are then instantiated with data to produce a full graphical model. TGMs allow users to specify and reason about graphical models that would otherwise be too large to specify. The key unit of a TGM is the *template* (represented with a t). Templates can take many forms, but the most common form in SRL is a first-order logical rule. Additionally, each SRL framework enforces its own limitations on the for of the logical rules (e.g., universal and existential quantification, horn clauses, etc.). For example, Prolog is based around horn clauses Lloyd [1987]; Markov Logic Networks (MLNs) Richardson and Domingos [2006] allow both universal and existential quantification and is not restricted to horn clauses; and Probabilistic Soft

Logic (PSL) Bach et al. [2017] only uses universal quantification and requires logical rules to reduce to a 1-DNF Kononenko and Kukar [2007], but also allows for non-logical templates in the form of inequalities of linear combinations. For simplicity, the remainder of this dissertation generally assumes that all templates are logical rules, but the introduced theory applies to any form of template or constraint that can be evaluated for truth (e.g., logical rules, arithmetic inequalities, etc). Thus, any reference to a "rule" will refer to a logical rule template, while references to "template" will refer to general templates.

Data in SRL is generally organized by logical *predicate*, and each instantiation of a logical predicate is referred to as an *atom*. These atoms will eventually become random variables in a graphical model, while the template establishes a dependency between the random variables it references. Figure 2.2 shows how data used by SRL is typically organized. Each table represents a predicate, while the rows in each table each represent an atom.

SimilarLook:

Arg1	Arg2	Value
orange	grapefruit	TRUE
orange	tangerine	TRUE
grapefruit	tangerine	FALSE

SimilarTaste:

Arg1	Arg2	Value
orange	grapefruit	FALSE
orange	tangerine	TRUE
grapefruit	tangerine	FALSE

Bought:

Arg1	Arg2	Value
alice	orange	TRUE

Recommend:

Arg1	Arg2	Value
alice	grapefruit	?
alice	tangerine	?

Figure 2.2: Example data used in SRL organized by logical predicate.

A rule that is grounded (contains only constants and no variables) is referred to as a *ground template* or *ground rule*, and a logical predicate with only constant arguments is referred to as a *ground atom*. The first conceptual component of grounding is to establish a mapping of variables seen in the rules to real data. Figure 2.3 shows what this mapping may look like for the rules and

data seen in the examples above.

Rule 1 Results:

User	Item1	Item2
alice	orange	grapefruit
alice	orange	tangerine

Rule 2 Results:

User	Item1	Item2
alice	orange	grapefruit
alice	orange	tangerine

Figure 2.3: An example mapping of variables to constants that results from grounding.

Once a mapping of variables to constants is created for each rule, that mapping can be used to instantiate one ground rule for each pairing. Example ground rules that use the above mapping are shows in Figure 2.4.

$\phi_1 - w_1$: Bought(*alice*, *orange*) $\wedge$ SimilarTaste(*orange*, *grapefruit*) $\rightarrow$ Recommend(*alice*, *grapefruit*)
$\phi_2 - w_1$: Bought(*alice*, *orange*) $\wedge$ SimilarTaste(*orange*, *tangerine*) $\rightarrow$ Recommend(*alice*, *tangerine*)

$\phi_3 - w_2$: Bought(*alice*, *orange*) $\wedge$ SimilarLook(*orange*, *grapefruit*) $\rightarrow$ Recommend(*alice*, *grapefruit*)
$\phi_4 - w_2$: Bought(*alice*, *orange*) $\wedge$ SimilarLook(*orange*, *tangerine*) $\rightarrow$ Recommend(*alice*, *tangerine*)

Figure 2.4: Example ground rules created when variable-constant mappings are applied to template rules.

As seen Figure 2.5, the collection of ground templates from all rules jointly create a graphical model that is then used for inference or parameter (weight) learning in SRL. Note that at the graphical model level random variables are represented by ground atoms, and not by the variables used in the logical expressions (e.g., `User`, `Item1`, and `Item2`). Shaded nodes in Figure 2.5 represent observed random variables, collectively referred to as X, while unshaded nodes represent unobserved random variables, collectively referred to as Y. I use the following notation to denote a template, t, being instantiated with observed, X, and unobserved, Y, random variables to create a

set of h ground templates, ϕ:

$$t(X, Y) = \{\phi_1, ...\phi_h\} \tag{2.1}$$

I intentionally keep the notation for instantiating templates vague, as the exact definition differs between different SRL implementations. For example, some languages like MLN Richardson and Domingos [2006] and FoxPSL Magliacane et al. [2015] use typed variables and infer domains for each type. These types are then used to instantiate ground atoms which are in-turn used to instantiate ground templates. Whereas, other languages like PSL Bach et al. [2017] explicitly define all unobserved ground atoms and can directly instantiate ground templates.

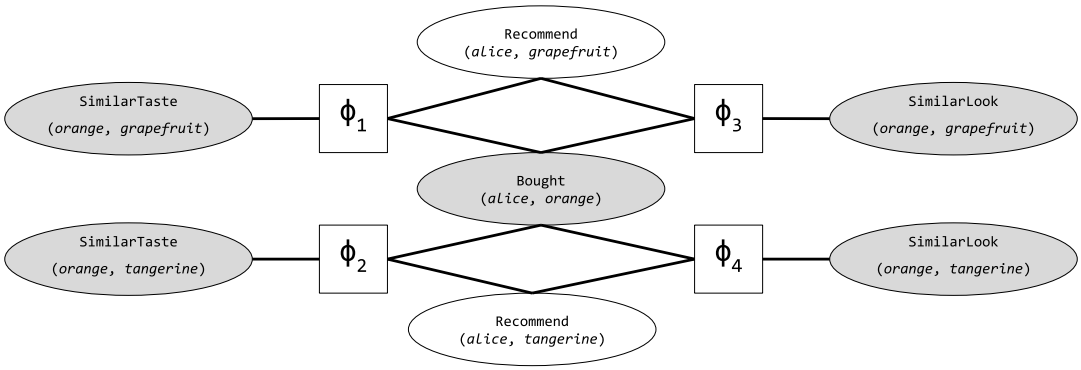


Figure 2.5: An example probabilistic graphical model that is derived from representing ground rules as cliques.

Once all ground templates are created, they produce a graphical model of the form:

Definition 1. *Let $X = (x_1, ..., x_m)$ be a vector of known variables, $Y = (y_1, ..., y_n)$ be a vector of unknown random variables, $T = (t_1, ..., t_l)$ be a set of rules, $W(t)$ be a set real-valued function that emits a weight corresponding to a template, and $t(X, Y) = \{\phi_1, ..., \phi_h\}$ be a set of ground templates created by instantiated the template t with X and Y. Additionally, let ϕ emit a score*

representing the ground template's satisfaction. Then, a templated graphical model is a probability distribution of the form:

$$P(Y|X) = \frac{1}{Z} \exp\big(- \sum_{t \in T} \sum_{\phi \in t(X,Y)} W(t) \cdot \phi \big)$$

where

$$Z = \sum_{X} \exp\big(- \sum_{t \in T} \sum_{\phi \in t(X,Y)} W(t) \cdot \phi \big)$$

Figure 2.6 shows how all of the above components fit together in the standard SRL pipeline that starts with rules and data and creates a complex graphical model.

2.2 Structure in SRL

Throughout this dissertation, I continually reference "structure" in data, in models, and in problems. But what exactly is structure? In a general sense, structure is anything that relates two or more variables in a model. More specifically, structure occurs in two forms: in a model's inputs and in a model's outputs.

Structure in a problem's inputs relate the input variables of the problem, and is often derived from the connected nature of the input data. Data in the form of graphs naturally provide input with structure. For example, a social network may provide structure in the form of a user's friendship connections, liked posts, and group memberships. Thus respectively relating two users, a

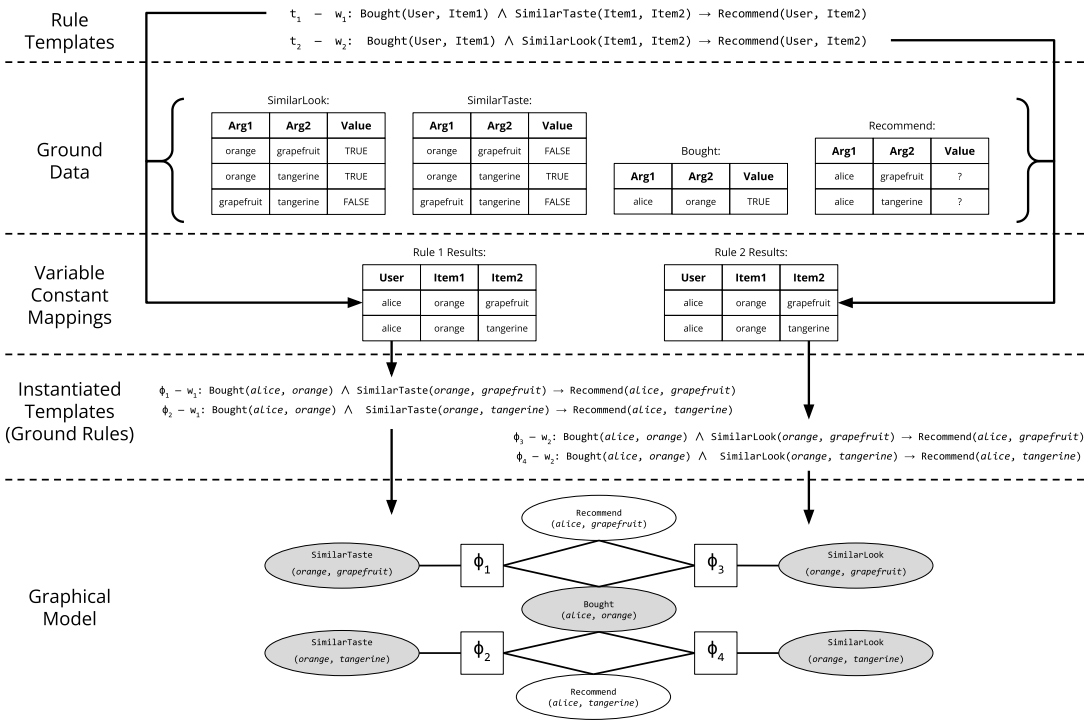


Figure 2.6: An overview of the grounding process for logical rules. Weighted logical rules ((t_1, w_1) and (t_2, w_2)) are provided by the user and combined with the provided ground data to produce all possible mappings of the variables used in each template to constants from the provided data. The mappings are used to instantiate the templates to produce the ground rules: $\{\phi_1, \phi_2, \phi_3, \phi_4\}$. The ground rules are then used to construct a graphical model (represented in this example as a factor graph).

user to a post, and a user to a group.

Structure in a problem's outputs relate the output variables of the problem, and are often derived from domain knowledge. This type of structure is sometimes also referred to as "constraints". For example, when predicting friendship links in a social network, a constraint can be applied that expresses users' tendency to form friendships with users in the same groups. This constraint conditions a prediction of friendship on the users' group memberships. Thus constraining an output variable (the prediction of friendship) using domain knowledge.

2.3 Grounding in SRL

The task of grounding in the context of SRL is to use the provided data to produce the complete set of instantiated templates, i.e., for each template grounding finds all possible substitutions of variables in the template to constants in the data. Grounding is heavily featured in Section 3.1, Section 3.2, and Section 5.2.

Specifically, grounding in SRL aims to create all ground rules in the set:

$$\{\phi | V(\phi) = \text{TRUE} \ \wedge \ \phi \in \bigcup_{t \in T} t_i(X, Y)\} \tag{2.2}$$

where V is a validation function used to filter out ground rules that do not belong in the final graphical model. This validation function serves two critical roles in SRL grounding: V ensures that each ground rule conforms to the structure defined by the ground rule's parent template, and V enforces the semantics of the specific SRL framework being used.

In preparation for grounding, data in SRL systems are generally organized into tables

that each represent a single logical predicate. The desired structure is shown in Figure 2.6. SRL frameworks differ on how they handle variable types, type domains, data loading, data parsing, and initial data representations, but to ground efficiently similar per-predicate tables structures are used. Once that data is loaded into the relevant tables, the logical predicates are categorized into two different sets: 1) *closed predicates*, which only contain observed data, X, and 2) *open predicates*, which contain unobserved data, Y, and may also contain observed data, X. Closed predicates are particularly important because the closed world assumption will be applied to them, i.e., any ground atom from a closed predicate that is not explicitly specified in the data will be assumed to have a value of FALSE (or the equivalent of FALSE in the respective SRL framework).

SRL frameworks utilize a ground rule validation function, V, to enforce common semantics that utilize the closed world assumption to remove useless ground rules Kok et al. [2009], Domingos and Lowd [2009], Glass and Barker [2012], Beltagy et al. [2014], Beltagy and Mooney [2014]. Ground rules may be determined to be useless, or *trivial*, if they are logically satisfied or unsatisfied in all possible worlds. Together, the closed-world semantics and trivial ground rules provide a substantial opportunity in the grounding of SRL programs that may not be present when grounding general logical programs. By applying these two concepts, SRL frameworks can remove large portions of ground programs that are not relevant to the distribution in Definition 1. For example, consider the ground rule ϕ_1 from Figure 2.6. SIMILARTASTE(`orange`, `grapefruit`) has a fixed value of $FALSE$, thereby causing the implication to always take on the value of $TRUE$ regardless of the value inferred for RECOMMEND(`alice`, `grapefruit`). This ground rule would be considered trivial and rejected by most SRL framework's ground rule validation function.

The procedure for creating ground rules from rules generally falls into two categories: top-down and bottom-up. *Top-down grounding* starts with the rules and seeks to find all the replacements for each variable with constants from the data source. Top-down grounding is simple to implement, as it reduces to nested loops, but generally regarded as inefficient, since it iterates through the cross product of variables in each rule. *Bottom-up* grounding starts at the data level and finds tuples that satisfy each rule. This is generally achieved with a database query[1]. Because bottom-up grounding leverages an existing database system, it is generally significantly faster than top-down grounding Niu et al. [2011]. Additionally, linking SRL grounding with database systems allows more opportunities for improving grounding through established database optimization techniques. For the remainder of this dissertation, all discussion of grounding techniques assume bottom-up grounding is used.

Grounding is SRL is done in two steps: creating a *grounding plan* and executing the grounding plan. A grounding plan G is a set of triples, where each tuple contains a query Q_i, referred to as a *grounding query*, a vector of k rules, $\boldsymbol{t_i}$ that Q_i can ground, and a corresponding vector of k mappings, $\boldsymbol{m_i}$, from the columns in Q_i to the variables in each rule of $\boldsymbol{t_i}$.

$$G = \{(Q_i, \boldsymbol{t_i}, \boldsymbol{m_i}), ...\} \tag{2.3}$$

Once a grounding plan, G, is created, it can be executed in various ways. The most common method is to run each query sequentially on a local machine, but work has been done on executing grounding plans in a distributed Magliacane et al. [2015] or out-of-core Srinivasan et al.

[1]When discussing queries, both relational algebra and SQL will be used. For reference material on relational algebra, database systems, and SQL I recommend Garcia-Molina et al. (2008).

[2020a] fashion.

For example, the grounding program used in Figure 2.6, may be:

$$G = (Q_1, (t_1), (m_I)), (Q_2, (t_2), (m_I))$$

where m_I is an identity function and

$$Q_1 = \rho_{B(User,Item1)}(Bought) \bowtie \rho_{S(Item1,Item2)}(SimilarTaste)$$
$$\bowtie \rho_{R(User,Item2)}(Recommend)$$
$$Q_2 = \rho_{B(User,Item1)}(Bought) \bowtie \rho_{S(Item1,Item2)}(SimilarLook)$$
$$\bowtie \rho_{R(User,Item2)}(Recommend)$$

2.3.1 Independent Grounding

The most commonly used strategy in the SRL community for generating a grounding plan is *independent grounding*. Independent grounding creates one grounding query for each rule by taking a natural join of each predicate instance used in the rule. This simple scheme makes independent grounding easy to implement and the variable mapping function trivial to create.

For example, independent grounding may produce the following grounding program for

Figure 2.6:

$$\text{IndependentGronding}(T, X, Y) = G_{ind}$$
$$= (Q_1, (t_1), (m_I)), (Q_2, (t_2), (m_I))$$

where m_I is an identity function and Q_1 (fully defined below) and Q_2[2] are the queries used to create the variable constant mappings for t_1 and t_2 respectively.

$Q_1 =$

```
SELECT
    B.Arg1 AS User, B.Arg2 AS Item1, S.Arg2 AS Item2
FROM
    Bought B
    JOIN SimilarTaste S ON B.Arg2 = S.Arg1
    JOIN Recommended R ON
        B.Arg1 = R.Arg1 AND S.Arg2 = R.Arg2
```

2.4 Probabilistic Soft Logic

Probabilistic Soft Logic (PSL) is the primary SRL framework used throughout this dissertation. As its underlying graphical model, PSL uses hinge-loss Markov random fields (HL-MRFs), a special class of the undirected graphical model following Definition 1, with the additional specifications:

- All variables in X and Y are real-valued in the domain $[0, 1]$.

[2] Q_2 uses the same query as Q_1, but replaces SIMILARTASTE with SIMILARLOOK.

- All ϕ are convex functions that emit a real-valued score in $[0, 1]$.

Given this definition, MAP inference is achieved by maximizing the sum of weighted potentials in the model:

$$\arg\max_{\boldsymbol{y}} \sum_{R \in \mathcal{R}} \sum_{r \in R} w_R \cdot \phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r) \tag{2.4}$$

The convex nature of the potentials makes MAP inference on an HL-MRF a convex optimization problem. Framing inference as a convex optimization problem allows PSL to provide an exact solution to continuous problems and an approximate answer to discrete (MAX SAT) problems with rounding guarantees Bach et al. [2015].

The continuous nature of the HL-MRF allows PSL to scale beyond what was previously feasible for SRL frameworks. Solving MAP inference as a convex optimization problem makes inference in PSL linear time with respect to the number of ground rules. This fast inference has allowed PSL to solve problems with tens of millions of ground rules in minutes Kouki et al. [2018]. PSL has also been shown to outperform industrial interior point methods Bach et al. [2017] as well as existing MLN implementations Pujara et al. [2013], Augustine and Getoor [2018].

2.4.1 Weight Learning in PSL

There are three primary approaches used to perform weight learning in PSL as discussed in Bach et al. (2013): Maximum Likelihood Estimation (MLE), Maximum Pseudolikelihood Estimation (MPLE), and Large-Margin Estimation (LME).

Maximum Likelihood Estimation (MLE)

This approach maximizes the log-likelihood function with respect to the weights of the rules based on the training data. Since all the potentials generated by a rule share the same weight, the density function in Definition 1 can be written as $\sum_{i=1}^{r}[w_i\Phi_i(\mathbf{x},\mathbf{y})]$ where r is the number of template rules, w_i represents the weight of the i^{th} rule, $\Phi_i = \sum_{j\in g_i}\phi_j(\mathbf{x},\mathbf{y})$, and g_i is a set of ground rules generated by the i^{th} rule. The partial derivative of the log of the likelihood function given in Definition 1 for PSL with respect to w_q, for $q \in \{1,\ldots,r\}$ is:

$$\frac{\partial logP(\mathbf{y}|\mathbf{x})}{\partial w_q} = \mathbb{E}_{\mathbf{w}}\Big[\Phi_q(\mathbf{x},\mathbf{y})\Big] - \Phi_q(\mathbf{x},\mathbf{y}) \tag{2.5}$$

where $\mathbf{w} = \{w_1,...,w_r\}$ and $\mathbb{E}_{\mathbf{w}}$ is expectation w.r.t. the weights. It is infeasible to compute the expectation, hence to make the learning tractable, a MAP approximation is used that replaces the expectation in the gradient with the corresponding values in the MAP state. This approach is a structured variant of the voted perceptron algorithm [Collins, 2002].

Maximum Pseudolikelihood Estimation (MPLE)

An alternative approach that maximizes the pseudolikelihood function, which is given by:

$$P^*(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P^*(y_i|MB(y_i),\mathbf{x}) \tag{2.6}$$

where n is the number of random variables and $MB(y_i)$ is the Markov blanket of y_i. Equation 2.6 is maximized using a gradient ascent based approach and the derivative of the log-pseudolikelihood

function with respect to w_q is given by:

$$\frac{\partial log P^*(\mathbf{y}|\mathbf{x})}{\partial w_q} = \sum_{i=1}^{n} \mathbb{E}_{y_i|MB}\Big[\sum_{j \in g_q : i \in \phi_j} \phi_j(\mathbf{x}, \mathbf{y})\Big] - \Phi_q(\mathbf{x}, \mathbf{y}) \tag{2.7}$$

where $i \in \phi_j$ implies that variable y_i participates in the potential ϕ_j. Using a Monte Carlo approach this derivative can be computed in linear time in the size of $\mathbf{y}$.

Large-Margin Estimation (LME)

This approach focuses on maximizing the MAP state rather than producing accurate probabilistic models. This approach uses the intuition that the ground-truth state $\mathbf{y}$ should have energy lower than any alternate state $\tilde{\mathbf{y}}$ by a large margin defined by a loss function L. The objective function to find the optimal set of weights is given by:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\xi \tag{2.8}$$

$$s.t : \mathbf{w}^T(\Phi(\tilde{\mathbf{y}}, \mathbf{x}) - \Phi(\mathbf{y}, \mathbf{x})) \leq -L(\mathbf{y}, \tilde{\mathbf{y}}) + \xi$$

where L is a loss function such as the L1 distance between $\mathbf{y}$ and $\tilde{\mathbf{y}}$, and ξ is a slack variable. Equation 2.8 is then solved by performing a large-margin estimation based on a cutting-plane approach for structural support vector machines [Joachims et al., 2009].

2.5 Markov Logic Networks

Markov Logic Networks (MLNs) is one of the most popular family of SRL frameworks. MLNs [Richardson and Domingos, 2006] use boolean logic and use discrete random variables.

Comparisons against MLNs are made in Section 5.1 and Section 6.1. Potential functions ϕ are indicator functions that are one if a ground rule is satisfied and zero otherwise. A potential in MLN is of the form:

$$\phi_i(\mathbf{x}, \mathbf{y}) = n_i(\mathbf{x}, \mathbf{y}) \tag{2.9}$$

where $n_i(\cdot, \cdot)$ is the satisfaction of the i^{th} ground rule. Since the potentials measure the satisfaction of the ground rules, the $\hat{s}$ is set to one and the MAP inference in MLN is written as:

$$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \mathbf{E}(\mathbf{y}|\mathbf{x}) \tag{2.10}$$

Many different efficient implementations of MLNs exist [Shavlik and Natarajan, 2009a, Niu et al., 2011, Venugopal et al., 2016, Islam et al., 2018], but throughout this dissertation I choose to use Tuffy [Niu et al., 2011] as my representative MLN framework. MAP inference in Tuffy is performed using the MaxWalkSAT algorithm [Kautz et al., 1996, Niu et al., 2011]. MaxWalkSAT proceeds by iteratively flipping atom values that would result in a more satisfied world.

Many weight learning approaches have been proposed for MLNs [Richardson and Domingos, 2006, Singla and Domingos, 2005, Lowd and Domingos, 2007]. It has been observed that second-order methods for weight learning tend to perform better than other approaches [Lowd and Domingos, 2007]. Tuffy uses a second-order method, the diagonal Newton approach introduced in [Lowd and Domingos, 2007], to perform weight learning.

2.5.1 Diagonal Newton Method for Weight Learning (DN)

This approach minimizes the negative conditional log-likelihood (CLL) using a Newton-based approach. A Newton-based approach reaches global minimum by multiplying the gradient with the inverse of the Hessian at every iteration:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}\mathbf{g}$$

where $\mathbf{H}$ is the Hessian, $\mathbf{g}$ is the gradient w.r.t. $\mathbf{w}$, and t is the iteration number. Since computing the Hessian can be expensive and infeasible, a diagonal Newton method is used as an approximation for the Hessian. The Hessian of the negative CLL is the covariance matrix of the CLL, this is approximated through samples generated using MC-SAT [Poon and Domingos, 2006]. The final update for the weight of a logical rule at each iteration is given by:

$$w_i = w_i - \alpha \frac{\mathbb{E}_{\mathbf{w}}(N_i) - N_i}{\mathbb{E}_{\mathbf{w}}(N_i^2) - (\mathbb{E}_{\mathbf{w}}(N_i))^2} \tag{2.11}$$

where $N_i = \sum_{j \in g_i} n_j(\mathbf{x}, \mathbf{y})$, g_i is a set of ground rules generated by the i^{th} rule, $\mathbb{E}_{\mathbf{w}}$ is expectation w.r.t. the weights and α is the step size.

Chapter 3

Scalability

Scalability is a key pillar for practical machine learning systems. If a system is slow or has no potential to scale to larger problems, then it will see limited adoption by the machine learning community. Scalability in SRL is particularly difficult because of two challenges: dealing with the interconnected nature of data in SRL and the joint reasoning over large graphical models. In this chapter, I will address both of these challenges in the form of *grounding* and *inference*[1].

3.1 Tandem Inference

SRL methods have seen a great deal of success, but they face a major challenge when scaling to large datasets. To address this issue, several approaches have been proposed. *Lifted inference* de Salvo Braz et al. [2005], Singla and Domingos [2008], Kersting [2012], Sarkhel et al.

[1]The work presented in this chapter was presented at the 2023 International Conference on Very Large Data Bases (VLDB) Augustine and Getoor [2023] and at the 2019 AAAI Conference on Artificial Intelligence (AAAI) Srinivasan et al. [2020a].

[2014], Kimmig et al. [2015], Das et al. [2016], Srinivasan et al. [2019a] is a commonly employed and effective approach that exploits symmetry in the data to generate a more compact model on which to perform inference. A key drawback of these approaches is that evidence or noisy data can break symmetries, making lifting less effective. To address this issue, *approximate lifting* approaches have also been proposed Sen et al. [2009], den Broeck et al. [2012], Venugopal and Gogate [2014], Das et al. [2019] which exploit approximate symmetries, allowing for greater and more robust compression. However, if the ground model lacks significant symmetry, even approximate lifting may not improve the tractability of inference on large models.

Several approaches orthogonal to lifted inference have also been proposed that attempt to perform efficient grounding by utilizing hybrid database approaches Niu et al. [2011], exploiting the structure of rules Augustine and Getoor [2018], or distributing models across multiple machines Magliacane et al. [2015]. However these methods only provide partial solutions to grounding at scale. The hybrid database approach increases runtime substantially, exploiting rule structure requires large amounts of memory to store the ground model, and distributing across several machines does not reduce the overall memory required to run a large program.

In this section, I propose an alternate approach to scaling which performs inference in tandem with grounding. This enables us to scale SRL systems to large, previously intractable, models. My approach, which I refer to as *tandem inference* (TI), uses a novel streaming grounding architecture and an out-of-core inference algorithm that utilizes a disk cache in order to consume a fixed amount of memory. This allows TI to scale unbounded by a machine's main memory. Furthermore, even with increased I/O overhead, my approach runs the entire process of grounding

and inference in a fraction of the runtime required by traditional approaches. Since TI is orthogonal to lifting and some of the other strategies, it can be combined with them for further improvements.

The TI concept is general and can potentially be applied to several different SRL frameworks. In this section, I show how it can be implemented in PSL. PSL is a SRL framework that generates a special kind of undirected graphical model called a hinge-loss Markov random field (HL-MRF). A key distinguishing factor of a HL-MRF is that it makes a continuous relaxation on RVs which transforms the inference problem into a convex optimization problem. This allows PSL to use optimizers such as alternating direction method of multipliers (ADMM) Boyd et al. [2010] to perform efficient inference.

My key contributions to this component of the scalability pillar include: 1) I propose a general framework, TI, which uses streaming grounding and out-of-core streaming inference to perform memory efficient, large-scale inference in SRL frameworks; 2) I derive a stochastic gradient descent-based inference method (SGD) and show that the SGD-based method can outperform the traditionally used ADMM-based method; 3) I develop an efficient streaming grounding architecture and SGD-based out-of-core inference system that runs faster than previous state-of-the-art systems; 4) through experiments on two large models, FRIENDSHIP-500M and FRIENDSHIP-1B, which require over 400GB and 800GB of memory respectively, I show that, using just 10GB of memory, I can perform inference on FRIENDSHIP-500M in under four hours and FRIENDSHIP-1B in under nine hours; and 5) I perform an empirical evaluation on eight realworld datasets to validate the speed and accuracy of TI. In addition to enabling inference on models too large to fit into memory, on my largest realworld dataset which does fit in memory, TI is 8 times faster than traditional approaches.

The work covered in this section was presented at the 2019 AAAI Conference on Artificial Intelligence (AAAI) Srinivasan et al. [2020a].

3.1.1 Tandem Inference Methodology

In order to define my proposed tandem inference (TI) algorithm I introduce two components: the *grounding generator* (GG) and the *inference engine* (IE). The GG supports *streaming grounding*, which is the process of generating ground rules in small batches without materializing all grounding results into memory. The IE supports *streaming inference*, which is the process of performing inference using a single potential at a time. Figure 3.1 shows the system architecture of TI. The GG takes as input the data $\mathcal{D}$ and the model $\mathcal{M}$, which GG uses to generate the ground model. With respect to storage, the GG can leverage the hard disk, the database, and RAM, while the IE can only utilize RAM.

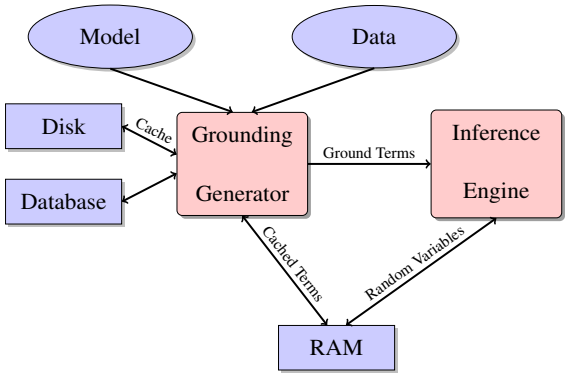


Figure 3.1: Block diagram showing the TI system architecture.

The process flow of TI is shown in the network sequence diagrams given in Figure 3.2 (for the first round of inference) and 3.3 (for subsequent rounds of inference). The IE begins by

requesting a potential function (also called a *ground term*) from the GG. During the first round of inference, the GG utilizes the database to generate ground rules. Once a ground rule is created, it is converted into a ground term, written to a disk cache, and then passed onto the IE. On subsequent rounds of inference, the GG uses the disk cache alone to provide ground terms to the IE. As each term is returned from the GG, the IE will optimize the term and update any relevant RVs held in RAM. After all terms have been seen, IE will then begin a new round of inference until convergence criteria are met. Since the GG uses a fixed-size in-memory cache and the IE discards terms after use, there is a maximum amount of memory in use by TI at any given time.

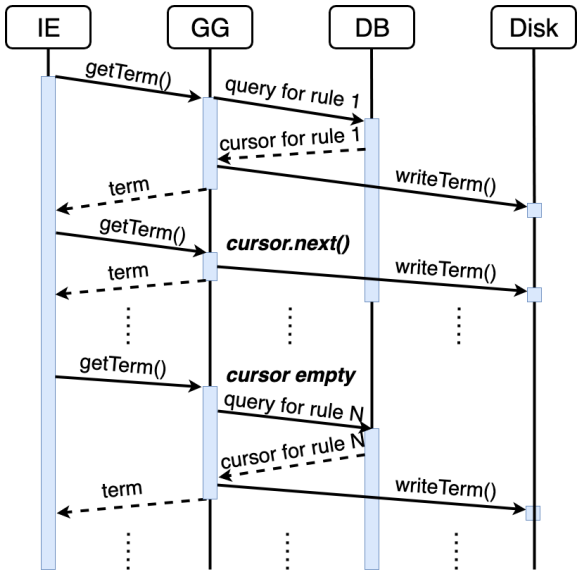


Figure 3.2: Network sequence diagram for iteration 1 of TI.

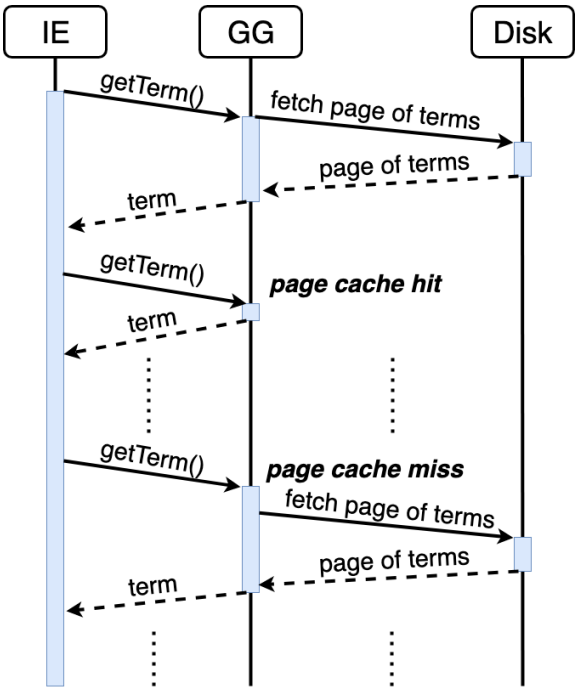


Figure 3.3: Network sequence diagram for iterations 2 through T of TI.

Streaming Grounding

Streaming grounding is the TI component that is responsible for providing ground terms one at a time to the IE. To support streaming grounding, the underlying SRL framework must be able to construct a single ground rule without instantiating large portions of the model, a process I will refer to as *partial grounding*. Constructing the full ground network is the SRL phase that is most prone to running out of memory, so it is imperative that this process can be broken up into small chunks. PSL is one of several SRL frameworks that supports *bottom-up grounding*Augustine and Getoor [2018], which frames the problem of grounding a rule as constructing and executing a SQL query. Relational database management systems (RDBMSs) have a built-in way to fetch only a portion of the query results through cursorsGarcia-Molina et al. [2008]. A cursor works as an iterator for a query's results. If the RDBMS and content of the SQL query allows for it, cursors can return results as they are generated by the query instead of waiting for the full query to complete. Cases that force a database to materialize all results before returning any records include sorting or deduplicating the results, both of which are avoided in PSL grounding queries.

During the initial iteration of streaming grounding, the database must be queried to fetch the ground terms. The process described here is also shown as a network sequence diagram in Figure 3.2. The rules in the model, $\mathcal{M}$, will be iterated over until all have been grounded. When asked for a term, the GG will first check if it has an open database cursor. If there is no cursor or the current cursor has been exhausted, then the database will be queried for the next rule and a new cursor will be constructed. If all rules have been grounded, then the GG will inform the IE that there

are no more ground terms for this iteration. With an open cursor, the next result tuple will be fetched. The tuple will then be instantiated into a ground rule and checked for triviality. A ground rule is trivial if it will not affect the result of inference; for example, a ground rule with a logical expression that is already satisfied by observed variables is considered trivial. If the newly instantiated ground rule is invalid, the process is repeated with the next result from the cursor until a valid ground rule is instantiated. After a ground rule is validated, it is converted into a ground term. This term is then put into a cache buffer and eventually passed on to the IE. Once the cache buffer is full, or there are no more ground rules, it is written to disk.

After the initial iteration of TI, ground terms are fetched from the disk cache in the order they were written during the initial iteration of streaming grounding. The process described here is shown as a network sequence diagram in Figure 3.3. The disk cache is written as several pages, one page to a file. The number of terms written to a page can be configured and the effect of this configuration is explored in Section 3.1.2. When asked for a term, the GG will first ensure that a cache page is loaded. If the current page has been exhausted and there are no more pages, then the IE will be informed that there are no more ground terms for this iteration. Each page is written in binary to minimize I/O. The first 16 bytes of a page contains the number of terms written to the page and the size in bytes of all the terms in the page. Because each term may contain a different number of RVs, the exact size of each term is not known until it is generated. Each term is then read into a preallocated term structure. A free list of available terms large enough to fill a page is maintained to minimize memory allocations. After all terms have been read into the memory cache, the next term from the cache will be returned to the IE until the page is exhausted.

The GG also provides the ability to return ground terms in a semi-random order. The effectiveness of randomizing term order is dependent upon the optimization algorithm employed by the IE. During the initial iteration of streaming grounding, the order of the terms is dependent upon the order that results are returned from the database. However, during subsequent iterations, there are more opportunities to induce randomness. First, the order that pages are accessed can be randomized. Second, the terms in each page can be shuffled in-place after they have been read from disk. Although not fully random since every term is guaranteed to be no more than a page length away from other terms in the same page, these steps provide a good amount of randomness without increasing the number of I/O operations or memory required.

Streaming Inference

The second core component of TI is streaming inference. The open-source PSL implementation uses ADMM to minimize the convex objective produced by a HL-MRF. While ADMM is an efficient algorithm, it places a high memory requirement on the system. However, the primary goal of using TI is to alleviate any memory constraints for performing inference. ADMM in PSL works by creating Lagrange variables (LVs) and a local copy of the RVs (LRVs) for every potential. Every potential is optimized w.r.t. the LRVs and an average over all LRVs is taken to obtain a global assignment for the RVs. All ground rules, LRVs, and LVs are kept in memory to make this process efficient. However, the need to keep all this information in memory makes ADMM less than ideal for streaming inference. To replace ADMM, I propose a gradient-based optimization to solve Equation 2.4.

Stochastic Gradient Descent for PSL

Consider the following energy function:

Definition 2 (Hinge-loss Markov Random Field Energy Function)**.** *Let* $\mathbf{y} = \{y_1, y_2, ..., y_n\}$ *be* n *RVs,* $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ *be* m *observed variables or evidence, and* $\phi = \{\phi_1, \phi_2, ..., \phi_K\}$ *be* K *potential functions such that* $\phi_i(\mathbf{x}, \mathbf{y}) = \max(\ell_i(\mathbf{x}, \mathbf{y}), 0)^{d_i}$ *or* $\ell_i(\mathbf{x}, \mathbf{y})^{d_i}$*. Where* ℓ_i *is a linear function and* $d_i \in \{1, 2\}$ *provides a choice of two different loss functions,* $d_i = 1$ *(i.e., linear) and* $d_i = 2$ *(i.e, quadratic). For weights* $\mathbf{w} \in \{w_1, w_2, ..., w_K\}$ *a hinge-loss energy function can be defined as:*

$$\mathbf{E}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{K} w_i \phi_i(\mathbf{x}, \mathbf{y}) \text{ ; s.t.,} \mathbf{y} \in [0, 1] \text{ ; } \mathbf{x} \in [0, 1] \tag{3.1}$$

and the HL-MRF is defined as:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{y})} exp(-E(\mathbf{y}|\mathbf{x})) \tag{3.2}$$

where $Z(\mathbf{y}) = \int_{\mathbf{y}} exp(-E(\mathbf{y}|\mathbf{x}))$*.*

To facilitate easy gradient computation, this can be re-written as:

$$E(\mathbf{y}|\mathbf{c}) = \sum_{i=1}^{K} w_i \phi_i(\mathbf{y}, c_i) \tag{3.3}$$

$$\phi_i(\mathbf{y}, c_i) = \begin{cases} (\mathbf{y}^T \mathbf{q}_i - c_i)^{d_i} & \text{if linear loss} \\ \max((\mathbf{y}^T \mathbf{q}_i - c_i), 0)^{d_i} & \text{otherwise} \end{cases}$$

$$c_i = \mathbf{x}^T \dot{\mathbf{q}}_i$$

where $\mathbf{q}_i \in \{0, 1, -1\}^n$ and $\dot{\mathbf{q}}_i \in \{0, 1, -1\}^m$ are respective n-dimensional and m-dimensional ternary vectors which indicates if a variable participates positively, negatively, or not at all in potential function ϕ_i. The scalar c_i incorporates the information of all the observed variables participating in ϕ_i and I use $\mathbf{c}$ to represent the vector of scalars c_i. As a reminder, in full gradient descent (GD), I iteratively optimize by taking weighted steps in the direction of the energy function's (Equation 3.3) gradient until convergence or for T steps. The weight of the steps is determined by the learning rate, η. Additionally for PSL, Equation 3.1 has a restriction that $\mathbf{y} \in [0, 1]^n$, therefore after each step I need to project $\mathbf{y}$ back in to the box $[0, 1]$ through truncation. The gradient step update at every step t can be represented as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) \tag{3.4}$$

$$\mathbf{y}_t = \min(\max(\mathbf{y}_t, 0), 1) \tag{3.5}$$

$$\text{where } \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) = \frac{1}{K} \sum_{i=1}^{K} w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i)$$

The gradient computation for the energy function involves computing projected gradients for the potential functions. The projected gradients can be written as:

$$\nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) = \begin{cases} 0 & \text{if hinge \& } \mathbf{y}^T \mathbf{x}_i \leq c_i \\ w_i \mathbf{q}_i & \text{if } d_i = 1 \\ 2 w_i \mathbf{q}_i (\mathbf{y}^T \mathbf{q}_i - c_i) & \text{otherwise} \end{cases} \tag{3.6}$$

Using the above equations, I can compute the gradient update and run the process to convergence. Since the objective is convex, the right choice of η will guarantee convergence. However, an issue

with GD is that at every step it needs to compute the full gradient, requiring all terms. This is expensive and does not support my streaming approach. To make it more compatible with my streaming approach, I use stochastic gradient descent (SGD). In SGD, the gradient is computed w.r.t. a single potential ϕ_i and an update to the variables can be made without examining all terms.

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) \tag{3.7}$$

The variables are then projected as in Equation 3.5. This update is aligned with my streaming approach, and allows the IE to request a single potential, perform an update on the participating RVs, and continue on to the next potential.

An important factor to make SGD work in practice is the correct choice of η, and many approaches have been proposed to compute adaptive learning rates Ruder [2016]. One of the more popular and successful methods for adaptive learning rates is SGD-ADAM Kingma and Lei Ba [2015]. Here, I investigate three approaches: 1) using SGD-ADAM, 2) using a time-decaying learning rate (i.e., $\eta = \frac{\eta}{t}$), and 3) using a constant learning rate η. In my experiments, a time-decaying learning rate is more effective than more complicated mechanisms such as SGD-ADAM. Finally, the overall process of performing non-streaming inference using SGD in PSL is shown in Algorithm 1.

Now that I have a streaming grounding infrastructure and the SGD-based PSL IE, I can perform TI. The algorithm for TI is the same as Algorithm 1, except that the potentials are read from the GG instead of directly from memory.

Algorithm 1: SGD for PSL

Data: list of ground terms $\phi = \{\phi_1, \phi_2, ..., \phi_K\}$

Result: RVs $\mathbf{y}$

$\eta =$ learning rate;

$\mathbf{y} \sim Unif(0,1)^n$;

$t = 1$;

while *not converged and* $t <= T$ **do**

  for $i \in \{1 \dots K\}$ **do**

    Update $\mathbf{y}$ using ϕ_i with Equation 3.7;

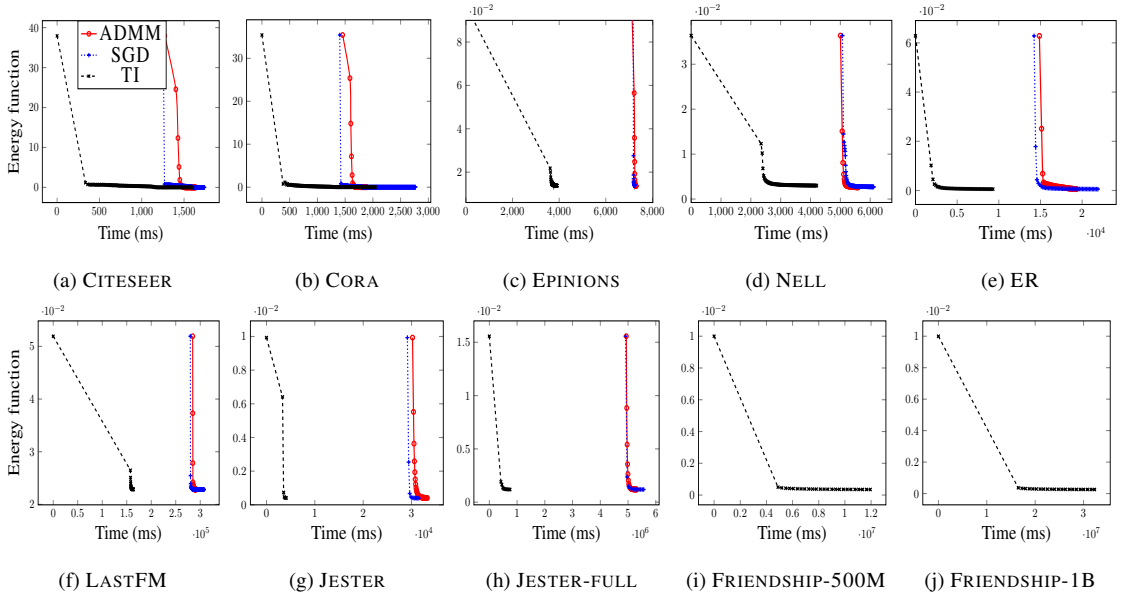
    $\mathbf{y} = \min(\max(\mathbf{y}, 0), 1)$;

  end

  $t = t + 1$;

  Update η;

end



(a) CITESEER (b) CORA (c) EPINIONS (d) NELL (e) ER

(f) LASTFM (g) JESTER (h) JESTER-FULL (i) FRIENDSHIP-500M (j) FRIENDSHIP-1B

Figure 3.4: Comparison of the runtimes of TI, ADMM, and SGD on 10 datasets.

3.1.2 Empirical Evaluation

In this section, I evaluate the performance of my proposed method on variety of realworld and two large synthetic datasets. I answer the following questions: Q1) Can I perform inference on large ground models that were previously intractable? Q2) Is streaming faster than and as accurate as traditional inference? Q3) How much memory does TI use? Q4) Can a gradient-based optimizer converge faster than ADMM? Q5) What is the best strategy for selecting the learning rate? I answer Q1 and Q2 in Section 3.1.2, Q3 in Section 3.1.2, and Q4 and Q5 in Section 3.1.2. For all my experiments, I set the max number of iterations, T, to 500, a convergence tolerance of 10^{-6}, and a machine with 400GB of memory.

I perform my experiments on eight realworld datasets and two synthetic datasets from a

data generator previously used to test scaling in PSL Augustine and Getoor [2018][2]. The details of the datasets are as follows:

CITESEER: a collective classification dataset with 2,708 scientific documents, seven document categories, and 5,429 directed citation links.

CORA: a collective classification dataset with 3,312 documents, six categories, and 4,591 directed citation links.

EPINIONS: a trust prediction dataset with 2,000 users and 8,675 directed links which represent positive and negative trust between users.

NELL: a knowledge graph construction dataset originally derived from the NELL project with 27,843 entity labels and 12,438 relations.

ER: an entity resolution dataset with a citation network of 1136 authors references and 864 paper references.

LASTFM: an artist recommendation dataset with 1,892 users, 17,632 artists, 92,834 user-artist ratings, and 12,717 friendship links.

JESTER: a joke recommendation dataset with 2,000 users, 100 jokes, and 200,000 user-joke ratings, sampled from the larger JESTER-FULL dataset.

JESTER-FULL: the full Jester dataset. Contains 73,421 users, 100 jokes, and 7.3M user-joke ratings. To the best of my knowledge, this is the first time the full Jester dataset has been used in with an SRL framework.

FRIENDSHIP-500M: a synthetic link prediction dataset with 2,000 users and 4M unobserved edges.

[2]Models, data, and code will be made public upon acceptance.

Dataset	Rules	Ground Rules	Random Variables	Memory (GB)	Source
CITESEER	10	36K	10K	0.10	Bach et al. (2017)
CORA	10	41K	10K	0.11	Bach et al. (2017)
EPINIONS	20	14K	1K	0.12	Bach et al. (2017)
NELL	26	91K	24K	0.13	Pujara et al. (2013)
ER	9	541K	485K	0.24	Bhattacharya and Getoor (2007)
LASTFM	22	1.4M	18K	0.45	Kouki et al. (2015)
JESTER	7	1M	50K	0.49	Bach et al. (2017)
JESTER-FULL	8	110M	3.6M	110	Goldberg et al. (2001)
FRIENDSHIP-500M	4	500M	4M	400+	Augustine and Getoor (2018)
FRIENDSHIP-1B	4	1B	7.6M	800+	Augustine and Getoor (2018)

Table 3.1: Details of models used and their memory consumption for non-streaming inference. Memory usage for FRIENDSHIP-500M and FRIENDSHIP-1B are estimates. The machine has only 400GB of memory and was unable to fully run FRIENDSHIP-500M and FRIENDSHIP-1B, so I had to estimate the memory usage.

FRIENDSHIP-1B: similar to FRIENDSHIP-500M containing 2,750 users and 7.5M unobserved edges.

Table 3.1 provides details on number of rules in each model, the number of ground rules generated, and the amount of memory required to hold each model in memory.

Scale, Speed, and Convergence

I begin by examining the inference time and convergence of TI, SGD, and ADMM on all ten datasets (Q1 & Q2). Weights for the rules in each model are learned and re-scaled to be in the range $[0, 1]$. Both SGD and TI use a time-decayed learning rate and the initial learning rate η needs to be tuned. For the LASTFM, FRIENDSHIP-500M, and FRIENDSHIP-1B datasets I use $\eta = 0.1$, for ER I use $\eta = 10$, and for all the other datasets I use $\eta = 1.0$. The rational for choosing these learning rates is explained in Section 3.1.2. TI also has a page size which can be tuned based on the amount of memory available. Since the machine has 400GB RAM, I choose a page size of 10M for all datasets, which uses about 10GB of memory. I discuss further details about trade-offs in page size, memory, and computation in Section 3.1.2.

Scaling to Large Datasets: Figure 3.4i and 3.4j show the inference convergence w.r.t. time (in milliseconds) for FRIENDSHIP-500M and FRIENDSHIP-1B. TI was able to run the FRIENDSHIP-500M dataset in under four hours using only 10GB of memory. Both SGD and ADMM exhausted the 400GB of memory available on the machine and failed to run. Similarly, the FRIENDSHIP-1B dataset, which I estimate to require more than 800GB to hold in memory, was able to run on the same machine in under nine hours using only 10GB of memory. These results answer Q1 affirmatively, TI can successfully perform inference on large ground models that were previously intractable.

Speed and Convergence: In order to address Q2, Figure 3.4 shows the inference convergence for all datasets using all three approaches. The time shown includes both the grounding and inference phases (which happen together in TI). In all datasets except CITESEER and CORA, TI converges before the other methods even fully finish grounding! In CITESEER and CORA, possibly due to some rules with high weights, tuning the learning rate is difficult, and, after the first steep drop, SGD and TI take many more iterations to converge compared to ADMM. As the ground model size increases, more significant timing differences. In EPINIONS, ER, and LASTFM, TI finishes the entire process of grounding and inference in just half the time taken by ADMM and SGD. For JESTER, TI is over 5 times faster than both ADMM and SGD. For my largest realworld dataset, JESTER-FULL, TI is about 8 times faster than both ADMM and SGD. This shows that TI is faster than traditional approaches especially on larger datasets and converges to the same function value.

Memory Efficiency

To answer Q3, how much memory TI uses, and test the effect of page size, I run TI on the JESTER-FULL dataset with page sizes between 10 and 1M potentials. I run SGD to establish baseline behavior. Since SGD holds all components in memory, I consider it to have an infinite page size. I measure the maximum memory usage during the entire run, the mean number of I/O operations performed in a single iteration of optimization, and the mean time to complete a single iteration of optimization. Because PSL is written in Java, the memory usage I report is the size of the JVM's heap. I/O operations are measured by the number of calls made to Java's low-level I/O methods `FileInputStream.read` and `FileOutputStream.write`. All reported values

are averaged over 10 runs.

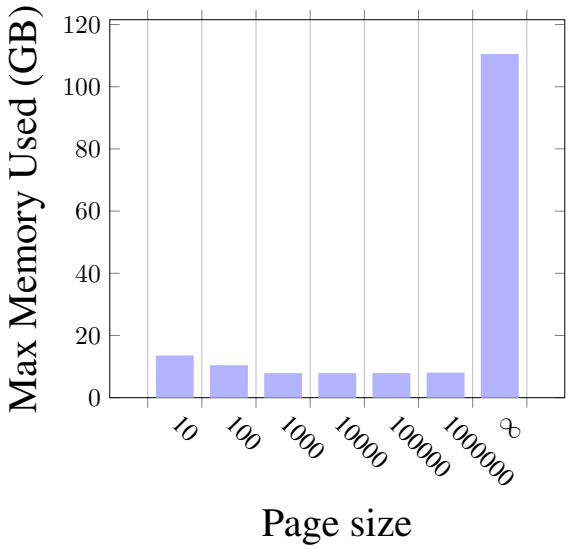


Figure 3.5: Maximum memory usage for TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages.

Figure 3.5 shows the peak memory usage over several runs using different page sizes for TI. Naively, I expect the amount of memory used to decrease with the page size: the fewer ground terms held in memory, the less overall memory is used. However, instead very small pages sizes (10 and 100) lead to more memory being used. To understand why, I must remember that Java is a garbage collected language and that memory marked for garbage collection will still count as being in use. The small page sizes cause many I/O operations to happen in quick succession. Every I/O operation requires Java to allocate memory buffers used in that operation. Therefore, the discarded buffers are eventually cleaned up but not before being counted in the memory total. A native language like C that can directly make system calls and that does not have to go through a virtual machine can avoid these extra allocations and inflated memory cost. Forcing the JVM to

garbage collect more frequently[3] shows a more consistent memory usage over all page sizes. This is because all the page sizes still fit into memory and Java will continue to accumulate memory until a point where garbage collection is triggered. This point depends on the maximum size of the heap and is common among all the runs. From this experiment I can conclude that although using a smaller page size will use less persistent memory, the JVM's garbage collector will keep most reasonable page sizes at the same memory usage levels.

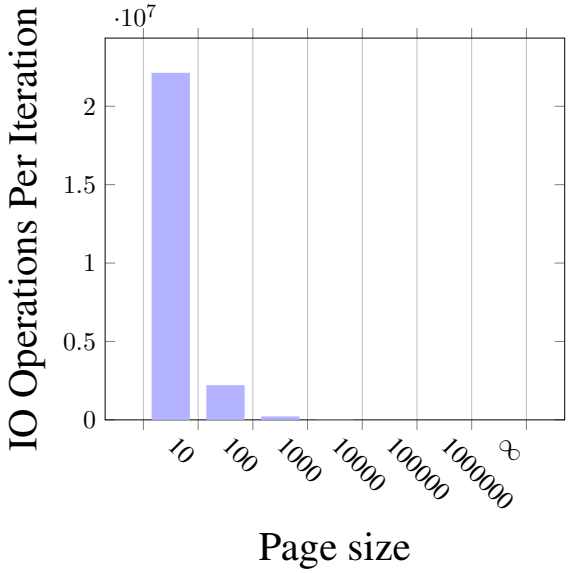


Figure 3.6: Number of I/O operations per optimization iteration of TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages.

Figure 3.6 shows the number of I/O operations per optimization iteration. SGD does not perform any I/O operations. As the page size increases, the number of I/O operations decreases. The impact of these additional I/O operations for a small page sizes can be seen in Figure 3.7

[3]A smaller value for the JVM parameter `XX:NewRatio` is used to force more frequent garbage collection.

(per-iteration runtime). The different page sizes result in approximately the same number of bytes read from disk and since the number of potentials is the same, the time spent in optimization will also be the approximately the same. However, the overhead of the additional I/O operations causes substantially slower iterations for page sizes of 10 and 100. For larger page sizes (1000+), the difference in I/O overhead becomes negligible and all have similar per iteration runtime.

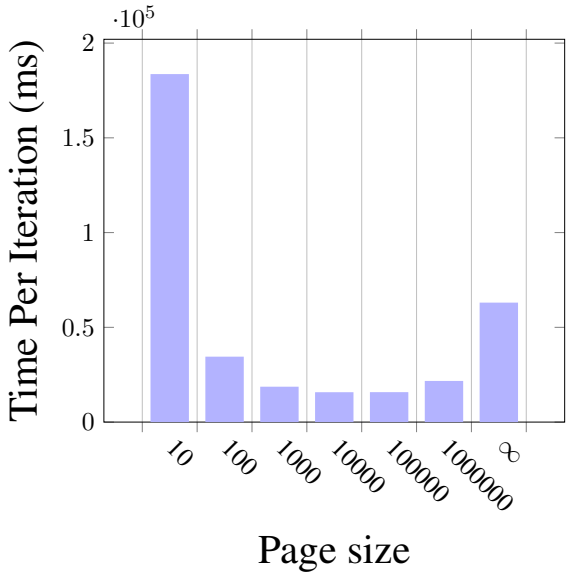


Figure 3.7: Runtime per optimization iteration of TI over multiple page sizes on the JESTER-FULL dataset w.r.t. page size. The page size listed as ∞ is run with SGD, which does not use pages.

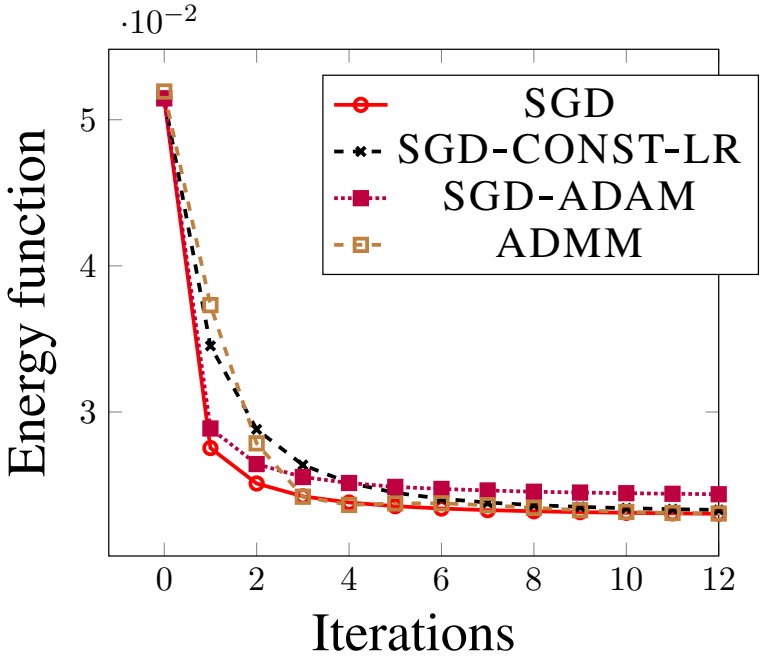
Optimizer Efficiency and Learning Rate

To answer Q4, can SGD converge as fast as ADMM, and Q5, how to choose learning rate, I run experiments on the LASTFM and JESTER datasets. The results here extend to other datasets. Here, I compare four different approaches: SGD with a decaying learning rate (SGD), SGD with a constant learning rate (SGD-CONST-LR), SGD with an adaptive learning rate (SGD-ADAM), and

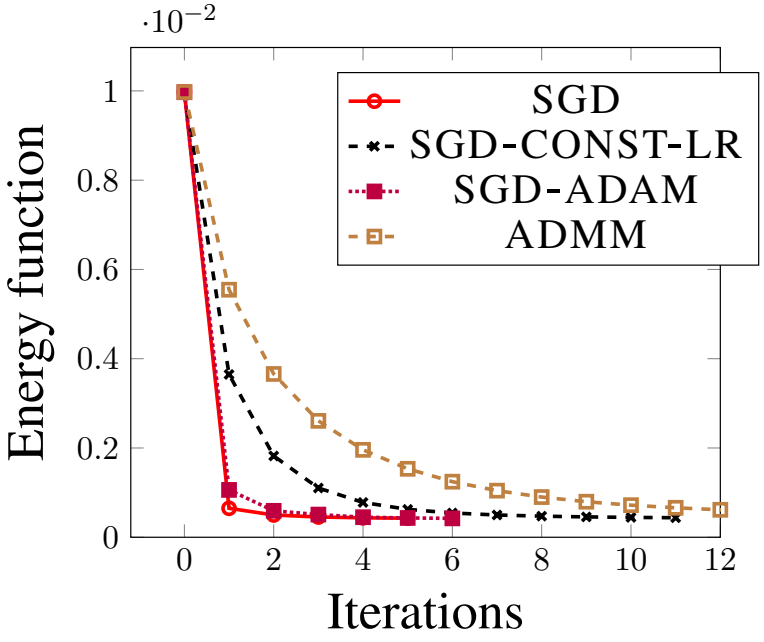
ADMM.

SGD vs. ADMM: Figure 3.8 shows the convergence of different approaches w.r.t. number of iterations for the LASTFM and JESTER datasets. SGD and ADMM converge to the same function value in different number of iterations. Typically, SGD takes fewer iterations to converge than ADMM. However, this is heavily dependent on the learning rate chosen for SGD. If one cannot find the right learning rate, then it is possible for SGD to take significantly more iterations than ADMM.

Choice of Learning Rate: From Figure 3.8 SGD-CONST-LR is the slowest to converge, and there seems to be little difference between SGD which uses time-decayed learning rate and SGD-ADAM which uses an adaptive learning rate. SGD and SGD-CONST-LR have an initial learning rate to be chosen. This can be chosen in range $\eta \in [\frac{1}{10\max(\mathbf{w})}, \frac{10}{\max(\mathbf{w})}]$ for SGD, and for SGD-CONST-LR, $\eta \in [\frac{1}{100\max(\mathbf{w})}, \frac{100}{\max(\mathbf{w})}]$. Thus, I choose an η of 1.0 and 0.1 for JESTER and LASTFM respectively for SGD, and η of 0.01 for SGD-CONST-LR. SGD-ADAM has four hyperparameter α, $\beta 1$, $\beta 2$, and ϵ to tune. This makes it harder to get ideal performance with SGD-ADAM. Further, SGD-ADAM uses additional parameters equal to three times the number of RVs to perform adaptive tuning. For my experiments, I choose $\alpha = 0.01$, and use $\beta 1 = 0.9$, $\beta 2 = 0.999$, and $\epsilon = 10^{-8}$ as suggested by Kingma and Lei Ba [2015]. From my evaluation I conclude that the simpler strategy, SGD with decaying learning rate performs just as well as SGD-ADAM, the more complicated adaptive strategy.



(a) LASTFM dataset.



(b) JESTER dataset.

Figure 3.8: The effect of different optimizers on convergence.

3.2 Collective Grounding

Combining logic and uncertainty has been a long-standing effort throughout computer science. In the database community this effort is typified by probabilistic databases, which extend traditional database management systems with uncertain data. While in the machine learning community, this effort is embodied by SRL. While the types of queries commonly made over probabilistic databases and SRL differ, they are similar in that they involve the same core task of inference over a probability space of possible worlds Dalvi et al. [2009]. Additionally, tuple-independent probabilistic databases have been shown to be a special case of SRL models Gribkoff and Suciu [2016]. Therefore, SRL models can be seen as probabilistic databases without tuple-independence assumptions and more complex hard and soft constraints.

However, abandoning these independence assumptions makes SRL models significantly more computationally intensive than tuple-independent probabilistic databases, specifically, with respect to grounding Schoenfisch and Stuckenschmidt [2017], Bremen et al. [2019], Tsamoura et al. [2020]. Similar to *the chase* in data integration/exchange Alhazmi et al. [2022] or the *immediate consequence operator* in Datalog, grounding starts with the provided data and iteratively instantiates more data using the provided logical formulae. However, unlike the chase and the immediate consequence operator, the result that grounding produces are complex dependencies between random variables. These complex dependencies makes the grounding done in SRL uniquely challenging, however, these complex dependencies also provide structure that can be exploited to improve groundings efficiency. In this section, I show how the structure provided by the problem can

be used to improve upon state-of-the-art SRL grounding by focusing on using database techniques to reduce the grounding overhead in a way that is accessible to a wide range of systems.

Most effective SRL grounding techniques rely on an underlying relational database. A query is constructed for each logical rule and each result tuple is a ground rule that corresponds to a factor in the graphical model. This approach is referred to as *bottom-up grounding* Niu et al. [2011] because it starts instantiating the model at the data level, such as in Datalog, as opposed to *top-down* grounding where logical formulae are iterated over using nested loops, such as in Alchemy and Prolog. Bottom-up grounding greatly improves upon top-down grounding, but still presents several challenges: logical formulae may be complex and map to similarly complex and slow database queries, queries may generate redundant or logically trivial results, and processing the logical formulae independently creates duplicate work for rules that share similar logical structures.

Several approaches have been proposed to deal with grounding complexity. Lifting Van den Broeck et al. [2021] and forward reasoning Tsamoura et al. [2020] techniques attempt to mitigate the effect of grounding by preforming calculations before grounding to eliminate less relevant factors and reduce the size of the ground program. These techniques are effective on models with a high degree of symmetry, but are less effective in more heterogeneous settings and may require multiple passes over the data. Another approach to reducing the grounding bottleneck is to distribute the burden of grounding over multiple machines Magliacane et al. [2015]. This approach shows promise, but shares the same challenges as any horizontal scaling effort, namely data availability, machine availability, setup time, and complexity for the average user. Additionally, the joint nature of these models makes them difficult to distribute.

In this section, I introduce a novel approach to tacking the bottleneck of grounding by treating grounding not as multiple independent workloads, but as a single workload to be jointly optimized. I refer to my approach as *collective grounding* (CG). Collective grounding leverages the structure provided by logical rules in addition to borrowing from established database research in query rewriting, query containment, and multi-query optimization to address three key challenges in collective grounding: 1) generating candidate grounding queries, 2) verifying logical rule satisfaction, and 3) computing a minimal grounding workload.

My key contributions to this component of the scalability pillar include:

1. I formalize the concept of grounding through the introduction of grounding plans.
2. I introduce the concept of collective grounding for templated graphical models.
3. I develop a method for generating and searching through alternate execution paths for grounding templates.
4. I develop an efficient method for computing a minimal shared grounding workload.
5. I perform an extensive empirical evaluation of collective grounding over multiple datasets and show that collective grounding can provide a 5x speedup over traditional grounding approaches.

The work covered in this section was presented at the 2023 International Conference on Very Large Data Bases (VLDB) Augustine and Getoor [2023].

3.2.1 Collective Grounding Methodology

Collective Grounding aims to speed up grounding by creating grounding plans that share database queries between multiple templates. More specifically, the goal of collective grounding is to compute an optimal grounding plan that minimizes the total time spent grounding a collection of rule templates T with data $D = (X, Y)$:

$$G^* = \arg\min_{G} \sum_{Q \in G^*} \text{Runtime}(Q(D)) \tag{3.8}$$

Unlike independent grounding, collective grounding looks at grounding not as a series of independent steps, but as a unified process. The procedure of collective grounding is composed of three main steps: *candidate generation*, *containment map construction*, and *coverage selection*. Candidate generation creates alternate grounding queries, referred to as *candidates*, that can be used in lieu of the original grounding queries executed in independent grounding. Containment map construction identifies the rules each of the newly generated candidates can ground and constructs variable mappings for each viable rule/candidate pair. Finally, coverage selection chooses the set of candidates that will cover the grounding needs for all templates in the final grounding plan. An overview of the collective grounding process is illustrated in Figure 3.9. The remainder of this section discusses the details for each of these three main steps. Pseudocode for collective grounding in included in Appendix A.1.1.

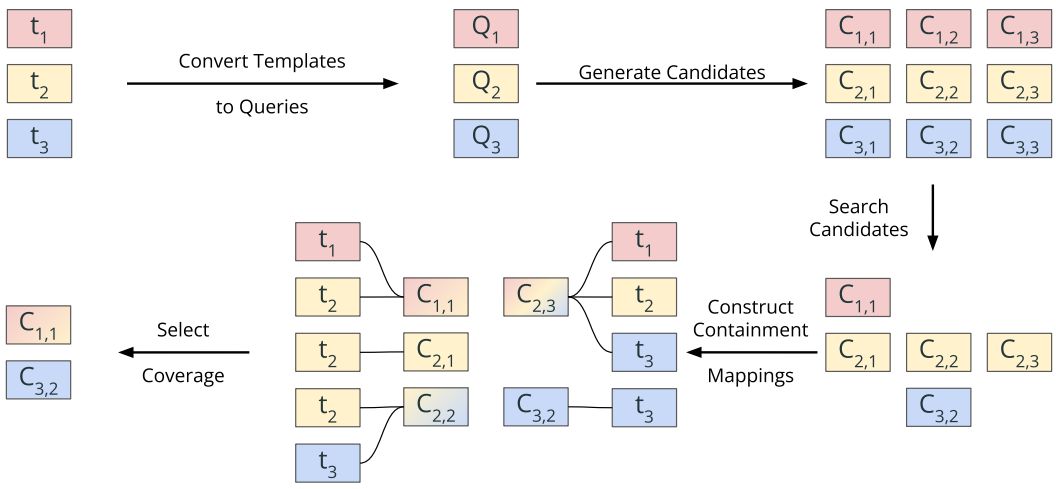


Figure 3.9: An example of how the collective grounding process converts relational templates (t_1, t_2, t_3) into a grounding plan. Colors are used to represent the rule that queries and candidates satisfy, color gradients indicate that multiple templates are satisfied.

Candidate Generation

The first step in collective grounding is to generate multiple possible queries that can be used to ground each rule. I refer to these possible replacement queries as *candidate queries*, or simply *candidates*, and the grounding queries from independent grounding as *base queries*. The goal of candidate generation is to generate multiple candidate queries that can produce the same ground rules as the base query while executing more efficiently when run collectively. Each candidate provides a different possible workload that can satisfy the same rule. For example, consider the candidate presented in Figure 3.10. The base query for t_1 produces two results, which both pass validation. A possible candidate for t_1, C_1, is a query that contains one less join and produces one additional result. The additional result tuple, however, is trivial and does not pass validation. Therefore, both the base query and candidate produce the same ground rules using

different workloads.

Strong similarities can be drawn between candidate generation and the very well studied problem of database query optimization. More specifically, generating candidate queries given a rule can be seen as a form of query rewriting. Like query rewriting, candidate generation starts with a base query (the independent grounding base query), and constructs a new query that can be executed more efficiently. In a classic RDBMS setting, the query optimizer can take advantage of indexes, table statistics, and knowledge of the system the query is being performed on (CPU, RAM, disk speed, etc). However, there is a very important difference between RDBMS query rewriting and collective grounding candidate generation: RDBMS query rewriting is limited by the constraint that a rewritten query must be equivalent to the base query. Collective grounding candidate generation, however, can take advantage of the structure inherent in the problem, specifically the validation function V, to simplify the process of candidate generation.

Because "invalid" query results can be discarded using the framework specific validation function, V, candidate generation is less constrained than classical query rewriting (shown in Figure 3.10). Instead of needing to return the same results as a base query, candidate queries return results that are a superset of the base query's results and the difference in these two result sets are considered invalid and discarded by V. This flexibility allows for the quick creation of candidates from a base query. Candidates can be generated by iterating through the power set of all relations present in the base query, while ensuring that each variable is still present in at least one relation. Figure 3.11 shows an example candidate search tree generated from rule t_1 from the running example. In this way, all possible candidates (that can be validated without examining the data in each relation)

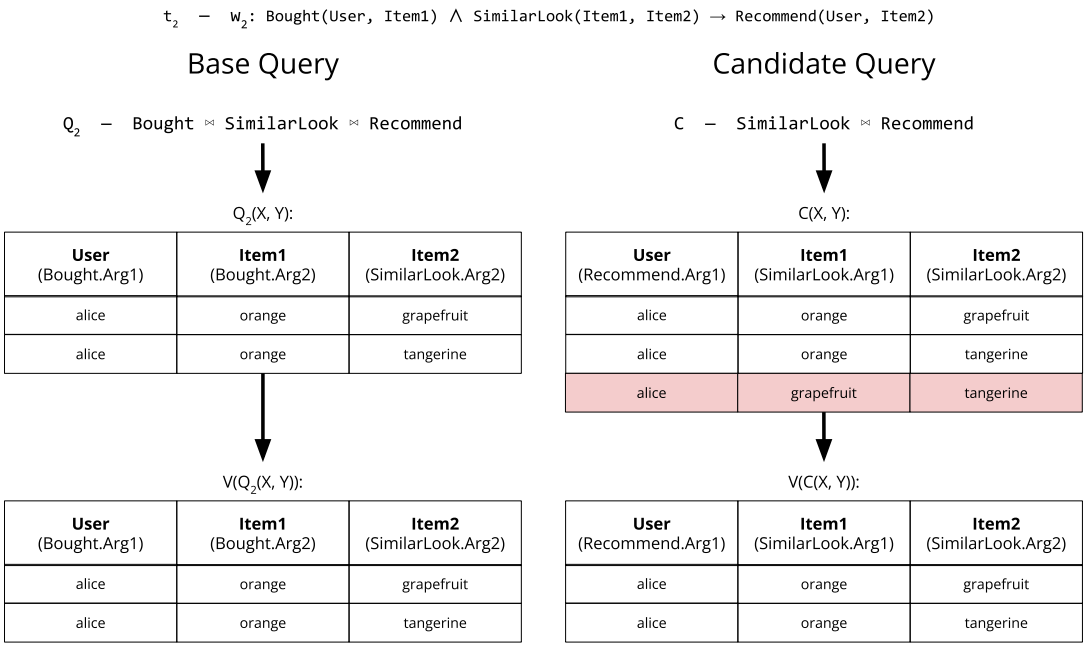


Figure 3.10: A base query, Q_2, and candidate query, C, both being used to ground the same rule template, t_2. The candidate query is simpler, but produces more results. The extraneous results (in red) from the candidate query can be filtered out by invoking the SRL framework's validation function, V.

are generated using just the base query. Candidates with fewer relations may produce more query results, but will be simpler to execute and more general, and therefore more likely to be shareable with other rules. This procedure can be efficiently done using a bit set as long as the number of relations fits into an unsigned integer type, as seen in Algorithm 2. Note that this algorithm ensures that the base query (a bit set of all 1s) is always added as a candidate, thereby guaranteeing that at least one valid candidate is always generated.

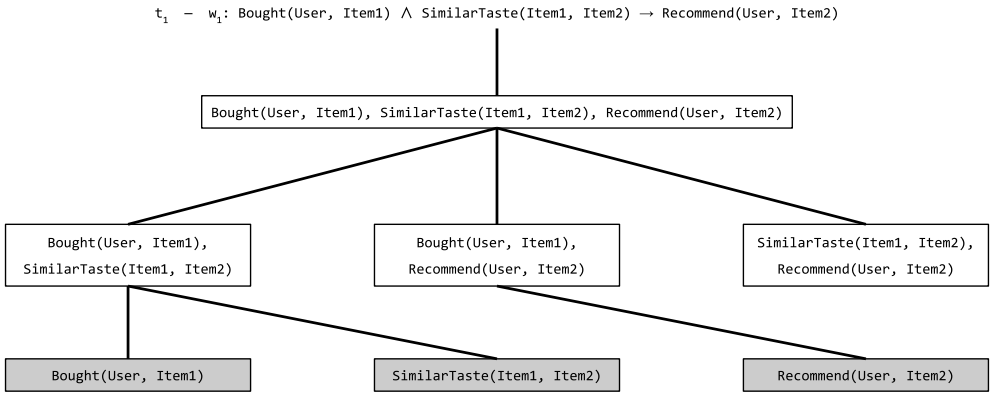


Figure 3.11: A fully materialized candidate search tree for rule t_1 from Figure 2.6. Duplicate nodes are excluded and nodes that do not contain the required variables (and are therefore invalid) are shaded.

However, in a rule with k atoms, there are $2^k - 1$ possible candidates. The number of candidates can quickly become intractable, so it is necessary to limit the number of overall candidates. Obtaining a query estimate for each candidate would allow for a dependable ranking of candidate performance where the top few may be selected. However, the cost of obtaining a query plan is non-trivial. Therefore, I create an approximate ranking of candidates while balancing the accuracy of the rankings with the cost of computing query estimates.

Algorithm 2: Generation of valid candidates from a rule. Note that the result set will always contain at least 1 item, the base query.

1 **Function** $GenerateCandidates$**:**

Input: $Rule$

Result: $Candidates = \{\}$

2 $BaseAtoms \leftarrow \{atom \mid atom \in Rule\}$

3 $AtomMasks \leftarrow (fn(atom) \rightarrow bitmask)$

4 $BaseVariables \leftarrow \{\}$

5 **foreach** $atom$ *in* $BaseAtoms$ **do**

6 $BaseVariables \leftarrow BaseVariables \cup \{variable \mid variable \in atom\}$

7 **for** $i \leftarrow 1$ **to** $2^{|BaseAtoms|}$ **do**

8 $Candidate \leftarrow \{atom \mid atom \in BaseAtoms \wedge AtomMasks(atom) | i \neq 0\}$

$Variables \leftarrow \{\}$

9 **foreach** $atom$ *in* $Candidate$ **do**

10 $Variables \leftarrow Variables \cup \{variable \mid variable \in atom\}$

11 **if** $Variables \equiv BaseVariables$ **then**

$Candidates = Candidates \cup \{Candidate\}$

Candidate Cost Estimation

A straightforward method of ranking each candidate is to assign them each a score based on the query execution time and number of query results the candidate will produce. A natural method of scoring each candidate is to obtain a query plan from the RDBMS, which may include expected runtime and expected number of results. In addition to runtime and number of results, the time to ground a rule also depends on the complexity of the rule itself. To incorporate all these factors, I can assign a single score for each candidate according to the following scoring function:

$$\begin{aligned} CandidateScore(C_1) = PredicateInstanceCount(C_1) * \\ (\alpha_{\{o,p\}} * EstimatedCost(C_1) + \beta_{\{o,p\}} * EstimatedCount(C_1)) \end{aligned} \tag{3.9}$$

where $PredicateInstanceCount(C_1)$ is the number of predicate instances present in candidate c_1 and acts as a proxy for the complexity of a candidate, $EstimatedCost(C_1)$ is the estimated cost by the RDBMS to execute the candidate C_1, and $EstimatedCount(C_1))$ is the RDBMS' estimated number of records returned by the execution of the candidate C_1. $\alpha_{\{o,p\}}$ and $\beta_{\{o,p\}}$ represent constants that relate the overhead of instantiating ground rules overall ($\alpha_{\{o,p\}}$) and per-instance ($\beta_{\{o,p\}}$). Each constant has two variants: optimistic (o) and pessimistic (p). These constants were computed using the results of running independent grounding on the validation splits of the datasets discussed in Section 3.2.2. The optimistic variants are one standard deviation less than the mean, while the pessimistic variants are one standard deviation greater than the mean. Section 11 will discuss how these constants are used to bound the search space for candidates.

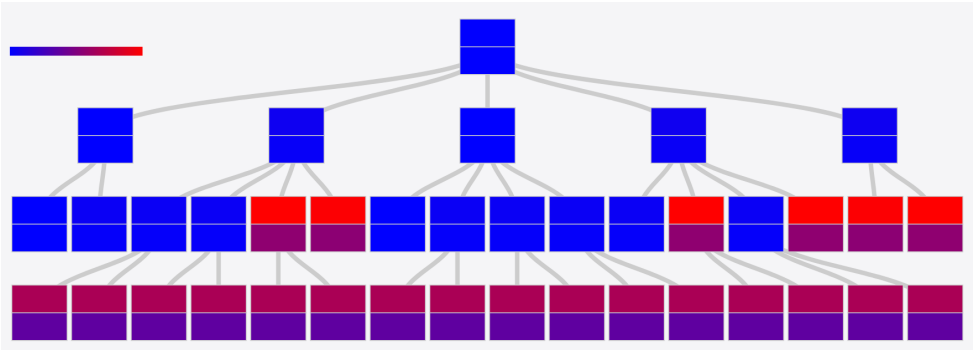


Figure 3.12: A fully expanded candidate search tree created from a single base query (the root). Each child represents a candidate that can be formed by dropping one relation from each parent. Each candidate has two colors: the top color represents the pessimistic cost, while the bottom color represents the optimistic cost. Colors closer to blue represent a lower cost while colors closer to red represent a higher cost. Duplicate nodes are omitted.

Candidate Search

To score each candidate while filtering out candidates that are likely to have a poor cost, I implement an approximate search. Because candidates can be generated starting from the base query and removing one relation at a time, the search space can be viewed as a tree rooted at the base query. Each level corresponds to candidates with the same number of relations. The maximum size of the search space is $2^n - 1$, where n is the number of relations in the base query. Figure 3.12 shows an example of a fully expanded search tree with optimistic and pessimistic costs represented using a heatmap. The base rule is the root and each child represents a candidate that can be formed by dropping one relation from each parent (duplicates nodes are omitted). Each node is colored with its pessimistic cost on top and optimistic cost on the bottom.

Because the search space grows exponentially with the number of relations in the rule, models with long rules may find exact search methods to be prohibitively costly. To counteract this,

I implemented budgeted versions of several classical search algorithms: breadth-first search (BFS), depth-first search (DFS), and uniform cost search (UCS). Additionally, I implemented approximate bounded versions of these searches using the optimistic and pessimistic costs to prune the search space. When these bounded methods encounter a node with an optimistic cost greater than its parent's pessimistic cost, the node is pruned. Using these search methods, a list of candidates can generated and scored by approximate runtime. Section 3.2.2 discusses experimental results exploring the effectiveness of each search method.

Candidate-Template Mapping Construction

Given the list of scored candidate queries from the candidate generation step, a mapping of candidates to rules each candidate can ground must be generated. For a candidate query to be used to ground a rule, the candidate's query results must be a superset of the results produced by the rule's base query. Checking if one query's results contains another query's results is the *query containment* problem Chandra and Merlin [1977], a well-studied problem in the database community. Formally, a query Q contains another query Q' if for any database D, $Q(D) \subseteq Q(D')$.

For conjunctive queries under set semantics, query containment is NP-complete Chandra and Merlin [1977]. However, the query containment problem can be even harder under bag semantics Khamis et al. [2021], and even undecidable with inequalities Jayram et al. [2006a]. There are also attributes of conjunctive queries that can make containment computation easier Kolaitis and Vardi [1998]. For example, query containment can be computed in linear time if each relation is guaranteed to appear no more than twice in a query Saraiya [1991]. I assume that all rules can

be represented with general conjunctive queries under set semantics with no additional restrictions. This requirement is straightforward to meet for many existing SRL frameworks. PSL ensures that all templates can be represented with conjunctive queries, and MLNs produce disjunctive queries that can be easily converted into conjunctive queries Augustine and Getoor [2018]. Therefore, I assume that query containment is NP-complete, even though specific SRL frameworks may be able to reduce the complexity.

For each candidate selected in the candidate generation process, query containment must be computed for each rule. However, as with candidate generation, the structure inherent in SRL models can be utilized to simplify the problem. Recall that candidates are generated in a structured manner by removing one relation at a time starting from the base query. Because of this, once it is verified that a candidate contains a template, it is known that all descendants of that candidate in the query generation tree also contain the template.

The general process is to keep the candidates in the tree form that I used when searching them, skipping over any candidates that were not chosen by the search. Each rule will produce one tree. I perform a breadth-first traversal of the candidate tree and check each candidate for containment against each rule that the candidates were not derived from. If a containment is found, then all descendants of that node are also marked for containment and they are not checked for that rule. The process continues until all candidates have been checked for containment of each rule.

Grounding Plan Creation

Given a scored list of candidates and a mapping of candidates to the rules each candidate can ground, the next task is to choose the set of candidates for a grounding plan that minimizes the total runtime of the entire grounding workload. Similar to multi-query optimization (MQO) in database literature Sellis [1988], the goal when choosing a grounding plan is to minimize a total workload over several queries with the knowledge that these queries may share common components or sub-workloads. However, unlike classic MQO, having multiple candidate queries for each rule allows for more flexibility when selecting a set of workloads to run. The problem of selecting candidates for the query plan is similar to the multiple-choice knapsack problem Kellerer et al. [2004], where the objective is to minimize the runtime of the collection of candidate queries while ensuring that each rule is represented. However, the differences with the classic multiple-choice knapsack problem are that each item for the knapsack (candidate query) can have multiple class labels (rules that the candidate satisfies), and a class label (rule) is allowed to be represented multiple times (as the rule will only be instantiated once). The classic multiple-choice knapsack problem is NP-hard problem, but the above two differences simplify this variant.

More formally, the goal is to select a subset of candidates, $\mathbf{C}^*$, from all candidates generated from candidate generation $\mathbf{C}$ according to the following minimization:

$$\begin{aligned} \mathbf{C}^* = \underset{C' \in \text{PowerSet}(\mathbf{C})}{\arg\min} & \sum_{c \in C'} CandidateScore(c) \\ \text{s.t.} & \sum_{c \in C'} S(c,t) \geq 1, \text{for } t \in T \end{aligned} \tag{3.10}$$

where $S(C,t)$ is an indicator function that outputs 1 if candidate C can ground for rule t, and zero

otherwise.

Since an exact solution is costly to compute, I use a greedy approach to find an approximate solution. I start by creating a sorted list of candidates for each template, including only candidates that can ground for the template and sorting ascending by $CandidateScore$ so that more favorable scores appear first. The first candidate in each of these lists represents the absolute best candidate that each rule can use individually. At this point, using the first candidate from each list can be no worse than independent grounding (assuming the query estimations are accurate). Now, each candidate receives a new score that is its original $CandidateScore$ minus the sum of $CandidateScore$ for the best candidate associated with each remaining rule this candidate can satisfy. Intuitively, this rewards each candidate for each rule it satisfies using the best (lowest) score for each rule. After all candidates are re-scored, the best (lowest scoring) candidate is selected to be in the final grounding plan, and the rules that the candidate can ground for are marked as complete and no longer considered in future score computations. This process repeats until all template are marked as complete.

3.2.2 Empirical Evaluation

To evaluate the impact of collective grounding, I performed an experimental evaluation of independent and collective grounding over a series of ten diverse problems/datasets[4]. As the state-of-the-art, PSL's implementation of independent grounding is used for all experiments Augustine and Getoor [2018], Augustine et al. [2019], Srinivasan et al. [2020a], and to produce the most

[4]Full code for replication of experiments is available at: `https://github.com/eriq-augustine/collective-grounding-experiments/tree/vldb23`

informative comparison, my implementation of collective grounding was built using the same PSL infrastructure.

Datasets

All datasets and models are from previously published papers and available at `https://github.com/linqs/psl-examples`. Each dataset includes between five and ten splits of the data. No changes were made to the rules or data for these experiments. The predictive tasks covered include collective classification (CC), link prediction (LP), and recommendation (REC). The details of each dataset are summarized in Table 3.2. Additionally, the "Minimum Queries" column shows the minimum possible queries that the model can be grounded with (determined by manual inspection). The gap between a dataset's number of template rules and their number of minimum queries represents possible common workloads in the rules that collective grounding may be able to exploit.

Hyperparameter Selection

The hyperparameters for collective grounding all pertain to the candidate generation step and include the maximum number of candidates chosen per rule, the maximum number of nodes explored during the candidate search (i.e., the search's budget), and the type of search used. To choose the best hyperparameters, the last (by lexicographical order) split for each dataset is held out for a hyperparameter search. Each configuration of hyperparameters is run in the same fashion as the rest of the experiments described in Section 3.2.2. Two sets of hyperparameters are

Dataset	Template Rules	Ground Rules	Task	Minimum Queries	Source
CITESEER	10	36K	CC	3	Bach et al. [2017]
CORA	10	41K	CC	3	Bach et al. [2017]
DDI	9	1M	LP	3	Sridhar et al. [2016]
EPINIONS	21	14K	LP	2	Huang et al. [2013]
ER	9	3M	CC	7	Bhattacharya and Getoor [2007]
IMDB-ER	6	129M	LP	4	Berry et al. [2018]
JESTER	8	1M	REC	2	Bach et al. [2017]
LASTFM	21	1.4M	REC	3	Kouki et al. [2015]
STANCE	11	2K	CC	3	Sridhar et al. [2015]
YELP	21	500K	REC	3	Kouki et al. [2015]

Table 3.2: Details of datasets and models used for evaluation. The generic task performed on each dataset include collective classification (CC), link prediction (LP), and recommendation (REC). "Minimum Queries" shows the minimum possible queries that the model can be grounded with (determined by manual inspection).

Dataset	Candidate Count	Search Budget	Search Type
CITESEER	10	5	UCS
CORA	5	3	DFS
DDI	5	3	BoundedDFS
EPINIONS	10	5	BoundedUCS
ER	3	3	DFS
IMDB-ER	3	5	BoundedUCS
JESTER	5	5	UCS
LASTFM	3	10	BFS
STANCE	5	5	BoundedDFS
YELP	10	3	BoundedDFS
OVERALL	10	10	BFS

Table 3.3: The best performing hyperparameters on the validation split for each dataset. The row labeled OVERALL contains the hyperparameters that on average performed the best over all datasets.

saved to be used in the rest of the experiments: the best set of hyperparameters for each dataset, referred to as the PER-DATASET hyperparameters, and the best set of hyperparameters averaged over all datasets, referred to as the OVERALL hyperparameters. The PER-DATASET hyperparameters represent situations where the user has enough data and time to tune collective grounding, whereas the OVERALL hyperparameters represent a set of hyperparameters that should generally perform well over a wide range of tasks and datasets.

Experiment Procedure

In all experiments, both independent and collective grounding are run ten times for each split not held out for hyperparameter search in each dataset. For each dataset, collective grounding was run using both the PER-DATASET and OVERALL sets of hyperparameters. Between each run system and disk caches were cleared and the database daemon was restarted. All experiments were performed on the same machine using 128GB of RAM, 20 threads clocked at 3.1 GHz, Ubuntu 20.10, and PostgreSQL 12.7.

Overall Results

Table 3.4 provides a detailed look of the runtime (time to run grounding), standard deviation, and statistically significantly best methods over all ten datasets. Significance is determined using a Student's T-test with a $p = 0.01$. The runtime provided is the time taken for the entire grounding process starting from templates and data and ending with the full set of ground rules. Figure 3.13 provides a graphical interpretation of the results. On all datasets collective grounding outperforms independent grounding.

Even in datasets where the rules do not exhibit structures that can be exploited, collective grounding still outperformed independent grounding by providing simpler queries (see Section 3.2.2). On datasets where there is potential to share workloads between the rules, collective grounding reduces the runtime by as much as two thirds. On the smallest dataset, STANCE, I can see that the overhead of collective grounding does not slow down the overall grounding process.

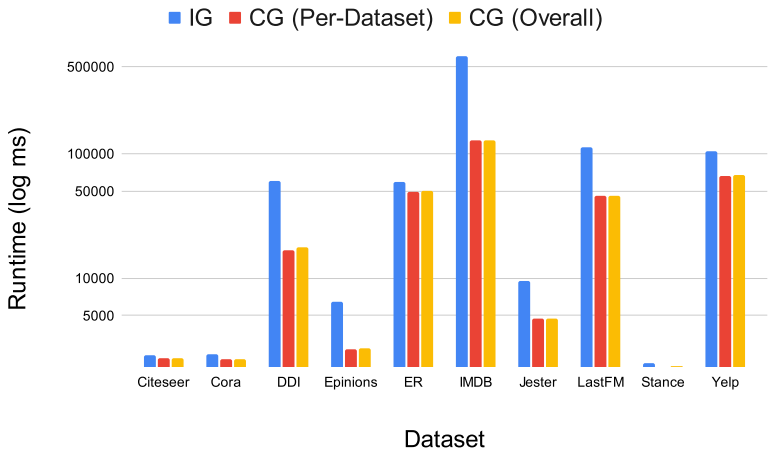
Dataset	IG	CG (PER-DATASET)	CG (OVERALL)
CITESEER	2387 ± 73	**2283 ± 40**	**2251 ± 103**
CORA	2434 ± 101	**2246 ± 66**	**2229 ± 43**
DDI	59852 ± 1584	**16764 ± 450**	17645 ± 712
EPINIONS	6418 ± 138	**2693 ± 78**	**2747 ± 62**
ER	59738 ± 286	**49585 ± 557**	**49916 ± 233**
IMDB-ER	606848 ± 15271	**129319 ± 1132**	**130907 ± 1376**
JESTER	9469 ± 196	**4722 ± 82**	**4733 ± 81**
LASTFM	112661 ± 2028	**45421 ± 481**	**45419 ± 391**
STANCE	2062 ± 65	**1933 ± 33**	**1957 ± 47**
YELP	104885 ± 609	**66478 ± 571**	**66980 ± 524**

Table 3.4: The runtime (in milliseconds) and standard deviation for independent grounding (IG) and collective grounding (CG) using both the PER-DATASET and OVERALL hyperparameters averaged over 10 splits. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold.

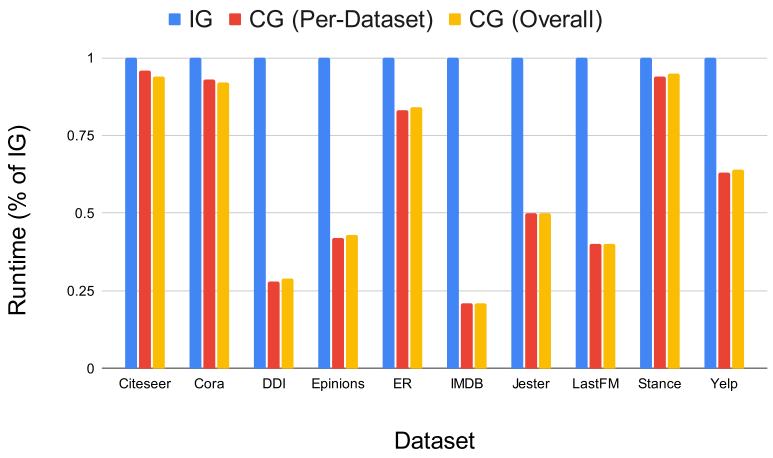
With one exception, using the general hyperparameters for collective grounding achieved the same result as using the hyperparameters tuned for each dataset. Overall the general hyperparameters I provide are sufficient for many different types and sizes of models, but tuning on specific datasets may provide a small advantage.

Candidate Search

Next I take a closer look at how the different parameters for the search phase in candidate generation effects the overall performance of collective grounding. Table 3.3 shows the highest



(a) Absolute Runtime (in log milliseconds).



(b) Runtime Relative to IG.

Figure 3.13: Runtime results for independent grounding (IG) and collective grounding (CG) using both the PER-DATASET and OVERALL hyperparameters. All results are aggregated over ten iterations, run on the same machine, and had database and disk caches cleared between runs.

performing hyperparameters for the validation split of each dataset. No single set of search parameters dominated.

A possible explanation for these results is that models with longer rules (more relations), such as ER, LASTFM, and YELP, tend to favor DFS search variants where candidates with fewer joins can be prioritized in the search, whereas models with shorter rules (fewer relations) tend to favor BFS and UCS search variants where simpler candidates are easier to access in the search space. This also explains why the best overall search setting is a combination of both extremes: BFS with a large search budget.

To better understand the search budget's effect on the overall results, I ran a set of experiments using a minimum budget of 1, the largest budget used in my hyperparameter search (the same budget as the OVERALL hyperparameters, 10), and an infinite budget. All other hyperparameters were set to the OVERALL values. I recorded the time it took to perform the candidate search and the overall runtime. The results are shown is Table 3.5. For most datasets, increasing the search budget from the OVERALL hyperparameter value made little difference. In most cases, a larger budget resulted in a slightly lower runtime. However, there are cases (e.g., ER and EPINIONS) where an unlimited budget caused CG to spend so much time searching for candidates that the overall runtime suffered.

Query Count Reduction

As shown by Table 3.6, collective grounding is able to reduce the number of queries performed in all datasets. Furthermore, Figure 3.13 shows that in all cases, collective grounding is

Dataset	Search Budget	Search Time	Runtime
CITESEER	1	0	2387 ± 73
	10	20 ± 1	2251 ± 103
	∞	19 ± 1	2229 ± 38
CORA	1	0	2434 ± 101
	10	20 ± 1	2229 ± 43
	∞	20 ± 1	2221 ± 30
DDI	1	0	59852 ± 1584
	10	63 ± 2	17645 ± 712
	∞	105 ± 3	17134 ± 264
EPINIONS	1	0	6418 ± 138
	10	127 ± 5	2747 ± 62
	∞	519 ± 5	3886 ± 133
ER	1	0	59738 ± 286
	10	225 ± 6	49916 ± 233
	∞	3294 ± 34	52616 ± 295
IMDB-ER	1	0	606848 ± 15271
	10	78 ± 5	130907 ± 1376
	∞	158 ± 7	130515 ± 1267
JESTER	1	0	9469 ± 196
	10	24 ± 2	4733 ± 81
	∞	22 ± 2	4733 ± 63
LASTFM	1	0	112661 ± 2028
	10	152 ± 11	45419 ± 391
	∞	359 ± 12	45074 ± 493
STANCE	1	0	2062 ± 65
	10	74 ± 4	1957 ± 47
	∞	128 ± 9	2044 ± 34
YELP	1	0	104885 ± 609
	10	154 ± 14	66980 ± 524
	∞	362 ± 15	66838 ± 431

Table 3.5: The effect of search budget on the search time and overall runtime. All times are shown in milliseconds, aggregated over ten runs, and include their standard deviation.

Dataset	# rules	# CG Queries
CITESEER	10	3
CORA	10	3
DDI	9	8
EPINIONS	21	2
ER	9	9
IMDB-ER	6	4
JESTER	8	2
LASTFM	21	3
STANCE	11	3
YELP	21	3

Table 3.6: The number of queries run in collective grounding (using the OVERALL hyperparameters) compared to the number of rules in each dataset.

able to overcome the additional overhead of computing more effective grounding plans.

The LASTFM dataset provides a clear illustration of how collective grounding can produce new query workloads that not only satisfy the given model, but also make sense when looked at in the context of the problem domain. The LASTFM dataset is a recommender system dataset where songs are recommended to users. The full LASTFM model contains 21 rules and can be seen in Appendix A.1.2. Kouki et al. (2015) divides these rules into eight different categories based on their use and data source [5]. Collective grounding is able to identify and build queries for three key

[5]The author's categorization of the rules does not affect the performance of the model, and only stands as a juxtaposition to collective grounding's grouping of the rules.

workloads in this model/dataset: two users and an item, two items and a user, and a user-item pair.

$$Q_1 = \text{RATING}(\texttt{U1}, \texttt{I}), \text{RATED}(\texttt{U2}, \texttt{I}), \text{RATED}(\texttt{U1}, \texttt{I})$$

$$Q_2 = \text{RATED}(\texttt{U}, \texttt{I2}), \text{RATING}(\texttt{U}, \texttt{I1}), \text{RATED}(\texttt{U}, \texttt{I1})$$

$$Q_3 = \text{RATING}(\texttt{U}, \texttt{I})$$

From a recommender systems perspective these three patterns are instantly recognizable and seen in many different approaches, with the first two patterns being the foundations of user and item-based collaborative filtering Ricci et al. [2011].

Increased Query Efficiency

As shown in Section 3.2.2, collective grounding can substantially speedup the grounding process by sharing queries between multiple rules. However, some datasets, such as DDI, that show a considerable reduction in runtime while still using almost the same number of queries as independent grounding. This raises two questions: 1) How does collective grounding manage to reduce DDI's runtime while using almost the same number of queries? 2) Why does collective grounding refuse the opportunity to use a single query for 7 rules?

To answer the first question, I start by looking at the full DDI model listed in Appendix A.1.3. The key modeling pattern of this model is the collective integration of several different

similarity measures. Collective grounding chose the following queries:

$$
\begin{aligned}
Q_1 =& \text{ATCSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_2 =& \text{SIDEEFFECTSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_3 =& \text{GOSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_4 =& \text{LIGANDSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_5 =& \text{CHEMICALSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_6 =& \text{SEQSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_7 =& \text{DISTSIMILARITY}(\texttt{D1}, \texttt{D2}), \text{INTERACTS}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}) \\
Q_8 =& \text{INTERACTS}(\texttt{D1}, \texttt{D2})
\end{aligned}
\tag{3.11}
$$

The final two rules share the same query, but more importantly the number of joins for each of the similarity-based rules has reduced. Table 3.7 shows the runtime and number of results for the base query and query chosen by collective grounding for each similarity measure. Even though the collective grounding queries typically return more results, the queries can be executed much more quickly. Here collective grounding finding success not though reducing the number of queries, but by using more efficient queries discovered during candidate generation.

Given the results observed in Table 3.7, where a simplification of the query (reduction in the number of joins) results in a faster query, collective grounding's refusal to ground all of DDI's similarity rules with the same general query seems even more surprising. For example, the below queries can be used to ground all of DDI:

$$
\begin{aligned}
Q_1 =& \text{VALIDINTERACTION}(\texttt{D1}, \texttt{D3}), \text{VALIDINTERACTION}(\texttt{D2}, \texttt{D3}) \\
Q_2 =& \text{INTERACTS}(\texttt{D1}, \texttt{D2})
\end{aligned}
\tag{3.12}
$$

Similarity Measure	Base Query Runtime	Base Query # Results	CG Query Runtime	CG Query # Results
ATC	7295 ± 152	1257947	1647 ± 57	1360876
Chemical	8105 ± 220	1380330	1933 ± 40	1483650
Dist	7786 ± 266	1358420	1732 ± 31	1461670
GO	7987 ± 232	1358420	1776 ± 47	1461670
Ligand	7298 ± 232	1242297	1628 ± 47	1345176
Seq	7991 ± 239	1358420	1797 ± 53	1461670
Side Effect	7864 ± 245	1380330	1812 ± 90	1483650

Table 3.7: The performance of base queries compared against simpler queries chosen by collective grounding for the DDI dataset. Note that the simpler queries used by CG results in significantly faster runtimes while only returning a few additional query results (which are subsequently filtered out by the SRL framework). The OVERALL hyperparameters are used for CG.

However as shown in Table 3.8, the grounding plan selected by collective grounding containing 8 queries easily outperforms the "Forced Sharing" grounding plan from Equation 3.12. Part of the reduced performance of the forced sharing grounding plan may come from the many extraneous results returned by the very general query. It seems that the tables containing the similarity measures are selective enough to warrant seven different queries instead of one more general query.

Method	# Queries	Runtime	Query Results
IG	9	59852	9435074
CG	8	17645	9583439
Forced Sharing	2	44031	31156650

Table 3.8: The performance of independent grounding and collective grounding against forcing a grounding plan that uses only two queries (see Equation 3.12). CG runs used OVERALL hyperparameters.

3.3 Conclusions and Future Work

In this chapter, I have shown my methods of addressing the two primary computational bottlenecks in SRL: inference and grounding. To improve the scalability of SRL inference, I introduce a new out-of-core inference method that allows SRL systems to (for the first time) scale up to ground network with more than 1 billion ground rules. To improve the scalability of grounding, I introduce a new grounding paradigm for SRL that uses techniques from database optimization to

collectively ground rules in a manner that finds more efficient and reusable query workloads. The result of my efforts in the context of the larger SRL community can be seen in Table 3.9. The last four milestones in this table are all direct results of my research.

Year	System	Size (Millions)	Runtime (Minutes)	Ground Rules Per Minute	% Time Grounding
2007 Kok et al. [2009]	Alchemy (MLN)	1	398	2.5K	96%
2011 Niu et al. [2011]	Tuffy (MLN)	2.5	120	20K	50%
2015 Magliacane et al. [2015]	FoxPSL	14	30	0.5M	33%
2018 Augustine and Getoor [2018]	PSL 2.1	129	25	5M	99%
2020 Srinivasan et al. [2020a]	TI	1000	500	2M	N/A
2021 Berry et al. [2018]	PSL 2.2	129	10	13M	40%
Collective Grounding	PSL 2.2 + CG	129	2	65M	8%

Table 3.9: A timeline of the grounding milestones in the SRL community. The size column refers to the number of ground rules in the model. All systems except FoxPSL were run on the same machine. All PSL rows are run on the IMDB-ER dataset (discussed in Section 3.2.2). Time spent grounding does not apply directly to TI, since grounding and inference occur at the same time.

While Section 3.1 introduces the fundamentals of TI, there remain several open areas for research. Incorporating lifted inference is a promising extension to TI. Because TI is orthogonal to lifting, these two can be combined to speed up inference further. Next, despite impressive performance on large datasets, the overall process of TI is largely sequential; parallelizing TI can be another way to speeding up inference further. Another interesting avenue for research is to create a hybrid IE using ADMM and SGD. SGD often minimizes quickly during the first few iterations, however may take many more iterations to fully converge (especially if the learning rate is poorly

selected). Conversely, ADMM converges more slowly than SGD, but more steadily. A hybrid IE could start with SGD and then switch to ADMM after the first few iterations. Finally, TI can be extended to any other SRL framework that can support streaming grounding and streaming inference.

A future avenue of research for collective grounding is to integrate it with orthogonal methods for improving the efficiency of grounding in templated graphical models. Specifically, TI (as discussed in Section 3.1 and lifted inference Srinivasan et al. [2019a] both aid in grounding extremely large models (with the billions of ground rules), and both work orthogonally to collective grounding, i.e., they both work with ground rules after grounding and do not affect the grounding plan. Integrating these techniques would create a synergy where collective grounding can produce highly efficient grounding plans that tandem or lifted inference can efficiently store and infer over.

Chapter 4

Expressivity

In this chapter, I discuss my work in producing expressive SRL systems that can represent a wide range of models[1]. Using first-order logic, PSL can already represent a wide range of models. However, there are limitations to the models that PSL can support. First, PSL lacks the ability to easily incorporate models that require high-dimensional feature spaces. Additionally, PSL restricts rule weights to be non-negative. In this chapter, I will address these two limitations.

4.1 NeuPSL

The integration of deep learning and symbolic reasoning is one of the earliest, yet open, challenges in the machine learning community [d'Avila Garcez et al., 2002]. A promising area of research that incorporates neural perception into logic-based reasoning is neuro-symbolic computing

[1]The work presented in this chapter is currently in submission Pryor et al. [2022] and was presented at the 2021 Workshop on Tractable Probabilistic Modeling (TPM) Dickens* et al. [2021].

(NeSy) [Besold et al., 2017, d'Avila Garcez et al., 2019, Raedt et al., 2020]. This integration has the benefit of allowing for expressive models that are able to perform joint inference (structured prediction), i.e., jointly predicting multiple labels or tasks, and joint learning, i.e., parameter learning over a joint loss function. Joint inference and learning have been used in strictly deep neural architectures to great success; graph neural networks [Wu et al., 2021], conditional random fields [Zheng et al., 2015], hidden Markov models [Li et al., 2013], etc. Furthermore, models designed for tasks such as translation [Bahdanau et al., 2015] or image segmentation [Nowozin and Lampert, 2011] without joint inference are unable to accurately distinguish predictions. For example, in translation, understanding the difference between homonyms such as *fair* (equitable vs. beautiful) or in image segmentation, making locally consistent decisions about the foreground and background.

In this section, I introduce a principled NeSy method that integrates deep learning with hard and soft logical constraints prioritizing joint learning and inference. My approach, *Neural Probabilistic Soft Logic* (NeuPSL), extends PSL. I choose PSL because it encodes an underlying graphical model that supports scalable and convex joint inference. Additionally, as mentioned earlier, PSL has been shown to support a wide variety of joint inference and learning tasks including natural language processing [Beltagy et al., 2014, Deng and Wiebe, 2015, Liu et al., 2016, Rospocher, 2018], data mining [Alshukaili et al., 2016, Kimmig et al., 2019], recommender systems [Kouki et al., 2015], knowledge graph discovery [Pujara et al., 2013], fairness modeling [Farnadi et al., 2019, Dickens et al., 2020], and causal reasoning [Sridhar et al., 2018].

My key contributions to this component of the expressivity pillar include: 1) I define a family of energy-based models called Neuro-Symbolic Energy-Based Models (NeSy-EBMs), 2) I

introduce NeuPSL and highlight how it fits into the NeSy ecosystem, paying particular attention to how it supports joint inference and learning, and 3) I perform an extensive evaluation, showing that NeuPSL consistently outperforms existing approaches on joint inference in low data settings while maintaining a significantly lower variance in most settings.

The work covered in this section is currently in submission Pryor et al. [2022].

4.1.1 Related Work

Neuro-symbolic computing (NeSy) is a very active area of research that aims to incorporate logic-based reasoning with neural computation d'Avila Garcez et al. [2002], Bader and Hitzler [2005], d'Avila Garcez et al. [2009], Serafini and d'Avila Garcez [2016], Besold et al. [2017], Donadello et al. [2017], Yang et al. [2017], Evans and Grefenstette [2018], Manhaeve et al. [2021a], d'Avila Garcez et al. [2019], Raedt et al. [2020], Lamb et al. [2020], Badreddine et al. [2022]. This integration allows for interpretability and reasoning through symbolic knowledge, while providing robust learning and efficient inference with neural networks. For an in-depth introduction, I refer the reader to these excellent surveys Besold et al. (2017) and Raedt et al. (2020). In this section, I identify key NeSy research categories and provide a brief description of each:

Differentiable frameworks of logical reasoning: Methods in this category use the universal function approximation properties of neural networks to emulate logical reasoning inside networks. Examples include: Rocktäschel and Riedel (2017), Bošnjak et al. (2017), Evans and Grefenstette (2018), and Cohen et al. (2020).

Constrained Output: These approaches enforce constraints or regularizations on the

output of neural networks. Examples include: Hu et al. (2016), Diligenti et al. (2017), Donadello et al. (2017), Mehta et al. (2018), Xu et al. (2018), and Nandwani et al. (2019).

Executable logic programs: These approaches use neural models to build executable logical programs. Examples include Liang et al. (2017) and Mao et al. (2019). I highlight Logic Tensor Networks (LTNs) [Badreddine et al., 2022], as I compare with them in the empirical evaluation. LTNs connect neural predictions into functions representing symbolic relations with real-valued or fuzzy logic semantics.

Neural networks as predicates: This line of work integrates neural networks and probabilistic reasoning by introducing neural networks as predicates in the logical formulae. This technique provides a very general and flexible framework for neuro-symbolic reasoning, and allows for the use of multiple networks as well as the full incorporation of constraints and relational information. Examples include DASL Sikka et al. [2020], NeurASP Yang et al. [2020], DeepProbLog (DPL) Manhaeve et al. [2021a], and my proposed method (Neural Probabilistic Soft Logic). DPL combines general-purpose neural networks with the probabilistic modeling of ProbLog Raedt et al. [2007] in a way that allows for learning and inference over complex tasks, such as program induction. I include DPL in the empirical evaluation.

4.1.2 Neuro-Symbolic Energy-Based Models

Energy-Based Models (EBMs) [LeCun et al., 2006] measure the compatibility of a collection of observed or input variables $\mathbf{x} \in \mathcal{X}$ and target or output variables $\mathbf{y} \in \mathcal{Y}$ with a scalar valued *energy function*: $E : \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}$. Low energy states of the variables represent high

compatibility. Prediction or *inference* in EBMs is performed by finding the lowest energy state of the variables $\mathbf{y}$ given $\mathbf{x}$. Energy functions are parameterized by variables $\mathbf{w} \in \mathcal{W}$, and *learning* is the task of finding a parameter setting that associates low energy to correct solutions.

Building on this well-known formulation, I introduce *Neuro-Symbolic Energy-Based Models* (NeSy-EBMs). NeSy-EBMs are a family of EBMs that integrate neural architectures with explicit encodings of symbolic relations. The input variables are organized into neural, $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$, and symbolic, $\mathbf{x}_{sy} \in \mathcal{X}_{sy}$, vectors. Furthermore, the parameters of the energy function, $\mathbf{w}$, are partitioned into neural parameters, $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$, and symbolic parameters, $\mathbf{w}_{sy} \in \mathcal{W}_{sy}$. NeSy energy functions are written as $E : \mathcal{Y} \times \mathcal{X} \times \mathcal{X}_{nn} \times \mathcal{W}_{nn} \times \mathcal{W}_{sy} \to \mathbb{R}$. Formally,

Definition 3 (Neuro-Symbolic Energy-Based Models)**.** *Let* $\mathbf{y} = [y_i]_{i=1}^{n_y}$ *and* $\mathbf{x} = [x_i]_{i=1}^{n_x}$ *be vectors of real valued variables with symbolic interpretations. Let* $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$ *be neural networks with corresponding parameters* $\mathbf{w}_{nn} = [\mathbf{w}_{nn,i}]_{i=1}^{n_g}$ *and, without loss of generality, inputs* $\mathbf{x}_{nn}$*. A* ***symbolic potential*** *is a function:* $\psi(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \in \mathbb{R}$*. A* ***NeSy-EBM energy function****, parameterized by* ***symbolic parameters*** $\mathbf{w}_{sy} = [w_{sy,i}]_{i=1}^{n_{sy}}$*, is a mapping of a vector of symbolic potential outputs,* $\Psi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = [\psi_i(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))]_{i=1}^{m}$*, to a real value:* $E(\Psi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{w}_{sy}) \in \mathbb{R}$*. For simplicity I also express a NeSy energy function as* $E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{sy}) \in \mathbb{R}$*.*

NeSy-EBM models are differentiated from one another by the instantiation process, the form of the symbolic potentials, and the method for combining the symbolic potentials to define the energy function. One example of a NeSy-EBM is DeepProbLog (DPL) [Manhaeve et al., 2018].

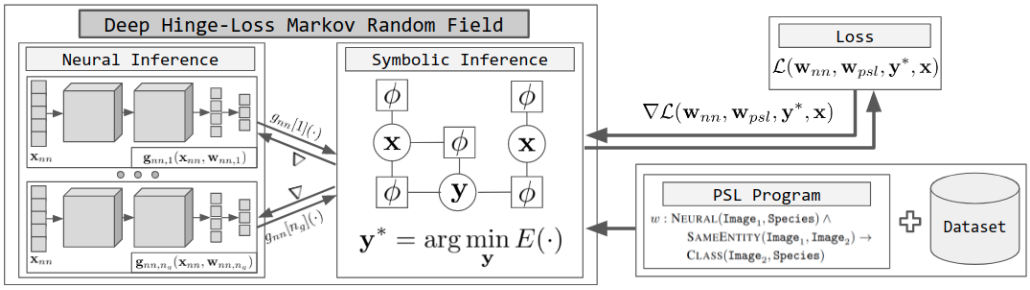


Figure 4.1: NeuPSL inference and learning pipeline.

DPL uses the output of a neural network to specify probabilities of events. The events are then used in logical formulae defining probabilistic dependencies and act as symbolic potentials. The DPL energy function is a joint distribution and inference is finding the highest-probability, i.e., lowest energy, state of the variables. Logic Tensor Networks (LTNs) [Badreddine et al., 2022] are another NeSy method that belongs to the NeSy-EBM family, and instantiates a model which forwards neural network predictions into functions representing symbolic relations with real-valued or fuzzy logic semantics. From the NeSy-EBM point of view, the fuzzy logic functions are the symbolic potentials and their composition defines the energy function.

4.1.3 Neural Probabilistic Soft Logic

Having laid the NeSy-EBM groundwork, I now introduce *Neural Probabilistic Soft Logic* (NeuPSL), a novel NeSy-EBM framework that extends the PSL framework. As illustrated in Figure 4.1, NeuPSL leverages the low-level perception of neural networks by integrating their outputs into a set of symbolic potentials created by a PSL program. The symbolic potentials and neural networks together define a *deep hinge-loss Markov random field* (*Deep-HL-MRF*), a tractable probabilistic

graphical model that supports scalable convex joint inference. This section provides a comprehensive description of how NeuPSL instantiates its symbolic potentials and how the symbolic potentials are combined to define an energy function, while the following section details NeuPSL's end-to-end neural-symbolic inference, learning, and joint reasoning processes.

NeuPSL instantiates the symbolic potentials of its energy function using the PSL language where dependencies between relations and attributes of entities in a domain, defined as *atoms*, are encoded with weighted first-order logical clauses and linear arithmetic inequalities referred to as *rules*. To illustrate, consider a setting in which a neural network is used to classify the species of an animal in an image. Further, suppose there exists external information suggesting when two images may contain the same entity. The information linking the images may come from a variety of sources, such as the images' caption or metadata indicating the images were captured by the same device within a short period of time. NeuPSL represents the neural network's animal classification of an image ($\texttt{Image}_1$) as a species ($\texttt{Species}$) with the atom $\textsc{Neural}(\texttt{Image}_1, \texttt{Species})$ and the probability that two images ($\texttt{Image}_1$ and $\texttt{Image}_2$) contain the same entity with the atom $\textsc{SameEntity}(\texttt{Image}_1, \texttt{Image}_2)$. Additionally, I represent NeuPSL's classification of $\texttt{Image}_2$ with $\textsc{Class}(\texttt{Image}_2, \texttt{Species})$. The following weighted logical rule in NeuPSL represents the notion that two images identified as the same entity may also be of the same species:

$$w : \textsc{Neural}(\texttt{Image}_1, \texttt{Species}) \wedge \textsc{SameEntity}(\texttt{Image}_1, \texttt{Image}_2) \rightarrow \textsc{Class}(\texttt{Image}_2, \texttt{Species}) \quad (4.1)$$

The parameter w is the weight of the rule and it quantifies its relative importance in the model. Note, these rules can either be hard or soft constraints. Atoms and weighted rules are templates for creating the symbolic potentials, or soft constraints. To create these symbolic potentials, atoms and rules are instantiated with observed data and neural predictions. Atoms instantiated with elements from the data are referred to as *ground atoms*. Then, valid combinations of ground atoms substituted in the rules create *ground rules*. To illustrate, suppose that there are two images $\{Id1, Id2\}$ and

three species classes $\{Cat, Dog, Frog\}$. Using the above data for cats would result in the following ground rules (analogous ground rules would be created for dogs and frogs):

$$w : \textsc{Neural}(Id1, Cat) \wedge \textsc{SameEntity}(Id1, Id2) \rightarrow \textsc{Class}(Id2, Cat)$$

$$w : \textsc{Neural}(Id2, Cat) \wedge \textsc{SameEntity}(Id2, Id1) \rightarrow \textsc{Class}(Id1, Cat)$$

Ground atoms are mapped to either an observed variable, x_i, target variable, y_i, or a neural function with inputs $\mathbf{x}_{nn}$ and parameters $\mathbf{w}_{nn,i}$: $g_{nn,i}(\mathbf{x}_{nn}, \mathbf{w}_{nn,i})$. Then, variables are aggregated into the vectors $\mathbf{x} = [x_i]_{i=1}^{n_x}$ and $\mathbf{y} = [y_i]_{i=1}^{n_y}$ and neural outputs are aggregated into the vector $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$. Each ground rule creates one or more potentials defined over $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{g}_{nn}$. Logical rules, such as the one given in example (4.1), are relaxed using Łukasiewicz continuous valued logical semantics Klir and Yuan [1995]. NeuPSL also supports arithmetic rules for defining penalty functions. Each potential ϕ_i is associated with a parameter $w_{psl,i}$ that is inherited from its instantiating rule. The potentials and weights from the instantiation process are used to define a member of a tractable class of graphical models, *deep hinge-loss Markov random fields* (Deep-HL-MRF):

Definition 4 (Deep Hinge-Loss Markov Random Field)**.** *Let* $\mathbf{y} = [y_i]_{i=1}^{n_y}$ *and* $\mathbf{x} = [x_i]_{i=1}^{n_x}$ *be vectors of real valued variables. Let* $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$ *be functions with corresponding parameters* $\mathbf{w}_{nn} = [\mathbf{w}_{nn,i}]_{i=1}^{n_g}$ *and inputs* $\mathbf{x}_{nn}$*. A* ***deep hinge-loss potential*** *is a function of the form*

$$\phi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = \max(l(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), 0)^{\alpha} \tag{4.2}$$

where $\alpha \in \{1, 2\}$*. Let* $\mathcal{T} = [t_i]_{i=1}^{r}$ *denote an ordered partition of a set of* m *deep hinge-loss potentials* $\{\phi_1, \cdots, \phi_m\}$*. Next, for each partition* t_i *define* $\Phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) := \sum_{j \in t_i} \phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$

and let $\mathbf{\Phi}(\mathbf{y}, \mathbf{x}, \mathbf{x_{nn}}, \mathbf{w_{nn}}) := [\Phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn})]_{i=1}^{r}$. *Further, let* $\mathbf{w}_{psl} = [w_i]_{i=1}^{r}$ *be a vector of non-negative weights corresponding to the partition* $\mathcal{T}$. *Then, a* ***deep hinge-loss energy function*** *is*

$$E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \sum_{i=1}^{R} \mathbf{w}_{psl}[i]\Phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = \mathbf{w}_{psl}^{T}\mathbf{\Phi}(\mathbf{y}, \mathbf{x}, \mathbf{x_{nn}}, \mathbf{w_{nn}}) \tag{4.3}$$

Further, let $\mathbf{c} = [c_i]_{i=1}^{q}$ *be a vector of* q *linear constraints in standard form, defining the feasible set* $\mathbf{\Omega} = \{\mathbf{y}, \mathbf{x} \,|\, c_i(\mathbf{y}, \mathbf{x}) \leq 0, \, \forall i\}$. *Then a* ***deep hinge-loss Markov random field****,* $\mathcal{P}$*, with random variables* $\mathbf{y}$ *conditioned on* $\mathbf{x}$ *and* $\mathbf{x}_{nn}$ *is a probability density of the form*

$$P(\mathbf{y}|\mathbf{x}, \mathbf{x}_{nn}) = \begin{cases} \frac{1}{Z(\mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})) & (\mathbf{y}, \mathbf{x}) \in \mathbf{\Omega} \\ 0 & o.w. \end{cases} \tag{4.4}$$

$$Z(\mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \int_{\mathbf{y}|\mathbf{y},\mathbf{x} \in \mathbf{\Omega}} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}))d\mathbf{y} \tag{4.5}$$

4.1.4 NeuPSL Inference and Learning

There is a clear connection between neural and symbolic inference in NeuPSL that allows any neural architecture to interact with symbolic reasoning in a simple and expressive manner. The NeuPSL neural-symbolic interface and inference pipeline is shown in Figure 4.1. *Neural inference* is computing the output of the neural networks given the input $\mathbf{x}_{nn}$, i.e., computing $g_{nn,i}(\mathbf{x}_{nn}, \mathbf{w}_{nn,i})$ for all i. NeuPSL *symbolic inference* minimizes the energy function over $\mathbf{y}$:

$$\mathbf{y}^* = \underset{\mathbf{y}|(\mathbf{y},\mathbf{x}) \in \mathbf{\Omega}}{\arg\min} \; E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) \tag{4.6}$$

Note that the energy function and constraints are convex in $\mathbf{y}$. Any scalable convex optimizer can be applied to solve (4.6). NeuPSL uses the alternating direction method of multipliers [Boyd et al.,

2010].

NeuPSL learning is the task of finding both neural parameters and symbolic parameters, i.e., rule weights, that assign low energy to correct values of the output variables, and higher energies to incorrect values. Learning objectives are functionals mapping an energy function and a set of training examples $\mathcal{S} = \{(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}) : i = 1, \cdots, P\}$ to a real-valued loss. As the energy function for NeuPSL is parameterized by the neural parameters $\mathbf{w}_{nn}$ and symbolic parameters $\mathbf{w}_{psl}$, I express the learning objective as a function of $\mathbf{w}_{nn}$, $\mathbf{w}_{psl}$, and $\mathcal{S}$: $\mathcal{L}(\mathcal{S}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$. Learning objectives follow the standard empirical risk minimization framework and are therefore separable over the training examples in $\mathcal{S}$ as a sum of per-sample loss functions $L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$. Concisely, NeuPSL learning is the following minimization:

$$\underset{\mathbf{w}_{nn}, \mathbf{w}_{psl}}{\arg\min}\, \mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathcal{S}) = \underset{\mathbf{w}_{nn}, \mathbf{w}_{psl}}{\arg\min} \sum_{i=1}^{P} L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

In the learning setting, variables $\mathbf{y}_i$ from the training set S are partitioned into vectors $\mathbf{y}_{i,t}$ and $\mathbf{z}_i$. The variables $\mathbf{y}_{i,t}$ represent variables for which there is a corresponding truth value, while $\mathbf{z}_i$ represent latent variables. Without loss of generality, I write $\mathbf{y}_i = (\mathbf{y}_{i,t}, \mathbf{z}_i)$.

There are multiple losses that one could motivate for optimizing the parameters of an EBM. The most common losses, including the loss I present in this section, use the following terms:

$$\mathbf{z}_i^* = \underset{\mathbf{z} | ((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}) \in \mathbf{\Omega}}{\arg\min} E((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

and

$$\mathbf{y}_i^* = \underset{\mathbf{y} | (\mathbf{y}, \mathbf{x}_i) \in \mathbf{\Omega}}{\arg\min} E(\mathbf{y}, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

In words, $\mathbf{z}_i^*$ and $\mathbf{y}_i^*$ are the lowest energy states given $(\mathbf{y}_{i,t}, \mathbf{x}_i, \mathbf{x}_{i,nn})$ and $(\mathbf{x}, \mathbf{x}_{i,nn})$, respectively. A special case of learning is when the per-sample losses are not functions of $\mathbf{z}_i^*$ and $\mathbf{y}_i^*$, and more specifically, the losses do not require any subproblem optimization. I refer to this situation as *constraint learning*. Constraint learning reduces the time required per iteration at the cost of expressivity.

All interesting learning losses for NeuPSL are a composition of the energy function. Thus, a gradient-based learning algorithm will necessarily require the following partials derivatives: [2]

$$\frac{\partial E(\cdot)}{\partial \mathbf{w}_{psl}[i]} = \Phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$$

and

$$\frac{\partial E(\cdot)}{\partial \mathbf{w}_{nn}[i]} = \mathbf{w}_{psl}^T \nabla_{\mathbf{w}_{nn}[i]} \Phi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$$

Continuing with the derivative chain rule and noting the potential can be squared ($\alpha = 2$) or linear ($\alpha = 1$), the potential partial derivative with respect to $\mathbf{w}_{nn}[i]$ is the piece-wise defined function:[2]

$$\frac{\partial \phi(\cdot)}{\partial \mathbf{w}_{nn}[i]} = \begin{cases} \frac{\partial}{\partial \mathbf{g}_{nn}[i]} \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{w}_{nn}[i]} \mathbf{g}_{nn}[i](\mathbf{x}_{nn}, \mathbf{w}_{nn}) & \alpha = 1 \\ 2 \cdot \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{g}_{nn}[i]} \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{w}_{nn}[i]} \mathbf{g}_{nn}[i](\mathbf{x}_{nn}, \mathbf{w}_{nn}) & \alpha = 2 \end{cases}$$

$$\frac{\partial \phi(\cdot)}{\partial \mathbf{g}_{nn}[i]} = \begin{cases} 0 & \phi(\cdot) = 0 \\ \frac{\partial}{\partial g_{nn}[i]} l(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \phi(\cdot) > 0 \end{cases}$$

Since $l(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ is a linear function, the partial gradient with respect to $\mathbf{g}_{nn}[i]$ is trivial. With the partial derivatives presented here, standard backpropagation-based algorithms for

[2]Note arguments of the energy function and symbolic potentials are dropped for simplicity, i.e., $E(\cdot) = E(\mathbf{y}, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$, $\phi(\cdot) = \phi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$.

computing gradients can be applied for both neural and symbolic parameter learning.

Energy Loss: A variety of differentiable loss functions can be chosen for $\mathcal{L}$. For simplicity, I will discuss the *energy loss*. The energy loss parameter learning scheme directly minimizes the energy of the training samples, i.e., the per-sample losses are:

$$L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = E((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

Notice that inference over the latent variables is necessary for gradient and objective value computations. However, a complete prediction from NeuPSL, i.e., inference over all components of $\mathbf{y}$, is not necessary. Therefore the parameter learning problem is as follows:

$$\begin{aligned}
&\arg\min_{\mathbf{w}_{nn}, \mathbf{w}_{psl}} \mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathcal{S}) = \\
&\arg\min_{\mathbf{w}_{nn}, \mathbf{w}_{psl}} \sum_{i=1}^{P} \min_{\mathbf{z} | ((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}) \in \mathbf{\Omega}} \mathbf{w}_{psl}^T \Phi((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn})
\end{aligned}$$

With L2 regularization, the NeuPSL energy function is strongly convex in all components of $\mathbf{y}_i$. Thus, by Danskin (1966), the gradient of the energy loss, $L_i(\cdot)$, with respect to $\mathbf{w}_{psl}$ at $\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}\mathbf{w}_{nn}$ is:

$$\nabla_{\mathbf{w}_{psl}} L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \Phi((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn})$$

Then the per-sample energy loss partial derivative with respect to $\mathbf{w}_{nn}[j]$ at $\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{psl}$ is:

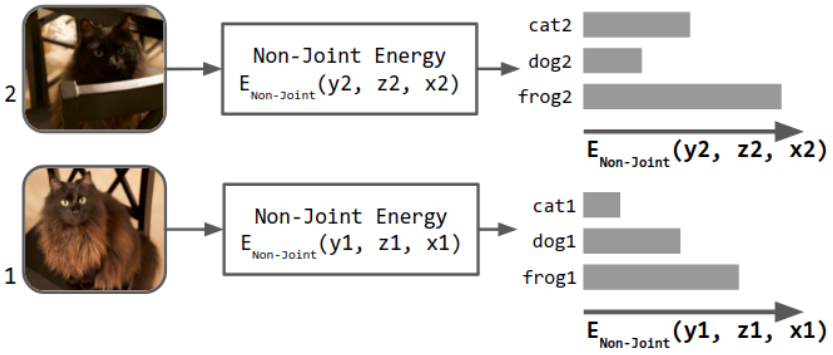
$$\begin{aligned}
&\frac{\partial L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})}{\partial \mathbf{w}_{nn}[j]} = \\
&\sum_{r=1}^{R} \mathbf{w}_{psl}[r] \sum_{q \in \tau_r} \frac{\partial \phi_q((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn})}{\partial \mathbf{w}_{nn}[j]}
\end{aligned}$$

Details on the learning algorithms and accounting for degenerate solutions of the energy loss are included in the appendix.

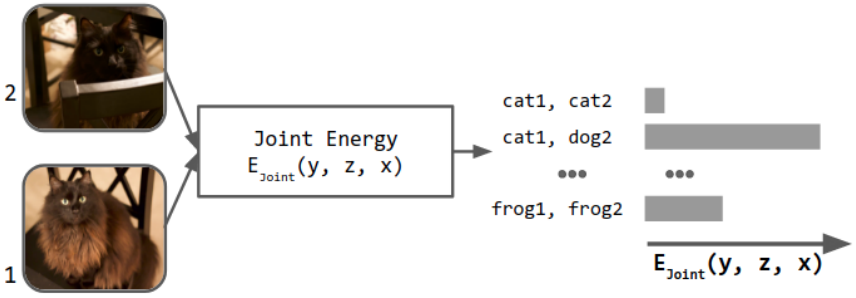
4.1.5 Joint Reasoning in NeSy-EBMs

The EBM energy function encodes dependencies between potentially high dimensional input variables, $\mathbf{x}$, and output variables, $\mathbf{y}$. I highlight two important categories of energy functions: *joint* and *independent* energy functions. Formally, I refer to an energy function that is additively separable over the output variables $\mathbf{y}$ as an *independent energy function*, i.e., there exists functions $E_1(\mathbf{y}[1], \mathbf{x}), \cdots, E_{n_y}(\mathbf{y}[n_y], \mathbf{x})$ such that $E(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n_y} E_i(\mathbf{y}[i], \mathbf{x})$. While a function that is not separable over output variables $\mathbf{y}$ is a *joint energy function*. This categorization allows for an important distinction during inference and learning. In independent inference, the predicted value for a variable $\mathbf{y}[i]$ has no influence over that of $\mathbf{y}[j]$ where $j \neq i$ and can therefore be predicted separately, i.e., independently. Independent energy functions speed up inference and learning, however, they cannot leverage joint information that may be used to improve predictions.

To illustrate, recall the example described in Section 4.1.3 where a neural network is used to classify the species of an animal in an image with external information. Figure 4.2 outlines the distinction between independent and joint prediction for this scenario. In Figure 4.2(a), the independent setting, the input is a single image, and the energy function is defined over the three possible classes: `dog`, `cat`, and `frog`. While in Figure 4.2(b), the joint setting, the input is a pair of images, and the energy function is defined for every possible combination of labels (e.g., (`dog`, `dog`), (`dog`, `cat`), etc.). The joint energy function of (b) leverages external information suggesting



(a) Independent Energy



(b) Joint-Energy

Figure 4.2: Example of non-joint and joint energy functions.

the images are of the same entity. Joint reasoning enables a model to make structured predictions that resolve contradictions an independent model could not detect.

For NeSy-EBMs, a joint energy function encodes dependencies between its output variables through its symbolic potentials. NeuPSL additionally benefits from scalable convex inference to speed up learning over a dependent set of output variables. As I will see in the experiments section, utilizing joint inference and learning in NeSy-EBMs not only provides a boost in performance but produces results that non-joint methods cannot (even with five times the amount of data).

4.1.6 Empirical Evaluation

I perform a series of experiments highlighting 1) the runtime of NeuPSL symbolic inference, 2) NeuPSL's ability to integrate neural and symbolic inference and learning, 3) the effectiveness of NeuPSL's joint inference and learning, and 4) NeuPSL's performance on Visual-Sudoku-Classification, a new NeSy task. NeuPSL is implemented using the open-source PSL software package and can be used with any neural network library (in this instance, I use TensorFlow). [3] In addition to NeuPSL, I include results for DeepProbLog (DPL) Manhaeve et al. [2021a], Logic Tensor Networks (LTNs) [Badreddine et al., 2022], and convolutional neural network baselines (CNNs).

The first three experiments are performed over two canonical NeSy tasks: **MNIST-Add1** and **MNIST-Add2** [Manhaeve et al., 2018]. These tasks extend the classic MNIST image classification problem [LeCun et al., 1998] by constructing addition equations using MNIST images

[3] Implementation details including hyperparameters, network architectures, hardware, and NeuPSL symbolic models are described in the appendix materials.

and requiring classification to be performed using only their sum as a label. For example, a **MNIST-Add1** addition is ($[\,] + [\,] = \mathbf{8}$), and a **MNIST-Add2** addition is ($[\,, \,] + [\,, \,] = \mathbf{41}$). Given n randomly selected MNIST images, I create $n/2$ **MNIST-Add1** or $n/4$ **MNIST-Add2** additions. I emphasize that individual MNIST images do not have labels, only the resulting sum. All results are averaged over ten splits constructed from a set of $10,000$ randomly sampled MNIST images.

Symbolic Inference and Runtime

I compare runtimes of a latent variable NeuPSL model, a constraint-based NeuPSL model, and a DPL model for both **MNIST-Add** tasks. Models are trained on additions constructed from $6,000$ MNIST images. Table 4.1 [4] shows the average inference times per addition and standard deviation across ten splits with the fastest approach bolded. The benefit of NeuPSL's convex symbolic inference as I consistently see a 2-3 times speedup for the fastest NeuPSL models. NeuPSL-Constraint requires significantly less runtime than NeuPSL-Latent for **MNIST-Add1**. This is because the instantiated latent model contains roughly twice the number of symbolic potentials and decision variables; hence, inference requires more time to converge. However, for **MNIST-Add2**, introducing additional latent variables results in a more concise set of symbolic potentials, and despite the latent model having more decision variables, the reduction in model size is enough to achieve significantly faster inference. For the remaining experiments, I report the results on NeuPSL-Constraint for **MNIST-Add1** and NeuPSL-Latent for **MNIST-Add2**. Note that inference

[4] Timing results includes both model instantiation and inference. In the interest of time, NeuPSL-Constraint was only run once for **MNIST-Add2**.

in neural baselines and LTNs for **MNIST-Add** are equivalent to making a feed forward pass in a neural network, and therefore not comparable to the complex symbolic inference done in DPL and NeuPSL. This trivial inference is very fast, but comes with a decrease in predictive performance that I will examine next.

	DPL	NeuPSL-Constraint	NeuPSL-Latent
MNIST-Add1	1.34e-2 ± 1.67e-4	**4.00e-3 ± 1.00e-4**	2.98e-2 ± 2.33e-4
MNIST-Add2	1.65e+3 ± 4.80e+2	8.76e+3 ± —.—[4]	**7.38e+2 ± 3.36e+1**

Table 4.1: Inference times (milliseconds) on test sets constructed from $10,000$ MNIST images.

Deep Learning and Symbolic Reasoning

I evaluate the predictive performance of NeuPSL, DPL, LTNs, and neural baselines on **MNIST-Add** with varying amounts of training data. Table 4.2[5] shows the average accuracy and standard deviation across ten splits. The best average accuracy and results within a standard deviation of the best are in bold. In all but two settings, NeuPSL is either the highest performing model or within a standard deviation of the highest performing model. Moreover, NeuPSL has markedly lower variance for nearly all amounts of training examples in both **MNIST-Add** tasks.

[5]Exact inference results are reported for DPL in most settings as in Manhaeve et al. (2021a). However, exact inference in both $50,000$ **MNIST-Add** experiments produce random predictions (likely due to a NaN in DPL's loss). For these settings I report the results of DPL's approximate inference as described in Manhaeve et al. [2021b], which generally produces slightly lower accuracy with higher variance compared to exact inference.

Method	MNIST-Add1			MNIST-Add2		
	Number of Additions					
	300	3,000	25,000	150	1,500	12,500
CNN	17.16 ± 00.62	78.99 ± 01.14	96.30 ± 00.30	01.31 ± 00.23	01.69 ± 00.27	23.88 ± 04.32
LTNs	69.23 ± 15.68	**93.90 ± 00.51**	80.54 ± 23.33	02.02 ± 00.97	71.79 ± 27.76	77.54 ± 35.55
DPL	**85.61 ± 01.28**	92.59 ± 01.40	74.08 ± 15.33[5]	**71.37 ± 03.90**	**87.44 ± 02.15**	92.23 ± 01.01[5]
NeuPSL	82.58 ± 02.56	**93.66 ± 00.32**	**97.34 ± 00.26**	56.94 ± 06.33	**87.05 ± 01.48**	**93.91 ± 00.37**

Table 4.2: Test set accuracy and standard deviation on **MNIST-Add**.

Joint Inference and Learning

To quantify the benefit of joint learning and inference, I propose variants for **MNIST-Add** in which digits are re-used across multiple additions. Figure 4.3 illustrates the new information that joint models can leverage to narrow the space of possible labels when MNIST images are re-used. Since the same MNIST image of a zero appears in two addition samples, the value it takes must satisfy both, i.e., the possible label space can no longer include a two or three as it would violate the second addition.

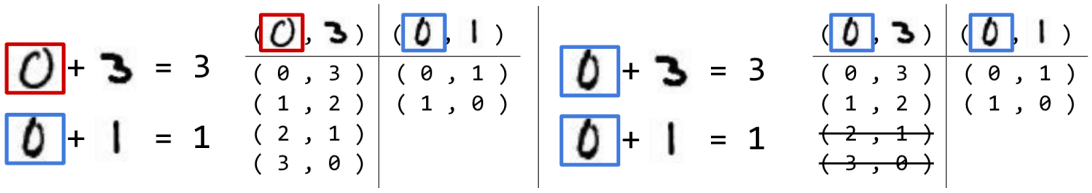


Figure 4.3: Example of overlapping MNIST images in **MNIST-Add1**. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros.

To create overlap (reused MNIST images), I begin with a set of n unique MNIST images

from which I re-sample to create a collection of $(n+m)/2$ **MNIST-Add1** and $(n+m)/4$ **MNIST-Add2** additions. I vary the amount of overlap with $m \in \{0, n/2, n, 2n\}$ and compare model performance in low data settings, $n \in \{40, 75, 150\}$. In low data settings there is not enough structure from the original additions for symbolic inference to discern the correct digit labels for training the neural models. Results are reported over test sets of $10,000$ MNIST images with overlap proportional to the respective train set.

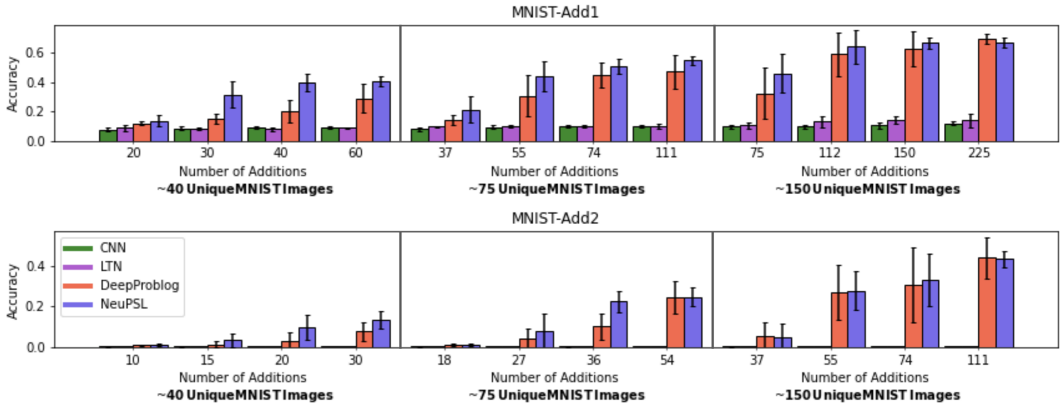


Figure 4.4: Average test set accuracy and standard deviation on **MNIST-Add** datasets with varying amounts of overlap.

Figure 4.4 shows the accuracy and standard deviation of all methods with varying amounts of overlap. Even though the number of unique MNIST images remains the same, as the number of additions increases, both DPL and NeuPSL are able to take advantage of the added joint information to improve their predictions. In almost all cases, NeuPSL has the best performance. Furthermore, NeuPSL has a lower standard deviation compared to DPL. LTNs and the CNN baseline benefit the least from joint information, likely this is a consequence of both learning and inference being performed independently across batches of additions.

Visual Sudoku Classification

Finally, I introduce **Visual-Sudoku-Classification**, a new NeSy task where 4x4 Sudoku puzzles are constructed using unlabeled MNIST images, and the task is to identify whether a puzzle is correct, i.e., has no duplicate digits in any row, column, or square. Unlike Wang et al. [2019], this task does not require learning the underlying label for images but rather whether an entire puzzle is valid. For instance, $\left[\,\mathbb{3}\,\right]$ does not need to belong to a "3" class, instead $\left[\,\mathbb{3}\,\right]$ and $\left[\,\mathbb{4}\,\right]$ need to be labeled as different. In this setting, n MNIST images creates $n/16$ **Visual-Sudoku-Classification** puzzles.

Method	Number of Puzzles			
	10	20	100	200
CNN-Visual	51.09 ± 15.78	51.36 ± 03.93	50.09 ± 02.17	51.23 ± 02.20
CNN-Digit	55.45 ± 05.22	54.55 ± 07.57	71.09 ± 04.99	82.22 ± 01.77
NeuPSL	**64.54 ± 12.93**	**72.27 ± 06.84**	**84.72 ± 03.61**	**85.05 ± 02.65**

Table 4.3: Test set accuracy and standard deviation on **Visual-Sudoku-Classification**.

I compare NeuPSL with two CNN baselines, CNN-Visual and CNN-Digit. CNN-Visual takes as input the pixels for a Sudoku puzzle and outputs the probability of the puzzle being valid. To verify whether the neural model is able to learn the Sudoku rules, I additionally introduce the CNN-Digit baseline that also outputs a probability of the puzzle being valid, but is provided all sixteen digit labels. Table 4.3 shows the test set accuracy and standard deviation of results

with the best performing method in bold. NeuPSL consistently outperforms both neural baselines. Furthermore, even when CNN-Digit is provided as input the sixteen digit labels, NeuPSL is able to outperform this baseline using half the amount of puzzles.

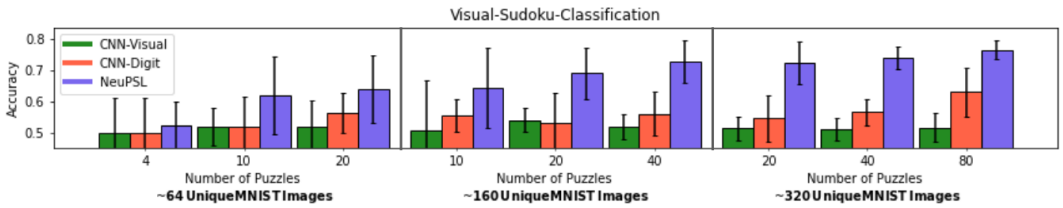


Figure 4.5: Average test set accuracy and standard deviation on **Visual-Sudoku-Classification** dataset with varying amounts of overlap.

Furthermore, I evaluate performance after adding relational structure in a similar fashion to the **MNIST-Add** variations. Figure 4.5 shows the accuracy and standard deviation of NeuPSL and CNN models on **Visual-Sudoku-Classification** with varying amounts of overlap. NeuPSL is efficient at leveraging the joint information across training examples, and is able to create a ~65% puzzle classifier using only about 64 MNIST images across 20 puzzles. This is a particularly impressive result as this performance implies the neural network in the NeuPSL model was trained to be a ~89% 4-digit distinguisher without any digit labels. Additionally, CNN-Visual is unable to generalize even in the highest settings (3200 MNIST images across 200 puzzles).

4.2 Negative Weights

To ensure scalable inference, PSL enforces constraints on the HL-MRF density function, and hence the structure of the rules. These constraints limit the expressivity of the framework. In

this section I address the PSL constraint restricting the sign of weights of rules to be non-negative. This constraint is designed to preserve the convexity properties of MAP inference. Other SRL frameworks, such as Markov logic networks (MLN) Richardson and Domingos [2006], are not concerned with the convexity properties of the MAP inference problem and can support negative weights Niu et al. [2011], Noessner et al. [2013]. In MLNs, a negative weight indicates that the negation of the associated rule should be satisfied. Since MLN's work with discrete valued variables, the negation of a rule is defined using Boolean logical semantics. PSL, on the other hand, translates rules to potentials by measuring the distance to satisfaction with Łukasiewicz real-valued logic Klir and Yuan [1995]. Instantiating a potential of a negated PSL rule using Łukasiewicz semantics results in concave potentials being added to the MAP objective function, breaking the convexity of inference.

Prior to the work discussed in this section, there have been two approaches proposed for supporting negative weights in PSL [Bach et al., 2017]. The first is to bias all user provided weights by a large enough constant to ensure non-negativity. This approach admits a convex MAP inference problem, however the semantics may not align with user expectations. The instantiated potentials do not measure the distance to satisfaction of the rules as one might expect. Another approach is to replace the non-convex potentials with multiple convex ones that have the same combined loss for critical variable assignments. This approach also results in a convex MAP inference problem, but may have unexpected properties for some variable values. Furthermore, this approach has scalability issues as the resulting PGM can grow exponentially with respect to the size of the rules. A larger HL-MRF model implies both slower MAP inference, as there are a larger number of potentials to

optimize over, and more memory consumption.

In this section, I propose three new approaches for supporting negative weights in PSL. The first instantiates potentials using an un-modified form of the rule and allows the weights to be negative in the objective. This method results in a MAP inference problem that is non-convex but admits an objective that can be easily separated into a difference of convex functions. The second method negates the negatively weighted rule and instantiates a potential using Łukasiewicz logic. This method again breaks the convexity of inference, but I show in special-cases that this method can be equivalent to the first. Lastly, I propose a method which similarly negates the negatively weighted rule, but instead instantiates a potential using Gödel logic Klement et al. [2000]. The conjunction semantics of Gödel logic is defined by the concave function: $T_{\min}(x, y) = \min\{x, y\}$. Using this property I show that potentials instantiated from rules negated with Gödel logic preserve convexity and faithfully measure distance to satisfaction. I analyze and compare the convexity and scalability properties of the two previously proposed and the three novel methods. In addition, I develop a taxonomy of the negative weight methods and show when the methods are equivalent and when they differ. I introduce synthetic dataset generator designed to evaluate the impact of negative weights and measure the effectiveness of each approach.

The work covered in this section was presented at the 2021 Workshop on Tractable Probabilistic Modeling (TPM) Dickens* et al. [2021].

4.2.1 Related Work

Increasing the expressivity of SRL formalisms is an active direction of research. Probabilistic logic programs (PLP) define a distribution over a set of logical clauses [Raedt et al., 2007, Vennekens et al., 2009, Sato and Kameya, 1997]. Meert and Vennekens (2014) extend CP-Logic, a causal PLP, by defining semantics for negations in the head of rules. The negations capture inhibition effects in the model, i.e., the inclusion of rules which can decrease the probability of an event. Based on similar motivations, an interpretation of negative probability weighted rules in PLPs was introduced by Buchman and Poole (2017a). The authors show that the semantics they introduce for negative probabilities capture the same relations as PLPs with negations and more. Moreover, they show that PLPs, even with negative weight semantics, are incapable of representing some relational distributions. Buchman and Poole (2017b) develop this line of research by showing show that PLPs with complex valued parameters are fully expressable.

Markov logic networks (MLN) use weighted logical rules to define probability distributions over binary $\{0, 1\}$ valued variables Richardson and Domingos [2006]. In MLNs, Boolean logic is used to define potentials and a negative weight is interpreted as a negation of the rule. Kuželka (2020) shows that the expressivity of the MLN framework is limited, i.e., only certain families of distributions can be represented by MLNs. The authors show that by allowing weights to take complex values, MLNs are fully expressive for binary valued random variables.

4.2.2 Methodology

In this section I remove the non-negativity constraint on the weights, $\mathbf{w}$, that PSL uses to define a HL-MRF. A negative weight has a direct impact on the structure of the HL-MRF conditional distribution and hence the MAP inference problem. I introduce five different approaches for interpreting negative weights in PSL. At a high level, the approaches are classifiable as either methods that consider weights independently from grounding or ones that modify the potential instantiation process by using the sign of the weight as an indicator to negate the rule. I refer to the former class of approaches as *weight based* and the latter as *negation based*.

To illustrate each interpretation, I use the following simple PSL model designed to predict unobserved instances of the atom $\mathrm{Q}(\mathtt{A})$:

$$w_1 : !\mathrm{Q}(\mathtt{A})\hat{}2$$

$$w_2 : \mathrm{P}(\mathtt{A}) \rightarrow \mathrm{Q}(\mathtt{A})\hat{}2$$

The first rule is a squared negative prior on $\mathrm{Q}(\mathtt{A})$ that encourages predictions near 0. Then the second rule is a squared implication that implies values for $\mathrm{P}(\mathtt{A})$ can be used as predictive signal for $\mathrm{Q}(\mathtt{A})$. A dataset with just a single entity $A = \{a\}$ is used to instantiate the PSL model. The grounding process will therefore create the following ground rules:

$$w_1 : !\mathrm{Q}(a)\hat{}2 \tag{4.7}$$

$$w_2 : \neg\mathrm{P}(a) \vee \mathrm{Q}(a)\hat{}2 \tag{4.8}$$

Throughout this discussion, let $\mathrm{P}(a)$ be observed with the truth value $x = 0.75$ and let y be the

random variable corresponding to the atom Q(a). Finally, I assume that $w_2 < 0$ and $w_1 >= 0$.

Weight-Based Approaches

For weight-based approaches, the grounding process described in Section 2.3 is unchanged in the presence of a negative weight. Thus the potential functions for the two rules are:

$$\phi_1(y, x) = [1 - (1 - y)]^2 = y^2$$

$$\phi_2(y, x) = (max\{1 - y - (1 - x), 0\})^2$$

$$= (max\{0.25 - y, 0\})^2$$

Therefore, the objective function, $f(\mathbf{y}, \mathbf{x})$, for this model is:

$$f(\mathbf{y}, \mathbf{x}) = w_1\phi_1(y, x) + w_2\phi_2(y, x)$$

$$= w_1y^2 + w_2(max\{0.25 - y, 0\})^2 \tag{4.9}$$

Weight-Based Approach: Remove Non-Negativity Constraint on Weights

The first approach is to simply remove the non-negativity constraint on the weights in Definition 4. When weights are not constrained to be non-negative the HL-MRF MAP inference objective is no longer necessarily a positive sum of convex functions and is therefore not guaranteed to be convex. This behavior is demonstrated in the plot of the objective function for the example PSL model in Figure 4.6.

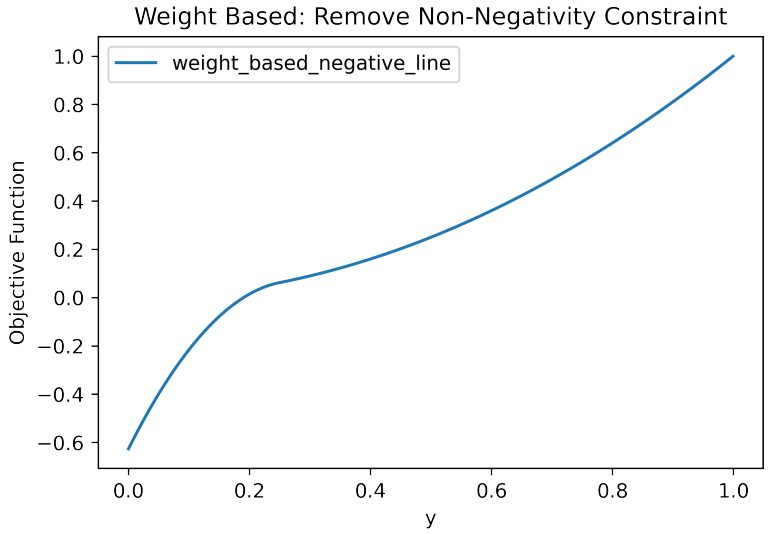


Figure 4.6: Example non-convex MAP inference objective function for a negative weight PSL model. The negative weight is interpreted using the approach described in Section 4.2.2. The weights for the model in Equation 4.8 are set to $w_1 = 1$ and $w_2 = -10$.

Though the objective is non-convex, it is expressable as a difference of convex (DC) functions Hartman [1959]. It is generally challenging to find a decomposition of an objective as a DC function. However, in this case the decomposition of the objective into a sum of a convex and concave functions comes naturally. Let Φ^+ and Φ^- index the positive and negative weighted hinge loss potentials, respectively. Then the HL-MRF MAP inference objective can be expressed as:

$$\underset{\mathbf{y}|(\mathbf{y},\mathbf{x})\in\Omega}{\arg\min} \sum_{i\in\Phi^+} w_i\phi_i(\mathbf{y},\mathbf{x}) - \sum_{i\in\Phi^-} (-w_i)\phi_i(\mathbf{y},\mathbf{x}) \tag{4.10}$$

The terms $\sum_{i\in\Phi^+} w_i\phi_i(\mathbf{y},\mathbf{x})$ and $\sum_{i\in\Phi^-}(-w_i)\phi_i(\mathbf{y},\mathbf{x})$ are both convex. MAP inference can thus be solved using the convex-concave procedure (CCCP) introduced by Yuille and Rangarajan (2003) and extended by Lipp and Boyd (2016). This approach would result in efficient optimization with convergence guarantees for finding local optimal solutions.

Weight-Based Approach: Biased Weights

Another weight-based approach, initially suggested by Bach et al. (2017), is to add a sufficiently large constant to make all weights positive. More formally, a non-negative parameter ϵ is introduced and all weights are biased by $\delta = \min(w_1, \cdots, w_m, 0) - \epsilon$. For instance, the resulting objective of the running example using this method is

$$f(\mathbf{y}, \mathbf{x}) = (w_1 - \delta)y^2 + (w_2 - \delta)(max\left\{0.25 - y, 0\right\})^2 \tag{4.11}$$

Biasing all weights in this way guarantees MAP inference is convex but results in an objective function with potentially different optimal solutions than that of the approach described in Section 4.2.2. To illustrate this behavior, consider the weight-based objective of the running example Equation 4.9. The derivative is:

$$\frac{df(\mathbf{y}, \mathbf{x})}{dy} = \begin{cases} 2w_1 y & y >= 0.25 \\ 2w_1 y + 2w_2(0.25 - y) & y < 0.25 \end{cases}$$

Choosing weights $w_1 = -5$ and $w_2 = -10$, by evaluating the critical points of this objective I find that it is minimized at $y = 1/6$. However, after adding a sufficiently large constant $\delta = \min(-5, -10, 0) - \epsilon = -(10 + \epsilon)$ to the weights, the objective is minimized at $y = 0.0$. This result demonstrates that the solution set of the MAP inference problem is generally not invariant to translations of the rule weights.

Negation-Based Approaches

Negation-based approaches modify the potential instantiation process, i.e., grounding. These approaches interpret negative weights as an indication to negate the corresponding rule. For instance, in the running example, the approaches described in this section first negate the rule with the negative weight w_2:

$$\mathrm{P}(a) \wedge \neg \mathrm{Q}(a) \tag{4.12}$$

Then, a potential is instantiated from the negated rule with an associated weight $|w_2|$. The differences between theses approaches comes from how the instantiated potentials measure the distance to satisfaction of this negated rule.

Negation-Based Approach: Łukasiewicz Negation

The first negation-based approach to negative weights directly applies Łukasiewicz logical semantics to define potentials. The degree of truth of the example negated rule, (4.12), using Łukaseiwicz logic is:

$$\max\{(x + (1 - y) - 1, 0\} \tag{4.13}$$

The distance to satisfaction of this rule, i.e., the potential that would be instantiated, is thus

$$\phi_2(y, x) = (1 - \max\{x + (1 - y) - 1, 0\})^2 \tag{4.14}$$

$$= \min\{1 + y - x, 1\}^2 \tag{4.15}$$

The surface plot of this potential as a function of x and y is shown in Figure 4.7. Notice that this potential is neither a convex nor a concave function of x and y. Adding this potential to the MAP objective breaks the convexity of inference. If, on the other hand, the potential was not squared, then it would be concave and the objective could be decomposed into a sum of concave and convex functions, i.e., a DC function. In this case the CCCP could be applied in the same way as described in Section 4.2.2. In fact, the relation between this approach and the approach discussed in Section 4.2.2 goes deeper. MAP inference for PSL models with Łukasiewicz negated implications and MAP inference with negative weighted potentials is equivalent when the rules are non-squared. That is to say, for non-squared rules, MAP inference, when one allows weights to be negative, is equivalent to allowing logical rules of the form: $w : A \wedge \neg B$, where A and B are arbitrary conjunctive and disjunctive clauses.

Theorem 1. *Let $A = (A_1 \wedge A_2 \wedge \cdots \wedge A_{n_A})$ be a conjunction of n_A (possibly negated) atoms and $B = (B_1 \vee B_2 \vee \cdots \vee B_{n_B})$ be a disjunction of n_B (possibly negated) atoms. Let $w > 0$. The MAP inference problem for a PSL model with the non-squared rule $w : A \wedge \neg B$ is equivalent to the MAP inference problem for a PSL model with the non-squared rule $-w : A \implies B$.*

Proof. I need to show both directions for equivalence.

($\implies$) Suppose I have a PSL model with the rule $w : A \wedge \neg B$ such that $w > 0$. This

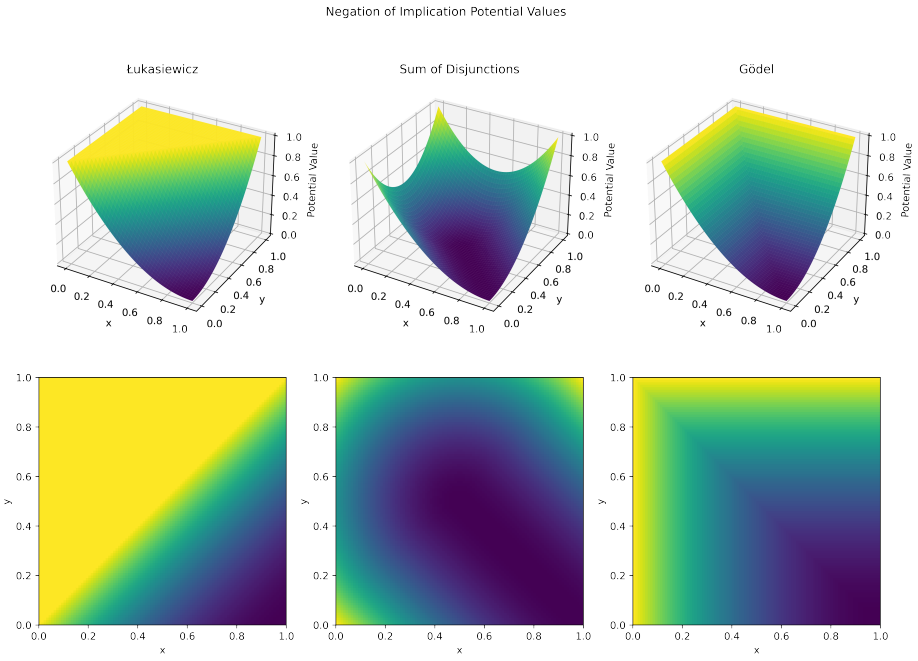


Figure 4.7: Surface plots and heat maps of the potential values instantiated by the example negated rule (4.12) using the three negation-based approaches.

rule is translated into a weighted potential of the form:

$$w\phi(\mathbf{y}) = w\phi_{A\wedge\neg B}(\mathbf{y}) \tag{4.16}$$

$$=w(1-\max\left\{\sum_{i=1}^{n_A} y_{A_i} + \sum_{i=1}^{n_B}(1-y_{B_i}) - (n_A+n_B-1), 0\right\}) \tag{4.17}$$

$$=^{*} w - w\max\left\{\sum_{i=1}^{n_A} y_{A_i} - \sum_{i=1}^{n_B} y_{B_i} - (n_A-1), 0\right\} \tag{4.18}$$

$$=^{**} w - w\max\left\{1 - \sum_{i=1}^{n_A}(1-y_{A_i}) - \sum_{i=1}^{n_B} y_{B_i}, 0\right\} \tag{4.19}$$

* The constant term in the summation corresponding to B cancels with the n_b term.

** Move n_A into summation corresponding to A.

Denote the potential instantiated by $A \to B$ by $\phi_{A\to B}(\mathbf{y})$. Note that (4.19) is equivalent to $w - w\phi_{A\to B}(\mathbf{y})$. Let t_- and t_+ be the index sets for the potentials instantiated by the rules $w : A \wedge \neg B$ and $w : A \to B$, respectively. Then MAP inference becomes:

$$\min_{\mathbf{y}\in[0,1]} \sum_{i\in t_+} w_i\phi_i(\mathbf{y}) + \sum_{i\in t_-} w_i\phi_{A\wedge\neg B,i}(\mathbf{y}) \tag{4.20}$$

$$= \min_{\mathbf{y}\in[0,1]} \sum_{i\in t_+} w_i\phi_i(\mathbf{y}) + \sum_{i\in t_-} w_i - w_i\phi_{A\to B,i}(\mathbf{y}) \tag{4.21}$$

$$= \min_{\mathbf{y}\in[0,1]} \sum_{i\in t_+} w_i\phi_i(\mathbf{y}) - \sum_{i\in t_-} w_i\phi_{A\to B,i}(\mathbf{y}) \tag{4.22}$$

($\Longleftarrow$) The other direction follows similarly. □

Negation-Based Approach: Conjunction to Sum of Disjunctions

Bach et al. (2017) propose a second method for supporting conjunctive rules of the form

$w : A \wedge B\hat{}2$ while preserving convexity. As the negative form of an implication has this structure, shown by the example in (4.12), this technique can be applied to support negative weights. They first express the rule as a CNF with the following structure:

$$A \wedge B = (A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B) \tag{4.23}$$

Each disjunction is then used to define a unique potential using Łukasiewicz semantics. For instance the potentials that are instantiated for the example negatively weighted rule, (4.12), are:

$$\begin{aligned}
\phi_{2,1}(y, x) &= (1 - \min\{x + (1 - y), 1\})^2 \\
\phi_{2,2}(y, x) &= (1 - \min\{(1 - x) + (1 - y), 1\})^2 \\
\phi_{2,3}(y, x) &= (1 - \min\{x + y, 1\})^2
\end{aligned} \tag{4.24}$$

Each instantiated potential is assigned a weight equal to the positive value of the original rule and added to the HL-MRF energy function. The additive loss of the potentials instantiated from each of the disjunctive terms is the same as the single conjunction for discrete variable value assignments. This property can be seen in Table 4.4. However, Figure 4.7 shows that the potential values corresponding to Łukasiewicz conjunction and the three disjunctions are different for non-discrete variable value assignments. The number of additional potentials instantiated by this procedure grows exponentially with the number of atoms involved in the rule. Thus, for complex rules involving many atoms, this method can run into scalability issues.

A	B	$\phi(A \wedge B)$	$\phi(A \vee B)$	$\phi(\neg A \vee B)$	$\phi(A \vee \neg B)$
1	1	0	0	0	0
1	0	1	0	1	0
0	1	1	0	0	1
0	0	1	1	0	0

Table 4.4: A truth table showing the value of the potential function of a conjunction and the three disjunctions that can represent it.

Negation-Based Approach: Gödel Negation

Another way to preserve convexity but still support conjunctive rules is to leverage the weak conjunction semantics of Łukasiewicz logic, also known as the Gödel logic Klement et al. [2000]. The degree of truth of the conjunctive rule $A \wedge B$ using Gödel semantics is:

$$\min\{A, B\} \tag{4.25}$$

This is a concave function of A and B, and hence the distance to satisfaction of the clause is guaranteed to be convex. For instance, the potential that would be instantiated for the example negative weighted rule in Equation 4.8 is:

$$\phi_2(y, x) = \max\{1 - x, y\}^2 \tag{4.26}$$

The plot of the potential in Equation 4.26 for the example negative weighted rule relative to the potential instantiated using Łukasiewicz strong conjunction semantics and the sum of disjunctions method is shown in Figure 4.7. Notice that for x fixed as 0 or 1, Lukasiewicz and Gödel potentials

are the same. Then as the value for x gets closer to 0.5 the potential functions deviate significantly. In general, values of potentials instantiated using Łukasiewicz semantics are equivalent to those instantiated using Gödel semantics for discrete variable value assignments. Moreover, no additional potentials need to be instantiated to achieve this behavior, as was the case in Section 4.2.2.

4.2.3 Empirical Evaluation

In this section, I evaluate the predictive performance and structural properties of PSL models instantiated using the different approaches to supporting negative weights discussed in Section 4.2.2. I answer the following questions for each method of interpreting the negative weighted rule: Q1) How does the method effect the predictive performance of the PSL model? and Q2) How does the method scale in terms of both the size of the instantiated model and the rate of convergence of PSL inference?

I evaluate all negative weight approaches on a collective prediction task. The following PSL model is used for all experiments and is designed to predict the unobserved ground $\textsc{Target}(\texttt{X}, \texttt{Y})$ atoms.

$$-1.0 : \textsc{Predictor1}(\texttt{X}, \texttt{Y}) \rightarrow \textsc{Target}(\texttt{X}, \texttt{Y})\texttt{\^{}2} \tag{4.27}$$

$$0.1 : \textsc{Target}(\texttt{X}_1, \texttt{Y}) \wedge \textsc{Similar}(\texttt{X}_1, \texttt{X}_2) \rightarrow \textsc{Target}(\texttt{X}_2, \texttt{Y})\texttt{\^{}2} \tag{4.28}$$

$$1.0 : \textsc{Predictor2}(\texttt{X}, \texttt{Y}) = \textsc{Target}(\texttt{X}, \texttt{Y})\texttt{\^{}2} \tag{4.29}$$

$$0.01 : \neg\textsc{Target}(\texttt{X}, \texttt{Y})\texttt{\^{}2} \tag{4.30}$$

The first rule in the model, 4.27, is the only negative weight rule. The atom $\textsc{Predictor1}(\texttt{X}, \texttt{Y})$

represents an abstract atom for predicting the target atom. Different approaches to interpreting this rule result in a different ground HL-MRF and hence a different MAP inference objective. The remaining rules are common modelling patterns. Each method I have described grounds the same potentials for these rules. The second rule in the model, 4.28, is a rule for propagating predictions or observed values of the $\text{TARGET}(\texttt{X}, \texttt{Y})$ atoms for similar entities $\texttt{X}_1$ and $\texttt{X}_2$. This rule can be read as "if $\texttt{X}_1$ and $\texttt{X}_2$ are similar then the value of $\text{TARGET}(\texttt{X}_1, \texttt{Y})$ should be close to that of $\text{TARGET}(\texttt{X}_2, \texttt{Y})$". The third rule, 4.29, is referred to as a local predictor rule. This rule uses an external model, the local predictor, as a signal for predicting the $\text{TARGET}(\texttt{X}, \texttt{Y})$ atoms. Specifically, this rules says that the local predictor value for $\texttt{X}, \texttt{Y}$ should be close to PSL's prediction for $\text{TARGET}(\texttt{X}, \texttt{Y})$. The final rule in the model, 4.30, acts as a negative prior. That is to say, this rule will result in a small loss for any non-zero prediction made for $\text{TARGET}(\texttt{X}, \texttt{Y})$. This rule can also be thought of regularizer and is commonly used to get more stable predictions from PSL since it ensures a strongly convex MAP inference objective.

Synthetic Dataset Generation

I generated three synthetic datasets to compare the properties and performance of HL-MRF models instantiated using the approaches discussed in Section 4.2.2. The three datasets are designed to represent a different distribution of the target atoms $\text{TARGET}(\texttt{X}, \texttt{Y})$. The first dataset is generated such that the target atoms have discrete, $\{0, 1\}$, truth values. The second and third datasets are generated such that the target atoms have real, $[0, 1]$, truth values with different distributions.

The generation process for all the datasets follows the same general pattern. There are a

total of 100 possible unique values that the X argument can be assigned and 1000 for the Y argument. First, the values for X are randomly clustered into 10 groups. Then, an ideal value for TARGET$(\cdot, \mathtt{Y})$ is generated for every possible value of Y for every group of X arguments. Values for TARGET$(\mathtt{X}, \mathtt{Y})$ are generated by adding noise to the ideal group values corresponding to the group assignment of X. The TARGET$(\mathtt{X}, \mathtt{Y})$ data is then split into an observed and test set. The cosine similarity of the observed TARGET$(\mathtt{X}, \mathtt{Y})$ values across the Y arguments for each X is used to define the observed SIMILAR$(\mathtt{X}_1, \mathtt{X}_2)$ atoms. The observed local predictor atoms PREDICTOR2$(\mathtt{X}, \mathtt{Y})$ are generated by adding Gaussian, $\mathcal{N}(0, 0.3)$, noise to the test TARGET$(\mathtt{X}, \mathtt{Y})$ values. Finally, the PREDICTOR1$(\mathtt{X}, \mathtt{Y})$ atom that is involved in the negative weighted rule is generated for all $(\mathtt{X}, \mathtt{Y})$ arguments by first drawing a uniform $(0, 1)$ random value. Then, if the random value is less than $(1 -$ TARGET$(\mathtt{X}, \mathtt{Y}))$, the corresponding PREDICTOR1$(\mathtt{X}, \mathtt{Y})$ atom is set to a uniform $(0, (1 -$ TARGET$(\mathtt{X}, \mathtt{Y})))$ random variable. Otherwise PREDICTOR1$(\mathtt{X}, \mathtt{Y})$ is set to PREDICTOR1$(\mathtt{X}, \mathtt{Y}) + \alpha$ where α is an exponential, $Exp(\beta = 0.05)$, random variable. In this way, PREDICTOR1$(\mathtt{X}, \mathtt{Y})$ can be used as a possibly noisy lower bound on PREDICTOR1$(\mathtt{X}, \mathtt{Y})$.

The difference between the three datasets occurs when the true TARGET$(\mathtt{X}, \mathtt{Y})$ atoms are generated. For the *Discrete targets* dataset, the ideal TARGET$(\cdot, \mathtt{Y})$ values for each X group are generated as independent Bernoulli, $Bern(p = 0.4)$, random variables. Then noise is added to the ideal values to generate TARGET$(\mathtt{X}, \mathtt{Y})$ values for each $(\mathtt{X}, \mathtt{Y})$. The ideal target value of the group X belongs to is flipped with probability $p = 0.3$. For the *Uniform Real Targets* dataset, TARGET$(\cdot, \mathtt{Y})$ values for each X group are generated as independent uniform, $\mathbb{U}(0, 1)$, random variables. Similarly, for the *Centered Real Targets* dataset TARGET$(\cdot, \mathtt{Y})$ values for each X group

are generated as independent Gaussian, $\mathcal{N}(\mu = 0.4, \sigma = 0.1)$, random variables. Then for both the Uniform Real Targets and Centered Real Targets datasets, $\textsc{Target}(\mathtt{X}, \mathtt{Y})$ values are generated by perturbing the ideal value of the group $\mathtt{X}$ belongs to with $\mathcal{N}(\mu = 0, \sigma = 0.1)$ noise.

Results

I first answer question Q1 by running each proposed negative weight approach on the three synthetic datasets and compare the RMSE of their predictions. Ten independent folds of each variation of the synthetic dataset is generated. Then, using the model introduced in this section, an HL-MRF is instantiated using one of the five methods discussed in Section 4.2.2 and MAP inference is performed. MAP inference optimization is solved via ADAM stochastic gradient descent (SGD) Kingma and Lei Ba [2015]. The stepsize hyperparameter is set to $\alpha = 0.1$ and the exponential decay rate parameters are set to the suggested defaults of $\beta_1 = 0.9$ and $\beta_2 = 0.999$. ADAM SGD is determined to converge when the difference in the objective value over a single epoch drops below a tolerance of 10^{-8}. The results of this experiment are shown in Figure 4.8.

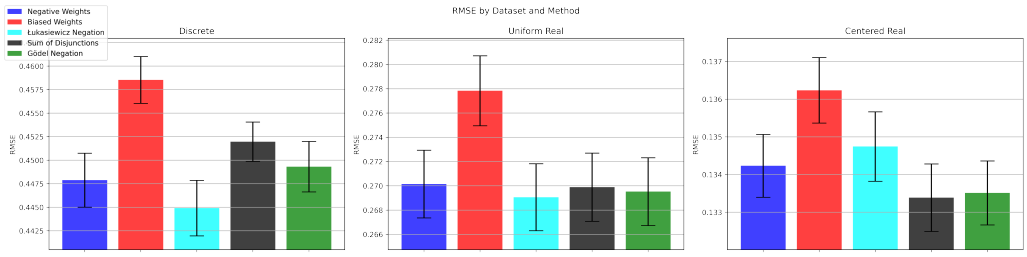


Figure 4.8: Bar plot of the mean RMSE of all five methods for supporting the negative weights in the experiment model for ten folds of each of the three variations of the synthetic datatset. One standard deviation is shown with error bars.

Across all variations of the dataset the Gödel negation method is always among the top

three performing approaches. This consistency is not seen for any of the other methods. This result is particularly encouraging as Gödel negation does not sacrifice the convexity properties of PSL MAP inference as does the methods of Łukasiewicz negation and negative weights. On the other end of spectrum, biased weights is consistently the worst performing method. This behavior can be explained by the fact that this method does not capture the relation between the instantiated PREDICTOR1(X, Y) and TARGET(X, Y) atoms. The synthetic dataset is designed so that PREDICTOR1(X, Y) is a reliable lower bound for $1 -$ TARGET(X, Y). However the instantiated potentials for the rule 4.27 using the biased weights method encourages solutions where the values for TARGET(X, Y) are greater than or equal to PREDICTOR1(X, Y) atoms.

I address question Q2 by examining the size of the ground models and the convergence properties of ADAM SGD on the MAP inference problem. Table 4.5 shows the mean and standard deviation of the number of epochs over the set of instantiated potentials that is required to converge to a local optimal solution of the MAP inference problem for each of the five negative weight methods. This table shows that the method of biased weights consistently requires a larger number of epochs to converge to a MAP state. This behavior can be explained by the fact that the method can create an ill-conditioned problem where there is a larger variation in the magnitude of the weights associated with each potential. Another interesting observation from the table is that the Łukasiewicz negation method can have a high variation in the number of epochs required to reach a MAP state. The Łukasiewicz negation method instantiates an objective that is neither convex nor concave and a local optimal solution can be difficult to find. The table also shows that the Gödel negation method, which preserves convexity of the MAP inference problem, is consistently

converging in a relatively low number of iterations.

Method	Discrete Epochs	Uniform Real Epochs	Centered Real Epochs
Negative Weights	33.7 (5.48)	22.3 (3.83)	16.4 (3.38)
Biased Weights	37.6 (6.13)	44.1 (5.20)	32.6 (5.06)
Łukasiewicz Negation	44.4 (5.08)	21.4 (4.14)	22.8 (8.88)
Sum of Disjunctions	24.9 (6.28)	26.0 (5.68)	19.5 (3.41)
Gödel Negation	24.6 (3.86)	28.0 (4.99)	20.6 (4.70)

Table 4.5: The mean and standard deviation of the number of epochs required to reach convergence of ADAM SGD optimization for each method and dataset.

Regarding the scale of the ground models, i.e., the number of instantiated potentials, the sum of disjunctions method is the only method that instantiates extra potentials. For reference, for the first fold of the synthetic dataset the sum of disjunctions method instantiated $79,744$ more potentials than the other four methods. This is a roughly 2% increase in model size, and this difference can increase exponentially with the number of atoms involved in the negative weighted rule. This has implications on both the time it takes to ground the model and the time to run inference. This is because each epoch of ADAM SGD optimization for the sum of disjunctions method has to optimize over more potentials.

4.3 Conclusions and Future Work

The work covered in this section allows SRL models to greatly expand the number and types of models that can be represented. Negative weights allows for more expressive models that indicate anti-relations, while the incorporation of neural models allows for the representation of a wide range of models that cannot be purely represented at the symbolic (logic) level.

For NeuPSL, there are many avenues for future work including exploration of different learning objectives, such as ones that balance traditional neural and energy-based losses, and new application domains. Each of these is likely to provide new challenges and insights. Additionally, the concepts in NeuPSL can be generalized and integrated with different SRL frameworks.

For negative weights, directions for future work include further exploration of alternate real-valued logical semantics that can be used for instantiating HL-MRF potentials. Another future direction is the integration of negative weight semantics into the weight learning process. Similarly, as negative weights increases the expressivity of PSL, there are certainly implications on the task of rule discovery in PSL, i.e., structure learning.

Chapter 5

Model Adaptability

In this chapter, I discuss my work in producing a SRL system that can support models that evolve over time[1]. In the simple case, working with an evolving model can mean tuning the hyperparameters of a model. In a more complex case, working with an evolving model may mean using a model whose distribution may change mid-inference, as in the online setting.

5.1 Weight Learning

Learning the weights of the rules is one of the key challenges for templated rule languages such as PSL and MLNs, since the weighted rules interact in complex ways and cannot be optimized independently. Because of their templated nature, the weights, i.e., the parameters of the model, are used in multiple places in the instantiated graphical model, and the context varies depending on the

[1]The work presented in this chapter was published in the Machine Learning Journal Srinivasan et al. [2021] and presented at the 2021 International Conference on Machine Learning (ICML) Dickens et al. [2021].

other rules that have been instantiated. In addition, the corresponding probability distribution is not easy to compute; specifically, computing the normalization constant is often intractable.

Typically, the weights of the rules are learned through maximizing some form of likelihood function [Bach et al., 2013, Lowd and Domingos, 2007, Singla and Domingos, 2005, Kok and Domingos, 2005, Chou et al., 2016, Sarkhel et al., 2016, Das et al., 2016, Farabi et al., 2018]. This is a well-motivated approach if the downstream objective makes use of the probability density function directly. However, the objective is to often improve an external domain metric such as accuracy, F1 for classification, or AUROC for ranking. Several approaches address this issue by augmenting the metric into a loss function and solving a max-margin problem [Huynh and Mooney, 2009, 2010a, Bach et al., 2013]. However, this does not directly optimize the desired metric but instead optimizes a surrogate loss. Further, such approaches do not easily extend to new metrics as they require deriving new losses, which may be non-convex and hard to optimize.

In this section, I introduce four search-based approaches for finding the best set of weights for a model (weight configuration) in weighted logic-based SRL frameworks. The key advantage of these approaches are that they directly optimize the chosen domain performance metric and, unlike other approaches, do not require re-derivation of the loss function for each metric. My proposed approaches are based on black-box optimization methods used in learning hyperparameters in other machine learning approaches [Claesen and Moor, 2015, Bergstra et al., 2011]. My first two approaches *random grid search for weight learning* (RGS) and *continuous random search for weight learning* (CRS) are based on simple yet powerful search approaches popularly used in tuning hyperparameters of deep learning models [Bergstra and Bengio, 2012]. While RGS is simple and its

effectiveness is determined by the human-specified grid, the effectiveness of CRS is determined by the new sampling space I introduce in this section. My next approach, *Hyperband for weight learning* (HBWL), is based on the Hyperband algorithm [Li et al., 2018b] that effectively distributes resources to perform efficient random search. Hyperband has been shown to efficiently allocate resources and maximize the search for the best solution. Finally, my fourth approach, *Bayesian optimization for weight learning* (BOWL), is based on Gaussian process regression (GPR) [Rasmussen and Williams, 2005] in a Bayesian optimization (BO) [Mockus, 1977] framework. BO is an effective approach for optimization of black-box functions [Lizotte et al., 2007, Martinez-Cantin et al., 2009, Srinivas et al., 2012, Brochu et al., 2010] and GPR is a non-parametric Bayesian approach that is often used to approximate arbitrary functions. GPRs have been used extensively with a lot of success for hyperparameter tuning in machine learning [Snoek et al., 2012].

In order to perform efficient and effective search of weights using these approaches, I introduce a new parameter search space, which I refer to as *scaled space* (SS), which is an accurate representation of the true weight space. I show that SS is both accurate and complete in representing the weights. Further, I also show that SS also takes into account the impact of model instantiation (also referred to as *grounding*). As sampling from SS is challenging, I introduce an approximation of SS which enables us to efficiently sample weight configurations to perform search-based weight learning. I develop all my search-based approaches for two powerful SRL frameworks, PSL and MLNs. I perform my empirical study on both PSL and MLNs to show the effectiveness of search-based approaches.

My key contributions to this component of the model adaptability pillar include: 1) I

generalize and introduce a new taxonomy of weight learning approaches in SRL frameworks by reformulating the problem of weight learning as a black-box optimization problem and introducing four search-based approaches, referred to as RGS, CRS, HBWL, and BOWL, to perform this optimization; 2) I introduce a new search space called the scaled space (SS) which I show is an accurate representation of the true weight space; 3) I introduce an approximation of SS which generalizes the search-based approaches and simplifies the process of sampling weight configurations in these methods; 4) I show that the search-based approaches are effective at learning weights in both PSL and MLNs and that these approaches outperform likelihood-based approaches on multiple datasets by up to 10%; and 5) finally I show scalability of search-based approaches and perform elaborate set of experiments to show that, of the four approaches, BOWL is robust to initializations, acquisition function (process of choosing next best point, explained in Section 10), and the hyperparameter used in the sampling of weight configurations.

The work covered in this section was published in the Machine Learning Journal Srinivasan et al. [2021].

5.1.1 Black-box Optimization

Here, I provide a brief overview of black-box optimization and Gaussian processes, which are essential in understanding my weight learning approaches introduced in this section. *Black-box optimization* is a well studied technique, which has been found to be especially useful in the context of hyperparameter tuning [Bergstra and Bengio, 2012, Shahriari et al., 2016].

Definition 5 (Black-box optimization)**.** *Given a black-box function* $\gamma(\tilde{\mathbf{x}}) : \mathbb{R}^d \rightarrow \mathbb{R}$*, where* d *is*

the input dimension, the task of finding an $\tilde{\mathbf{x}}$ *that yields the optimal value for* $\gamma(\tilde{\mathbf{x}})$ *in a predefined amount of time is called* black-box optimization.

The goal of black-box optimization is to find the best possible value for $\tilde{\mathbf{x}}$ that optimizes the function $\gamma(\tilde{\mathbf{x}})$ in a predefined amount of resources (generally, number of epochs or time). While this process can be embarrassingly parallel for some approaches, the general strategy can be defined through a sequential setup. The general approach for black-box optimization is to define a search space, choose a point $\tilde{\mathbf{x}}$, evaluate $\gamma(\tilde{\mathbf{x}})$ to update a model, and repeat this process until the resources have been exhausted. Finally, the $\tilde{\mathbf{x}}$ with the best $\gamma(\tilde{\mathbf{x}})$ is returned. The procedure is summarized in Algorithm 3. While the process of selection of $\tilde{\mathbf{x}}$ and evaluation of $\gamma(\tilde{\mathbf{x}})$ on different points can be simple and made to run in parallel for algorithms like RGS, CRS, and HBWL (introduced in Section 5.1.2), other algorithms like BOWL (also introduced in Section 5.1.2) use BO framework with Gaussian process to approximate γ and assume a serial setup to ensure optimal selection of the next point on which to evaluate.

In the context of BO, various strategies have been proposed to choose the next point to evaluate given the previous evaluations [Srinivas et al., 2010, Kushner, 1964, Mockus, 1977, Thompson, 1933]. Each strategy is encoded through an acquisition function α. The objective of these strategies is to minimize the number of epochs required to find the best solution. A simple black-box Bayesian approach iteratively obtains a point to explore from the acquisition function α using the prior distribution; then the function γ is evaluated to obtain a new outcome at that point which is then used to update the posterior. Gaussian process regression (GPR) is a non-parametric

Bayesian approach which is effective in performing black-box optimization in a BO framework.

Algorithm 3: Black-box optimization via search.

Result: $\tilde{\mathbf{x}}^*$: point with the best value for $\gamma(\cdot)$

$\tilde{\mathbf{X}} =$ search space;

while *stopping criteria not met* **do**

  choose a $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$;

  update the model of choice with $(\tilde{\mathbf{x}}, \gamma(\tilde{\mathbf{x}}))$;

  update $\tilde{\mathbf{x}}^*$ if current $\gamma(\tilde{\mathbf{x}})$ is better than previous value or based on the model

end

Gaussian Process Regression

A Gaussian process (GP) is fully characterized by its mean function μ_0 and either a positive definite covariance matrix $\mathbf{K}$ or a kernel function k. Consider a finite set of s inputs $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}_{1:s}$ and a random variable $g_i = \gamma(\tilde{\mathbf{x}}_i)$ representing the function γ evaluated at $\tilde{x_i}$ and let $\tilde{y}_i$ be the noisy output of the function. In GP, I assume that $\mathbf{g} = g_{1:s}$ is jointly Gaussian and $\tilde{y}_i$ given $\mathbf{g}$ is Gaussian. The generative model is of the form: $\mathbf{g}|\tilde{\mathbf{X}} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$, and $\tilde{\mathbf{y}}|\mathbf{g} \sim \mathcal{N}(\mathbf{g}, \sigma^2\mathbf{I})$, where $m_i = \mu_0(\tilde{\mathbf{x}}_i)$, $\mathbf{K}$ is an $(s \times s)$ positive definite matrix such that $K_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. Since the distributions are Gaussian and using the kernalization trick [Rasmussen and Williams, 2005], the

posterior mean and variance given a set of observed data can be written as:

$$\mu_s(\tilde{\mathbf{x}}_{s+1}) = \mu_0(\tilde{\mathbf{x}}_{s+1}) + k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(y - m)$$

$$\sigma_s(\tilde{\mathbf{x}}_{s+1}) = k(\tilde{\mathbf{x}}_{s+1}, \tilde{\mathbf{x}}_{s+1}) - k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}k(\tilde{\mathbf{x}}_{s+1})$$

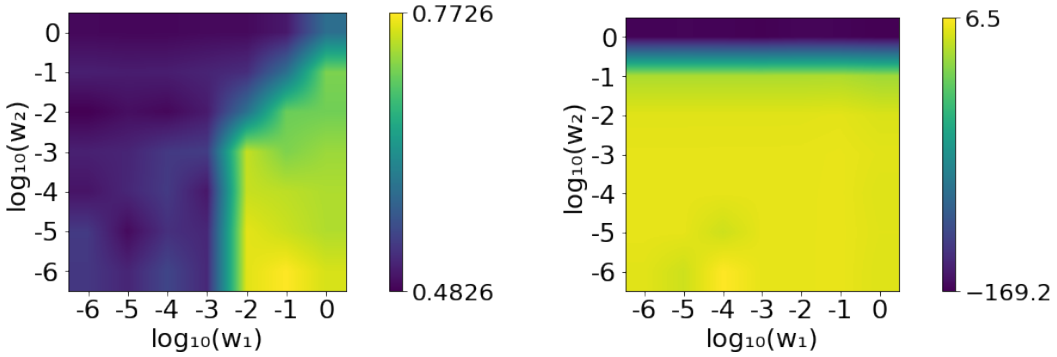
where $k(\tilde{\mathbf{x}}_{s+1})$ is a kernel function applied to the inputs with observed function evaluations and the new input; i.e., it represents the covariance between observed inputs and any unobserved input. Using the above expressions, the mean and variance for any point can be computed. There is a suite of kernel functions available in the literature [Rasmussen and Williams, 2005]. Note that the kernel function should be chosen based on the problem domain and it is often the key to finding the best approximation of the true function.

5.1.2 Search-Based Approaches for Weight Learning

As mentioned earlier, commonly used approaches for rule weight learning in SRL are generally based on maximizing a likelihood function. In this section, I first give a motivating example that highlights the issues with likelihood-based approaches and then I propose four search-based approaches to learn weights in SRL frameworks.

Motivating Example

Consider a simple classification model with two rules parameterized by their respective weights w_1 and w_2, applied to a toy dataset. Figure 5.1 shows the performance of the model in PSL as I vary the rule weights logarithmically from 10^{-6} to 1.0. Figure 5.1a shows AUROC and Figure



(a) AUROC (b) log-likelihood

Figure 5.1: Heat map for AUROC and log-likelihood for a model with two rules parameterized by their respective weights w_1 and w_2. The lighter color indicates higher values and higher values are desired for both metrics.

5.1b shows the log-likelihood of the model. Lighter shades (yellow) represent a high value and darker shades (dark blue) represent a low value. The AUROC is maximized when the first rule's weight is 0.1 and the second rule's weight is 10^{-6}. However, the likelihood is not maximized at these weights. For this simple model and dataset, the likelihood is not well correlated with the AUROC. While this behavior is shown using PSL, similar outcome can be seen when using MLN models as well.

Problem definition

Consider a SRL model with r template rules where each rule $i \in \{1 \dots r\}$ is associated with a weight $w_i \in \mathbb{R}^+$. Grounding all the rules with data D yields a set of m observed random variables $\mathbf{x} = \{x_1, \dots, x_m\}$, n unobserved random variables $\mathbf{y} = \{y_1, \dots, y_n\}$, and ι potentials $\phi = \{\phi_1, \phi_2, \dots, \phi_\iota\}$. The unobserved random variables $\mathbf{y}$ are inferred by optimizing Equation 2.4. Further, all unknown random variables are associated with corresponding ground truth $\mathbf{y}^* =$

$\{y_1^*, \ldots, y_n^*\}$ used to compute evaluation metrics. Let $\mathbf{w} = \{w_1, \ldots, w_r\}$ be the vector representing the set of rule weights, i.e., the weight configuration. Next, let $\omega(\mathbf{y}, \mathbf{y}^*) : (\mathbf{y}, \mathbf{y}^*) \rightarrow \mathbb{R}$ be a problem-specific evaluation metric (e.g., accuracy, AUROC, or F-measure) and let $\gamma(\mathbf{w}) : \mathbf{w} \rightarrow \omega(\mathbf{y}, \mathbf{y}^*)$ be the same function ω parameterized by $\mathbf{w}$ that maps weights to the metric. Then the objective of weight learning can be expressed as finding the set of weights that maximize the function γ which represents the true metric function ω, i.e., $\arg\max_{\mathbf{w}} \gamma(\mathbf{w})$. The objective of the search-based approaches is to find an approximate function $g \approx \gamma$ by sampling t weight configurations from a set of possible weight configurations $\mathbf{W}$ (the weight space). In order to perform this optimization, I introduce four approaches based on hyperparameter search methods in other areas of machine learning. The first two approaches are based on random grid search and continuous search used in deep learning [Bergstra and Bengio, 2012], the next approach is based on Hyperband algorithm used in statistical machine learning [Li et al., 2018b], and the last one is based on BO with GPR used for hyperparameter tuning in deep learning [Snoek et al., 2012].

Random Grid Search for Weight Learning

A straightforward search-based approach to weight learning is an exhaustive exploration over the set of weight configurations $\mathbf{W}$ generated through a user-specified grid of weights. The user-specified grid to generate weight configurations $\mathbf{W}$ is typically constructed by specifying a finite collection of v values $V = \{V_0, \cdots, V_v\}$ that can be assigned as weights for each of the rules, e.g., $V = \{0.01, 0.1, 1.0\}$. If a model contains r rules, then I can define $\mathbf{W}$ to be the r-ary Cartesian product of V, $\mathbf{W} = V \times \cdots \times V$, defining a grid with the intersections representing different weight

configurations. Then, for each configuration $\mathbf{w} \in \tilde{\mathbf{W}}$, $\gamma(\mathbf{w})$ is evaluated after performing MAP inference in the SRL model. Finally, the weight configuration with the highest $\gamma(\mathbf{w})$ is selected.

However, a comprehensive grid search is usually infeasible due to the combinatorial explosion in the size of the grid; if a model contains r rules where each rule can take on one of v possible values, then $|\mathbf{W}| = v^r$. Thus, to make the approach tractable (as mentioned in Algorithm 4), I uniformly draw t unique samples from $\mathbf{W}$ to stay within an established budget of resources; this approach is referred to as *random grid search for weight learning* (RGS). It is important to ensure that the resources are used to evaluate distinct weight configurations, therefore at the time of sampling I ensure that every weight configuration chosen has a possibility of yielding a distinct solution. In Section 5.1.3, I show how two weight configurations that look distinct can yield the same solution at the time of MAP inference (and hence for the value of γ as well) and how I can identify and avoid wasting resources on such inherently identical weight configurations. Therefore, to ensure uniqueness of weight configurations explored, I keep track of a set $\mathbf{W}_{explored}$ which contains all the weight configurations explored so far and the configurations that are inherently the same as an explored weight configuration and sample new weight configuration from $\{\mathbf{W} - \mathbf{W}_{explored}\}$.

Continuous Random Search for Weight Learning

A primary drawback of RGS is the need to define a grid over the space which captures weights that will lead to a good model. While specifying a grid might seem straightforward, several unique properties of the weight space makes the process of specifying the right grid non-trivial (details in Section 5.1.3). Further, specifying grids can easily result in unexpected biases. For

Algorithm 4: Random grid search for weight learning.

Result: $\mathbf{w}^*$: weight configuration with best evaluation metric among samples

$\mathbf{W}$ = set of weight configurations from the user-defined weight grid;

t = maximum number of weight configurations to explore;

$\mathbf{W}_{explored}$ = weight configurations explored so far;

for $iter \in \{1, \ldots, t\}$ **do**

  $\mathbf{w}_{next} = Random(\mathbf{W} - \mathbf{W}_{explored})$;

  perform MAP inference to compute $\gamma(\mathbf{w}_{next})$;

  $\forall \dot{\mathbf{w}} \in \mathbf{W}$ such that $\gamma(\dot{\mathbf{w}}) = \gamma(\mathbf{w}_{next})$, add to $\mathbf{W}_{explored}$;

  if $\gamma(\mathbf{w}_{next}) > \gamma(\mathbf{w}^*)$ **then**

    $\mathbf{w}^* = \mathbf{w}_{next}$;

  end

end

instance, one may be tempted to simply define a grid of evenly spaced points in a unit hypercube. However, this leads to a sampling bias towards configurations with moderate ratios which might not be ideal (more details in Section 5.1.3).

Continuous random search for weight learning (CRS) is similar to RGS in that, rather than exploring the entire space, t weight configurations are chosen for evaluation and the highest performing configuration is returned. The difference is that CRS does not define a discrete grid of weights but samples continuously from the search space. Therefore, it is crucial for the search

space to be an accurate representation of the true weight space. In this approach (as mentioned in Algorithm 5), I sample t weight configurations to explore $\mathbf{W}_{explore}$ from a Dirichlet distribution as an approximation of the weight space and finally return the weight configuration with the best value obtained for the γ function. Here, the Dirichlet distribution represents the weight space $\mathbf{W}$. In Section 5.1.3, I discuss in detail on why this is an appropriate choice. For CRS the Dirichlet distribution is parametrized by a r-dimensional hyperparameter $\mathrm{A} \in \mathbb{R}^{+r}$, which can be tuned to obtain the best approximation of the space based on the application and prior knowledge. Note that a Dirichlet distribution can generate positive weights only. This can be restrictive for MLNs which support negative weights. In Section 5.1.4, I show how the sampling approach is extended to support negative weights for MLNs.

Algorithm 5: Continuous random search for weight learning.

Result: $\mathbf{w}^*$: weight configuration with best evaluation metric among samples

```
t = maximum number of weight configurations to explore;
W_explore ~ Dirichlet(A)^t, t distinct samples from a Dirichlet distribution;
for w_next ∈ W_explore do
    perform MAP inference to compute γ(w_next);
    if γ(w_next) > γ(w*) then
        w* = w_next;
    end
end
```

Hyperband for Weight Learning

So far, the search-based methods I have discussed iteratively select a set of t weight configurations to explore, $\mathbf{W}_{explore}$, from a set of weight configurations, $\mathbf{W}$ and then run inference until completion for each $\mathbf{w}_i \in \mathbf{W}_{explore}$ in order to calculate $\gamma(\mathbf{w}_i)$. Then the weight configuration $\mathbf{w}^* = \arg\max_{\mathbf{w}_i \in \mathbf{W}_{explore}} \gamma(\mathbf{w}_i)$ is chosen as the model weights. Ideally, in these methods, I want t to be as large as possible, as increasing the number of weight configurations explored (t) can potentially improve $\gamma(\mathbf{w}^*)$ obtained. However, it is generally infeasible to have a large t due to limited resources. In order to maximize my gain with the limited resources, I make use of a practical observation that the weight configurations which initially show a slow rate of improvement during inference will tend to converge to a poor $\gamma(\cdot)$ evaluation. For instance, let us assume two potential weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$ for a SRL model, if MAP inference for both the weight configurations takes $\hat{t}$ iterations to converge to the final solution and results in $\gamma(\mathbf{w}_1) > \gamma(\mathbf{w}_2)$, then it is highly likely that the $\gamma(\cdot)$ computed by interrupting the MAP inference at $\frac{\hat{t}}{\hat{s}}$ number of iterations, where $\hat{s} > 1$, will still result in $\gamma(\mathbf{w}_1) > \gamma(\mathbf{w}_2)$. Therefore, running MAP inference to convergence for all weight configurations could be wasteful. By early termination of unpromising weight configurations a larger number of configurations can be explored resulting in a better overall solution. This idea has been exploited in other areas of machine learning to tune hyperparameters and is referred to as *Hyperband* [Li et al., 2018b]. Here, I adapt this approach in the context of weight learning and refer to it as *Hyperband for weight learning* (HBWL).

HBWL allows for more exploration while still operating within a budget (generally

time or iterations) through adaptive resource allocation and early-stopping. To understand how the algorithm operates, I will first describe *SuccessiveHalving*, a critical subroutine of HBWL, and then describe how SuccessiveHalving is used in HBWL. SuccessiveHalving (as mentioned in Algorithm 6) requires two input parameters, namely t the total number of weight configurations I wish to explore and B the total number of iterations for MAP inference [2]. Initially t configurations $\mathbf{W}_{explore} = \{\mathbf{w}_1, \cdots, \mathbf{w}_t\}$ are sampled from the search space $\mathbf{W}$. Then, SuccessiveHalving proceeds in rounds. At the start of each round, a fraction of the budget ($b = \frac{B}{t}$) is allocated to the t weight configurations to perform MAP inference and compute $\gamma'(\cdot, b)$ where $\gamma'(\cdot, b)$ is the evaluation metric computed after b iterations of MAP inference. Finally, the weight configurations are ranked based on $\gamma'(\cdot, b)$, the bottom half of the weight configurations are removed, and $|\mathbf{W}_{explore}|$ is reduced to $\frac{t}{\eta}$ where $\eta > 1$ is the proportion of configurations to be removed (for classic SuccessiveHalving $\eta = 2$). This process is repeated for multiple rounds until only one weight configuration remains which is chosen as the $\mathbf{w}^*$.

The hyperparameters B and t of SuccessiveHalving can trade-off between having a large number of weight configurations with a small amount of resource allocated to each configuration (a.k.a. exploration), or a small number of weight configurations with a large amount of resource allocated to each weight configuration (a.k.a. exploitation). The best trade-off between exploration and exploitation is typically unknown. HBWL (as mentioned in Algorithm 7) extends Successive-Halving by trying several possible values for the $\frac{t}{B}$ ratio to choose the best explore exploit trade-off.

[2]Note that, for simplicity, in the algorithm I overload B to be number of iterations in MAP inference in the first round instead of maximum number of iterations. The maximum number of iterations in Algorithm 6 is $B\eta^{log_\eta(t)}$, where η is a parameter defined in HBWL which represents the proportion of weights removed every round in SuccessiveHalving.

The possible values for $\frac{t}{B}$ are constructed strategically from the two user provided parameters for HBWL, $\hat{R}$ the maximum amount of resource that can be allocated to a single weight configuration and $\eta \in (1, \infty)$ the proportion of weight configurations to be removed in each round of Successive-Halving. Each complete execution of SuccesiveHalving in HBWL is referred to as a bracket. Every bracket is parameterized by the values (t, B, s) that are constructed uniquely for each bracket using $\hat{R}$ and η, where s is the number of initial configurations being tested in that bracket. HBWL chooses a bracket size of $s = \lfloor \log_\eta(\hat{R}) \rfloor + 1]$ and decrements it every round until one. Finally, the weight configuration with the best value for function γ is returned.

Bayesian Optimization for Weight Learning

Next, I introduce BOWL (Bayesian Optimization for Weight Learning), which uses GPR to perform weight learning in the BO framework[3]. Previously discussed approaches make no assumptions about the search space and the evaluation function (γ). They randomly sample weight configurations from the search space and evaluate the function γ. This process can be made more efficient by assuming that the metric obtained by two weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$ are likely to be similar ($\gamma(\mathbf{w}_1) \approx \gamma(\mathbf{w}_2)$) if the distance between them is small. This implies that when searching the space I can make use of this information and either explore more diverse weight configurations or exploit and choose weight configurations closer to previously best performing configurations. This can be obtained by using BOWL. Next I explain BOWL and my choices for both the kernel function in GPR and the acquisition function in BO which are key in determining the performance

[3]Note that, in my method I loosely refer to this specific way of using GPR in BO framework as Bayesian optimization.

Algorithm 6: SuccessiveHalving

Result: $\mathbf{w}^*, \gamma^*$: weight configuration with the best evaluation metric and the metric value $\gamma(\cdot)$

$B =$ number of iterations in MAP inference for first round;

$\eta =$ the proportion of weight configurations to be removed in each round;

$\mathbf{W}_{explore} =$ weight configurations to be evaluated;

$t = |\mathbf{W}_{explore}|$;

$b = B$;

$iter = 0$;

while $t > 1$ **do**

  $iter$++;

  $\forall \mathbf{w} \in \mathbf{W}_{explore}$ perform MAP inference with max iterations set to b;

  if $\arg\max_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b) > \gamma^*$ **then**

    $\gamma^* = \max_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b)$;

    $\mathbf{w}^* = \arg\max_{\mathbf{w} \in \mathbf{W}_{explore}} \gamma'(\mathbf{w}, b)$;

  end

  $t = \lfloor \frac{t}{\eta} \rfloor$;

  $b = B\eta^{iter}$;

  $\mathbf{W}_{explore} = top_t(\mathbf{W}_{explore})$, ;

  // top_t ranks $\mathbf{W}_{explore}$ based on γ' and returns top t weights;

end

Algorithm 7: Hyperband for weight learning.

Result: $\mathbf{w}^*$: weight configuration with best evaluation metric

$\hat{R} =$ maximum number of resources to be allocated;

$\eta =$ the proportion of weight configurations to be removed in each round;

$s_{max} = \lfloor \log_{\eta}(\hat{R}) \rfloor$, maximum bracket size;

$\gamma^* = -\infty$, the best γ value obtained so far;

for $s \in \{s_{max}, s_{max} - 1, \cdots, 1\}$ **do**

  $t = \lfloor \frac{(s_{max}+1)\eta^s}{(s+1)} \rfloor$;

  $B = \hat{R}\eta^{-s}$;

  $\mathbf{W}_{explore} \sim Dirichlet(\mathrm{A})^t$, t distinct samples from a Dirichlet distribution;

  $\mathbf{w}_s, \gamma_s = SuccessiveHalving(B, W_{explore})$;

  if $\gamma_s > \gamma^*$ **then**

    $\gamma^* = \gamma_s$;

    $\mathbf{w}^* = \mathbf{w}_s$;

  end

end

of BOWL.

A high-level sketch for BOWL (as mentioned in Algorithm 8) is as follows: first, a weight configuration $\mathbf{w} \in \mathbf{W}$ is chosen using an acquisition function α (discussed in Section 10). Next, inference is performed using the current weight configuration $\mathbf{w}$, and $\gamma(\mathbf{w})$ is computed. Then GPR is updated with $\mathbf{w}$ and $\gamma(\mathbf{w})$. Finally, after t iterations, the weight configuration that resulted in highest value for γ is returned. As mentioned earlier, there are two primary components of BOWL that need to be defined: the kernel function used in GPR and the acquisition function α.

In order to use GPR and choose a kernel function, I must make an assumption about the function γ. Here, I assume that the function γ is smooth. This assumption is true if the problem is well-conditioned and the metric function ω being optimized is a smooth function, such as *mean square error* (MSE). For now, I make this assumption (justified further in Section 10), and choose the *squared exponential kernel* as the kernel for the GP:

$$k(\mathbf{w}_i, \mathbf{w}_j) = \tilde{\sigma} \cdot exp^{\{\frac{\Delta_{i,j}}{2\rho^2}\}} \tag{5.1}$$

where $\tilde{\sigma}$ is the amplitude, ρ is the *characteristic length-scale*, and $\Delta_{i,j}$ is the distance between the two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$. ρ and σ are the kernel hyperparameters. The scaling factor ρ affects the smoothness of the approximation (a large value implies more smooth) and the number of iterations required to explore the space. I choose ρ such that a reasonable exploration of the space is possible in t iterations. The value of $\tilde{\sigma}$ is chosen based on the range of the metric being learned.

The distance function Δ is crucial in determining the co-variance between two weight configurations. Ideally, if the distance between two weight configurations is zero then the output

of the function γ should be the same. And, as the distance between the two weight configurations increases, the correlation between the output of the γ function should go to zero. In Section 5.1.3 I introduce a new projection for the weights and show that distances measured in the projected space exhibit these properties.

Algorithm 8: Bayesian optimization for weight learning.

Result: $\mathbf{w}^*$: weight configuration with best evaluation metric among samples

$\mathbf{W} = Dirichlet(A)$;

$t =$ maximum number of weight configurations to explore;

for $iter \in \{1, \ldots, t\}$ **do**

  $\mathbf{w}_{next} = \arg\max_{\mathbf{w} \in \mathbf{W}} \alpha(\mathbf{w})$; α//an acquisition function chosen from Section 10;

  perform MAP inference to compute $\gamma(\mathbf{w}_{next})$;

  update GPR with $\gamma(\mathbf{w}_{next})$;

  if $\gamma(\mathbf{w}_{next}) > \gamma(\mathbf{w}^*)$ **then**

    $\mathbf{w}^* = \mathbf{w}_{next}$;

  end

end

Justification for Squared Exponential Kernel As mentioned earlier, the squared exponential kernel makes a strong smoothness assumption on the γ function. While this might seem restrictive, I argue that this is a reasonable assumption in the context of weight learning in SRL. To do this, I constrain ourselves to only those metrics ($\omega(\mathbf{y}, \mathbf{y}^*)$) that are smooth with respect to the random

variables (such as MSE). Note, my assumption is that the function γ is smooth and γ is parametrized with $\mathbf{w}$ and not the random variables $\mathbf{y}$. Hence, it is non-trivial to prove smoothness in γ. With the above constraint on the possible metrics, I know that if a small change in $\mathbf{w}$ leads to a small change in $\mathbf{y}$, then the function γ is also smooth. I formally define smoothness of the function γ as follows:

Definition 6. *Given two sets of weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *for a SRL model with* r *rules that generates* n *unobserved random variables* $\mathbf{y}$*, the function* γ *is considered to be smooth if* $\Delta_{1,2} < \epsilon$ *where* $\epsilon \to 0$*, then* $||\mathbf{y}_1 - \mathbf{y}_2||_2 < \nu$ *where* $\nu \to 0$*,* $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *are the random variables inferred using weights* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *respectively.*

This directly leads to the conditioning of the problem. If a problem is well-conditioned then my assumption about the smoothness of γ is precise. If the problem is ill-conditioned then this assumption fails to hold and the function learned in BOWL could be a poor approximation of γ. Further, in practice small changes in weights generally do not affect the γ function significantly which indicates smoothness. Even though I assume $\omega(\mathbf{y}, \mathbf{y}^*)$ is a smooth function such as the MSE to justify squared exponential kernel, in my empirical evaluation this is effective even on other non-smooth evaluation functions such as accuracy.

Acquisition Function Another crucial component of my algorithm to be defined is the acquisition function α. The function α determines the next weight configuration on which to evaluate the function γ, i.e., $\mathbf{w}_{next} = \arg\max_{\mathbf{w} \in \mathbf{W}} \alpha(\mathbf{w})$. Since my approach approximates the function γ with g, I would like to choose points that allow us to learn the approximation g while also maximizing the metric γ. To achieve this, I consider four well studied acquisition functions in the context of BO.

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

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

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

$$\alpha(\mathbf{W}) = \tilde{p}(\mathbf{w})$$

$$\tilde{p}(\mathbf{w}) \sim \mathcal{N}(\mu(\mathbf{w}), \sigma(\mathbf{w}))$$

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

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

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

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

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

function can be written as:

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

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

Efficiency of Search-Based Approaches

In practice, it is inefficient to use search-based approaches for high-dimensional problems. For instance, GPR has been shown to work best when the number of dimensions is less than 50 [Wang et al., 2016]. This makes weight learning an ideal use case for search-based approaches, because typically SRL models have just tens of rules and most often the number of rules does not exceed 50.

The success of all the weight learning approaches discussed so far relies on two important factors: 1) the weight configurations chosen are accurate representation of the true search space; and 2) the distances measured between weight configurations correlate to the solution obtained by using them. In the next section, I first show that the weight space in both PSL and MLN are redundant making the representation imprecise and weight configuration distances do not correlate to the solution obtained. Next, assuming positive weights for rules, I introduce a novel projection that address these challenges. Finally, I also provide an efficient strategy to approximately sample from the projected space ensuring the effectiveness of the search-based approaches introduced.

5.1.3 Efficient Space to Search for Weights

The weights of a SRL model consisting of r rules is represented via a vector in an r-dimensional space and is referred to as *original space* (OS). However this is an inefficient space to preform weight learning using a search-based approach. This is because there exists many weight configurations which yield the same solution when performing MAP inference in SRL models. The main reason for this is because weights in SRL models are relative and scale invariant at the time of MAP inference. I can show that any SRL model with weights in $\mathbb{R}$ can be re-scaled with a positive constant $\tilde{c}$ without any change to the solution obtained through the objective in Equation 2.4. This shows that weights in SRL models are scale invariant when performing MAP inference.

Theorem 2. *Consider any SRL model with r rules and weights $\mathbf{w} = \{w_i, \ldots, w_r\}, w_i \in \mathbb{R}^+$ ($w_i \in \mathbb{R}$ for MLN). For all weight configurations $\tilde{c} \cdot \mathbf{w}$ where $\tilde{c} > 0$, the solution obtained for $\mathbf{y}$ by performing MAP inference using weights $\mathbf{w}$ and $\tilde{c} \cdot \mathbf{w}$ are the same, i.e.,* $\arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{c} \cdot \mathbf{w})$.

Proof. The objective generated by using weights $\tilde{c} \cdot \mathbf{w}$ can be written as:

$$\begin{aligned}
\arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{c} \cdot \mathbf{w}) &= \arg\max_{\mathbf{y}} \hat{s} \cdot \sum_{i}^{\iota} \tilde{c} \cdot w_i \phi_i(\mathbf{x}, \mathbf{y}) \\
&= \arg\max_{\mathbf{y}} \tilde{c} \cdot \hat{s} \cdot \sum_{i}^{\iota} w_i \phi_i(\mathbf{x}, \mathbf{y}) \\
&= \arg\max_{\mathbf{y}} \tilde{c} \cdot \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w})
\end{aligned}$$

Since the MAP inference objective using both $\mathbf{E}_{psl}$ and $\mathbf{E}_{mln}$ are scale invariant, the re-scaling of the weight configuration $\mathbf{w}$ to $\tilde{c} \cdot \mathbf{w}$ leaves the solution of the inference unchanged. □

Since in my search-based approaches I optimize w.r.t. a user-defined evaluation metric, and this function depends only on the random variables obtained by MAP inference, I have that the user-defined evaluation metric function is also scale invariant.

Challenges in the Original Space

OS for weights has two fundamental challenges for search-based approaches: 1) OS is redundant and 2) the distance between weights in OS does not translate to true correlation of the solution obtained by using these weights. The redundancy of space is clear from Theorem 2 as the weights on any line intersecting origin in OS will have the same solution. This also means that the Euclidean distance $\delta_{i,j}$ between two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$ can be extremely large and still result in the exact same solution and vice versa. The example below clearly illustrates this phenomenon:

Example 1. *Consider a model with two rules* $\mathbf{w} = \{w_1, w_2\}$. *Let us assume three possible weight configurations for this problem:* $\mathbf{w}_1 = \{0.1, 0.1\}$, $\mathbf{w}_2 = \{1.0, 1.0\}$, *and* $\mathbf{w}_3 = \{0.1, 0.0001\}$. *Assuming that the number of groundings for both rules are the same, the weights of the rules in* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *indicate that both rules are equally important and lie on a line intersecting origin, while in* $\mathbf{w}_3$ *the first rule is* 1000 *times more important than the second rule and is not on the same line. This results in the function* γ *producing the same output for* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, and potentially a different value for* $\mathbf{w}_3$. *Based on this, the weight configuration* $\mathbf{w}_3$ *should be significantly different from the weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, while* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *should be similar. Unfortunately, the Euclidean distances measured between the weight configurations,* $\delta_{1,2} = 1.27$, $\delta_{1,3} = 0.09$, *and* $\delta_{2,3} = 1.34$,

do not behave in this manner. The distance $\delta_{1,2}$ is much larger than distance $\delta_{1,3}$. Therefore, some of the search-based approaches such as BOWL *would incorrectly infer that the function value of $\gamma(\mathbf{w}_1)$ is more correlated with $\gamma(\mathbf{w}_3)$ than $\gamma(\mathbf{w}_2)$. However, as argued above, I want the opposite behavior.*

In order to address these challenges, I introduce a new projection for the weights. For the remainder of this section I assume weights of the rules to be positive. While this does not restrict PSL which supports only positive weights, it does constrain MLNs which support negative weights. However, it has been shown that a negative weighted rule in MLNs can be replaced with a negated rule and positive weight with the same magnitude. Further in Section 5.1.4, I show how I extend and accommodate for positive and negative weights in MLNs when using the search-based approaches.

Scaled Space

In order to perform efficient and effective search, I define a new space for the weight configurations called *scaled space* (SS). SS is a projection of weights onto a relative space. I use the ratio of weights between the rules to define the relative importance of weights in the configuration. This projection eliminates redundancies and results in distances that correspond to the actual correlation between the weight configurations. Formally, I define SS as:

Definition 7. *Given a set of weights $\mathbf{w} = \{w_1, \ldots, w_r\} \in (0, \infty]^r$, SS $\mathcal{E}$ is a projection defined on $\mathbf{w}$ such that $\mathcal{E}(\mathbf{w}) \in \mathbb{R}^{(r-1)}$ is given by:*

$$\mathcal{E}(\mathbf{w}) = \{\forall_{i=2}^{r}(ln(w_i) - ln(w_1))\} \tag{5.2}$$

Given the definition of SS, I next define the distance between two weight configurations in SS as:

Definition 8. *The distance Δ between two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$ in SS is defined as:*

$$\Delta_{i,j} = ||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 \tag{5.3}$$

Given the definition of SS, I can now show that SS does not have the two challenges mentioned for OS. I first show that any weight configuration on a line intersecting the origin in OS will be represented by the same point in SS eliminating the redundancy that exist in OS. Next I show that in SS ($\mathcal{E}$), a distance of zero ($\Delta_{i,j} = 0$) between two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$, implies that the two weight configurations yield the same solution for the random variables $\mathbf{y}$ at the time of MAP inference hence proving accurate representation of distances between weight configurations.

Theorem 3. *Given two weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$, if $\mathbf{w}_1 = c \cdot \mathbf{w}_2$, i.e., $\mathbf{w}_1$ and $\mathbf{w}_2$ lie on a line intersecting the origin in OS then the resultant value in SS will be the same, i.e., $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$*

Proof. Given, $\mathbf{w}_1 = c \cdot \mathbf{w}_2$ implies:

$$w_{1,i} = c \cdot w_{2,i}; i \in 1, \ldots, r$$

$\mathcal{E}(\mathbf{w}_1)$ by definition is given by $\{\forall_{i=2}^{r}(ln(w_{1,i}) - ln(w_{1,1}))\}$. By replacing $w_{1,i}$ with $cw_{2,i}$ I get:

$$\mathcal{E}(\mathbf{w}_1) = \{\forall_{i=2}^{r}(ln(w_{2,i}) - ln(w_{2,1}))\} = \mathcal{E}(\mathbf{w}_2)$$

Therefore, if $\mathbf{w}_1 = c \cdot \mathbf{w}_2$ then $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$. □

Theorem 4. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, if* $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$ *(i.e.,* $\Delta_{1,2} = 0$*) then the solution obtained for* $\mathbf{y}$ *by optimizing Equation 2.4 with both the weight configurations are the same.*

Proof. Let $\mathbf{w}_1 = \{w_{1,1}, \ldots, w_{1,r}\}$, $\mathbf{w}_2 = \{w_{2,1}, \ldots, w_{2,r}\}$ and $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$. As the two weight configurations are the same in SS, the equality can be written as:

$$ln(\mathbf{w}_1) - ln(w_{1,1}) = ln(\mathbf{w}_2) - ln(w_{2,1})$$

$$\mathbf{w}_1 = \frac{w_{1,1}}{w_{2,1}}\mathbf{w}_2$$

Since $w_{1,1} \in (0, 1]$ and $w_{2,1} \in (0, 1]$ are constants, the resulting optimization problems are equivalent:

$$\begin{aligned}\arg\max_y \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_1) &= \arg\max_y \frac{w_{1,1}}{w_{2,1}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2) \\ &= \arg\max_y \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2)\end{aligned}$$

Therefore, if the distance between two weight configurations is 0 in SS, then the solutions of their corresponding SRL model by optimizing Equation 2.4 are the same. □

Theorem 3 shows that SS is not redundant and Theorem 4 proves the distances in SS are accurate. These theorems show the correctness of SS for weight learning using search-based approaches. Note that in the definition of SS I use the weight of the first rule to compute the projection. This choice is arbitrary and can be switched to any rule without affecting the space.

Example 1. *(Continued) Consider the earlier example. The weights and the distances of my running*

example in SS $\mathcal{E}$ using Equation 5.2 and 5.3 are: $\mathcal{E}(\mathbf{w}_1) = \{0\}$, $\mathcal{E}(\mathbf{w}_2) = \{0\}$, $\mathcal{E}(\mathbf{w}_3) = \{6.907\}$, $\Delta_{1,2} = 0$, $\Delta_{1,3} = 47.7$, *and* $\Delta_{2,3} = 47.7$.

A drawback of SS is that it does not support a weight of zero for any rule in the model. This means that all rules in the configuration must participate in the model. However, in practice, I mitigate this by using SS only to measure distances between weight configurations which is performed by adding a small positive value (e.g., $10^{-\varrho}$, where $\varrho \in Z^+$, $\varrho \gg 0$) to all weights. In order to generate weight configurations for the search, I sample from a hypersphere in OS which has similar properties as SS. I discuss this in detail in the Section 5.1.3.

The Effect of Varied Number of Groundings in the Scaled Space

My discussion on SS so far has made a very important simplifying assumption, that the number of groundings for each rule in the model is the same. However, the number of groundings produced by different rules are seldom the same and the number of groundings produced by a rule has an impact on the inference of the random variables. The weight associated with each rule is repeated for each ground instance of that rule. This leads to the weight of each rule having varied influence on the minimization of the energy function. For instance, if a model has two equally weighted rules, but one rule produces 10 times more groundings than the other, then that rule implicitly becomes 10 times more important in the model.

Next I show that while the number of groundings has an impact on the solution obtained by SRL models, it does not impact the correctness of SS. I can modify the weights to accommodate the number of groundings of the rules in the model. Consider a model with r rules and let

$\beta = \{\beta_1, \ldots, \beta_r\}$ be the number of groundings for each of the r rules. I define a grounding factor κ for each rule. For rule z, the grounding factor $\kappa_z = \frac{\beta_z}{\max(\beta)}$, where $\boldsymbol{\kappa} = \{\kappa_1, \ldots, \kappa_r\}$ is the vector of grounding factors. Therefore, the true weight associated with the z^{th} rule is $\kappa_z \cdot w_z$ and the grounding adjusted weight configuration can be represented as an element-wise dot product between $\boldsymbol{\kappa}$ and $\mathbf{w}$, i.e., $\tilde{\mathbf{w}} = \boldsymbol{\kappa} \cdot \mathbf{w}$. The distance between two weight configurations i and j in OS can be re-written as $||\tilde{\mathbf{w}}_i - \tilde{\mathbf{w}}_j||_2^2$. Similarly the distance in SS can be re-written as $||\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)||_2^2$. However, the scaling factor $\boldsymbol{\kappa}$ does not affect the distance in SS as $\boldsymbol{\kappa}$ is constant for both weight configurations and cancels when computing the distance leaving the distance in SS unchanged.

Theorem 5. *Given two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$, a set of grounding factors of $\boldsymbol{\kappa}$, and grounding adjusted weight configurations $\tilde{\mathbf{w}}_i = \boldsymbol{\kappa} \cdot \mathbf{w}_i$ and $\tilde{\mathbf{w}}_j = \boldsymbol{\kappa} \cdot \mathbf{w}_j$, the distance measured between both $(\mathbf{w}_i, \mathbf{w}_j)$ and $(\tilde{\mathbf{w}}_i, \tilde{\mathbf{w}}_j)$ in SS are equal, i.e. $||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 = ||\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)||_2^2$.*

Proof. To prove the above theorem I consider the difference between the weight configurations $\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)$:

$$\begin{aligned}
\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j) &= (ln(\boldsymbol{\kappa} \cdot \mathbf{w}_i) - ln(\kappa_1 \cdot w_{i,1})) - \\
&\quad (ln(\boldsymbol{\kappa} \cdot \mathbf{w}_j) - ln(\kappa_1 \cdot w_{j,1})) \\
&= (ln(\mathbf{w}_i) - ln(w_{i,1})) - \\
&\quad (ln(\mathbf{w}_j) - ln(w_{j,1})) \\
&= \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)
\end{aligned}$$

Since $\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j) = \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)$, the distances are also equal. □

Theorem 5 shows that the distance measured between two weight configurations in SS is robust while considering the size of their groundings.

Sampling Weight Configurations for Search

Most search-based approaches work by choosing different possible weight configurations to explore and search for the weight configuration with the best evaluation score. In order to do this effectively, I must ensure that the samples generated for exploration are representative of the space. While it might be ideal to directly sample uniformly from SS, this is challenging due to the one-to-many mapping between SS and OS and, as mentioned earlier, points in SS cannot represent rules with zero weights. One straightforward approach to handle this is to uniformly sample weight configurations from the positive quadrant of a unit hypercube in OS and and project the points on to SS to measure distances. The approach can be summarized as:

$$\mathbf{w} \sim \textit{Unif}([0,1])^r$$

where $Unif$ generates uniform random numbers between $[0,1]^r$. Since, the influence of weights in SRL models are scale invariant for MAP inference, I can ensure all possible weight configurations that can be represented in the $\mathbb{R}^{+r}$ can be represented in $[0,1]^r$. However the main problem with this approach is that the resulting configurations will have a low spread in SS as OS has a lot of redundancies and the projection will place several weight configurations around the same region. This is not desirable as I would prefer to have a uniform/large spread of possible weight configurations in SS.

As mentioned earlier, MAP inference in SRL models are scale invariant and hence weights along a line passing through 0 in OS are equivalent, i.e., the direction of a vector starting at the origin in OS is sufficient to represent all weight configurations in SRL models. This implies that the weight configurations obtained by sampling from the surface of an r-dimensional unit hypersphere in the positive quadrant represents all possible weight configurations in OS. Therefore, uniformly sampling from the surface of this hypersphere (I only refer to the positive quadrant of the unit hypersphere) is a close approximation of SS. It is trivial to show that Theorem 3 and 4 apply to this space (full proofs are in Appendix A.2.1). However, Theorem 5 does not hold true for this space. This is because the grounding factor generally does not preserve the orientation of the vector, and as a result will modify the distances in this space (full proof shown in Appendix A.2.1). Therefore, I treat the hypersphere as an approximation of SS and sample weight configurations from the hypersphere but compute distances in SS. Uniform samples of weight configurations from the surface of the hypersphere can be obtained by first sampling points from a standard multivariate-normal distribution and projecting the values to the hypersphere [Muller, 1959, Marsaglia, 1972] (since I want samples only from the positive quadrant I project all samples on to this quadrant by taking the absolute value):

$$\mathbf{s} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{w} = |\frac{\mathbf{s}}{||\mathbf{s}||}|$$

$\mathbf{0}$ is a r-dimensional zero vector and $\mathbf{I}$ is an r-dimensional identity matrix.

While uniform sampling from the surface of a hypersphere ensures that every orientation of the weight vector is equiprobable, in practice this might not always be desirable. The primary

reason for this is that for any weight configuration sampled from the hypersphere, if I choose two weights $w_i < w_j$ the $P(\frac{w_i}{w_j} < 0.1) \approx 0.11$ (assuming $r = 2$). This implies that the ratio between weights will typically be close to one which is not ideal as I would expect the evaluation metric to show large variance with larger ratios. In order to circumvent this, I propose another sampling strategy which gives us full control over the distribution of the weight configurations. I sample from an r-dimensional Dirichlet distribution which generates samples from the probability simplex. It is easy to see a one-to-one correspondence of the probability simplex and the surface of a sphere. The hyperparameter of the Dirichlet distribution can be modified to generate samples skewed towards the center (all equal weights) or the poles (extreme ratios) or anywhere in between. The sample generation process is as follows:

$$\mathbf{w} \sim Dirichlet(\mathrm{A}) \tag{5.4}$$

where $\mathrm{A} \in R^{+r}$ is the hyperparameter that defines the Dirichlet distribution. The visualization of different densities obtained using different A in three dimension is shown in Appendix A.2.1 along with its impact on the empirical evaluation in Section 5.1.5.

5.1.4 Accommodating Negative Weights in Markov Logic Networks

In this section I discuss how the sampling strategy discussed in Section 5.1.3 is modified for MLNs to accommodate negative weights when using CRS, HBWL, and BOWL. Since the Dirichlet distribution samples from the probability simplex, the weights sampled are strictly non-negative. To introduce the possibility of negative weights, I first sample weight configurations

from the Dirichlet distribution and then randomly select an orthant in the r-dimensional Euclidian space. This random selection of an orthant has the same effect as independently flipping the sign of each weight in the sampled configuration so every orthant is equiprobable. I further apply the positive scale invariance property for weight configurations to ensure the uniqueness of samples. While no sampling is needed for RGS in MLNs, I ensure uniqueness of explored configurations by projecting the values on to a unit hypersphere instead of SS. In order to use BOWL in MLNs, instead of computing distance in SS, I compute the Euclidean distances of the weight configurations by projecting the weights onto the hypersphere [4].

5.1.5 Empirical Evaluation

In this section, I evaluate the search-based approaches for weight learning on various realworld datasets. I investigate four research questions through my experiments:

[Q1] How do search-based approaches perform on realworld datasets compared to the existing methods?

[Q2] Which search-based approach performs better weight learning in SRL?

[Q3] Are search-based approaches scalable?

[Q4] Are search-based approaches robust?

In order to answer these questions I selected five realworld datasets from different domains

[4]Note this may lead to over generalization of the function as two points might be very close to each other but have significantly different outcome on the γ function (as mentioned in Example 1).

for which SRL models have promising results [Bach et al., 2017, Kouki et al., 2015][5]. Details of these datasets are as follows:

Jester: contains 2,000 users and 100 jokes [Goldberg et al., 2001]. The task is to predict user's preference to jokes.

LastFM: contains 1,892 users and 17,632 artists. The task is to recommend artists to users by predicting the ratings for user-artist pairs.

Citeseer: contains 2,708 scientific documents, seven categories, and 5,429 directed citations. The task is to assign a category to each document.

Cora: is similar to Citeseer dataset, but contains 3,312 documents, six categories and 4,591 directed citations.

Epinions: contains 2,000 users and 8,675 directed links which are positive and negative trust links between users. The task is to predict the trust relation.

I evaluate the three search-based methods (RGS, CRS, and BOWL) for both MLNs and PSL, and HBWL for PSL only. While the implementation of RGS, CRS, and BOWL are efficient to run for both MLNs and PSL (more details in Section 5.1.5), HBWL with MLNs on my datasets is not as efficient and each experiment was estimated to take about a month. While the search-based approaches have been fully integrated into PSL codebase, for MLNs, I implement search-based methods as wrappers, i.e., I use an external script to generate weights and use Tuffy as a black-box evaluator. Because of this, in HBWL, the SuccessiveHalving step requires us to reground the same model multiple times which substantially increases time required to run. I use MLE, MPLE, and

[5]Models, code, and data: https://github.com/linqs/srinivasan-mlj20

LME as baseline methods for PSL and use DN implemented in Tuffy as my baseline for MLN. As MLNs are discrete, I perform evaluations only on the three discrete datasets, Citeseer, Cora, and Epinions. For each dataset, depending on the problem, I leverage the metric that has been used for the problem to measure its performance. Hence, I report *MSE* and *AUROC* for the Jester and LastFM datasets, *categorical accuracy (CA)* and *F1* for the Cora and the Citeseer datasets, and *AUROC* and *F1* for the Epinions dataset. Further, all my experiments were run on a machine with 16 cores and 64GB of memory.

Performance analysis

To address [Q1] and [Q2], I compare the performance of all search-based approaches RGS, CRS, HBWL, and BOWL with MLE, MPLE, and LME in PSL and DN in MLN on several metrics. For the datasets, I use the 8 folds generated by Bach et al. (2017) for Citeseer, Cora, Epinions and Jester and 5 folds generated by Kouki et al. (2015) for LastFM and perform cross validation. I perform a paired t-test (p-value ≤ 0.05) across methods to measure statistical significance. In RGS for PSL, I specify the set $V = \{0.001, 0.01, 0.1, 1, 10\}$, while for MLNs I use $V \cup -V$. For CRS, HBWL, and BOWL, I specify each dimension of A to be 0.05 for PSL (more experiments with different values can be found in Section 5.1.5) and 0.1 for MLNs. In HBWL for PSL, $\hat{R}$ represents the number of iterations of ADMM inference and is set to 25000 and η is set to 4. For RGS, CRS, HBWL, and BOWL, the maximum number of weight configurations to explore in order to approximate the user-defined evaluation metric function is set to $t = 50$. Although in my experiments the best metric value is usually obtained at $t < 25$ (especially for BOWL,

this is likely because the function I intend to learn has several flat regions). I also use a stopping criterion for BOWL which terminates the exploration if the standard deviation at all sampled weight configurations is less than 0.5. MLE, MPLE, LME, and DN are allowed to run for 100 iterations or until convergence whichever is smaller. For BOWL, I use UCB as the acquisition function with $\psi = 1$ to favor modest exploration. However in Section 5.1.5 I show that similar performance can be obtained with PSL by using the other acquisition functions discussed in Section 10. Other hyperparameters that I use for BOWL are: $\tilde{\sigma} = 0.5, \rho = 1$, and the mean function is a constant 0.5. I set the value of ρ to one after exploring different values in $[10^5, 10^{-5}]$, and I set $\tilde{\sigma}$ to 0.5 as my metrics are mostly in the range $[0, 1]$.

Table 5.1 and 5.2 show the comparison between search-based approaches and other methods across the different datasets for both PSL and MLN respectively. In each row of the table, the best performing method and those that are not significantly different from the best performing method are shown in bold. I first analyze the performance results of PSL in Table 5.1. Here, the search-based approaches are the best performing method across all the datasets and metrics. Specifically, BOWL is the best or not significantly different from the best performing method on all datasets and metrics (except LastFM MSE). For the Epinions and Cora datasets, there is no statistically significant difference among all approaches on both metrics. However, on the Citeseer dataset for CA all search-based approaches perform better than other approaches. An interesting observation here is that on Citeseer dataset the weights found by MLE and LME perform as well as search-based approaches on F1 but not on CA. This further strengthens my motivation to directly optimize for the evaluation metric rather than the likelihood. In LastFM, BOWL is significantly

Method (Metric)	MLE	MPLE	LME	RGS	CRS	HBWL	BOWL
Jester (MSE)	0.053	0.068	0.063	0.053	0.053	**0.052**	**0.053**
Jester (AUROC)	0.730	0.700	0.702	**0.762**	0.744	**0.756**	**0.761**
LastFM (MSE)	0.061	**0.060**	0.061	**0.061**	0.062	0.064	0.063
LastFM (AUROC)	0.548	0.552	0.549	0.574	0.572	0.556	**0.587**
Citeeseer (CA)	0.835	0.837	0.839	**0.844**	**0.844**	**0.845**	**0.844**
Citeeseer (F1)	**0.283**	0.293	**0.303**	**0.321**	**0.322**	**0.321**	**0.319**
Cora (CA)	**0.881**	**0.888**	**0.889**	**0.888**	**0.887**	**0.888**	**0.889**
Cora (F1)	**0.404**	**0.439**	**0.442**	**0.438**	**0.438**	**0.438**	**0.443**
Epinions (AUROC)	**0.811**	**0.792**	**0.807**	**0.814**	**0.797**	**0.810**	**0.780**
Epinions (F1)	**0.712**	**0.711**	**0.712**	**0.711**	**0.712**	**0.711**	**0.713**

Table 5.1: Performance of PSL weight learning methods across datasets; the best scoring methods (with $p < 0.05$) are shown in bold. Note: the metric values are rounded to three point precision making some numbers the same, but the significance test was performed on values with six point precision.

better than all other approaches on AUROC, however, RGS outperforms BOWL in the MSE metric on LastFM. One reason for likelihood-based approaches to perform better in LastFM MSE could be because the rating values mentioned in LastFM was constructed by fitting a negative binomial distribution [Kouki et al., 2015]. This could potentially imply that maximizing the likelihood in PSL could lead to minimizing the MSE. Finally on the Jester dataset, HBWL and BOWL perform the best on both metrics. Overall, as mentioned earlier, search-based approaches perform better than other approaches and in general even the worst performing search-based method is better than likelihood-based approaches (like in Jester and LastFM AUROC). These experiments clearly show that search-based approaches are better suited for weight learning in PSL. Further, of the search-based weight learning approaches BOWL performs the best followed by HBWL. This is because BOWL performs smart exploration of space to approximate the evaluation function and HBWL evaluates more points by smart resource allocation to get better approximation compared to CRS and RGS.

Next, in Table 5.2 I analyze the performance of the MLN weight learning methods on all datasets and metrics. Here, search-based methods achieved the best performance for every dataset and metric. Except for F1 score in the Cora dataset, where DN performs as well as search-based approaches, DN performs worse than search-based approaches on all datasets and metrics. This observation further supports the use of search-based weight learning methods across SRL frameworks. Similar to the PSL experiments, overall BOWL performs the best followed by HBWL. BOWL is either the best method and or not significantly different from the best method on all but the Epinions dataset when evaluating the AUROC metric. The reason for poor performance of

Method (Metric)	DN	RGS	CRS	BOWL
Citeeseer (CA)	0.829	0.828	0.829	**0.833**
Citeeseer (F1)	0.267	0.251	0.253	**0.281**
Cora (CA)	0.867	0.865	0.866	**0.871**
Cora (F1)	**0.339**	**0.333**	**0.342**	**0.355**
Epinions (AUROC)	0.614	**0.689**	**0.699**	0.659
Epinions (F1)	**0.705**	**0.708**	**0.711**	**0.710**

Table 5.2: Performance of MLN weight learning methods across datasets; the best scoring methods (with $p < 0.05$) are shown in bold.

BOWL in Epinions with AUROC could be because Epinions has the maximum number of rules (20 rules) compared to all other datasets. Further, since distances for BOWL in MLN are computed in OS, the function approximated for AUROC might need more fine tuning which might not be possible in OS leading to poor exploration. On the same dataset for F1 BOWL performs as well as CRS which is the best. This indicates that the approximation of the evaluation function through BOWL is dependent on the function being approximated as well.

Scalability

In this section, I compare the runtimes of MLE, MPLE, LME, RGS, CRS, HBWL, and BOWL in PSL to measure the scalability of search-based approaches and address [Q3]. I do not compare runtimes of my experiments with Tuffy. While DN is fully integrated in Tuffy, my search-based approaches were implemented as a wrapper to Tuffy. Hence, the runtime numbers

of different approaches will not be a fair comparison[6]. The number of parameters to learn in PSL and MLN is equal to the number of rules in the model and the data size translates to the number of groundings generated by the model. In Figure 5.2a, I show the number of groundings generated by each of the datasets.I also show the number of rules in each model. The Jester dataset produces the largest number of groundings ($\sim 1M$) using seven rules and the Epinions dataset produces the least number of groundings ($\sim 14K$) using the largest model (20 rules).

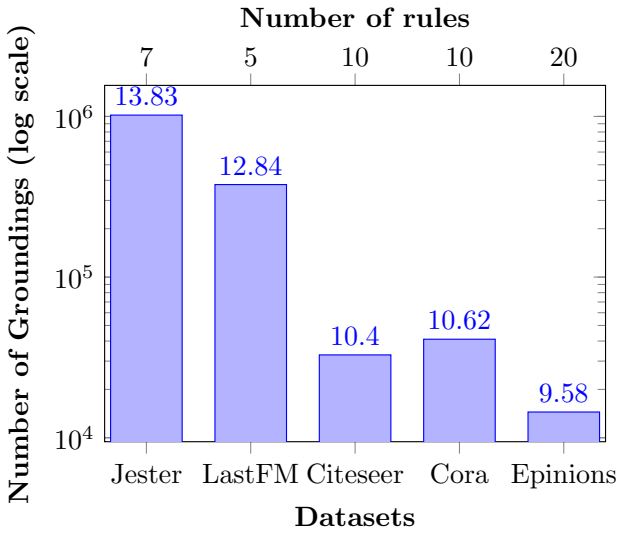
In Figure 5.2b, I show the average runtimes measured accross all folds for all approaches on all datasets. Runtimes for some approaches depend on number of groundings while some others depend on number of rules. The runtime for MPLE primarily depends on number of groundings and as the number of groundings increases the runtime also increases by a factor of ~ 45 from Epinions to Jester dataset. The time taken to run LME depends not only on the number of groundings, but also the complexity of finding the margin. Therefore, the runtime of LME on the LastFM dataset is higher than the Jester dataset. MLE, RGS, CRS, HBWL, and BOWL depend on number of groundings through the time taken to perform inference on larger models. Since inference in PSL is efficient due to its convex objective, these approaches can scale better with the number of groundings compared to the other approaches. Further, inference time in PSL depends on both the number of groundings (which affects per iteration cost in solving) and ease of solving the optimization (which affects the number of iterations required to converge). This can be observed when I compare runtimes of the above mentioned five methods on the Epinions dataset and Cora dataset. The

[6]Even though I reground the model every iteration for search-based methods in Tuffy, the search-based approaches were at least two times faster than DN in Citeseer and Cora datasets and due to early-stopping, BOWL in MLN was at least 5 times than DN.

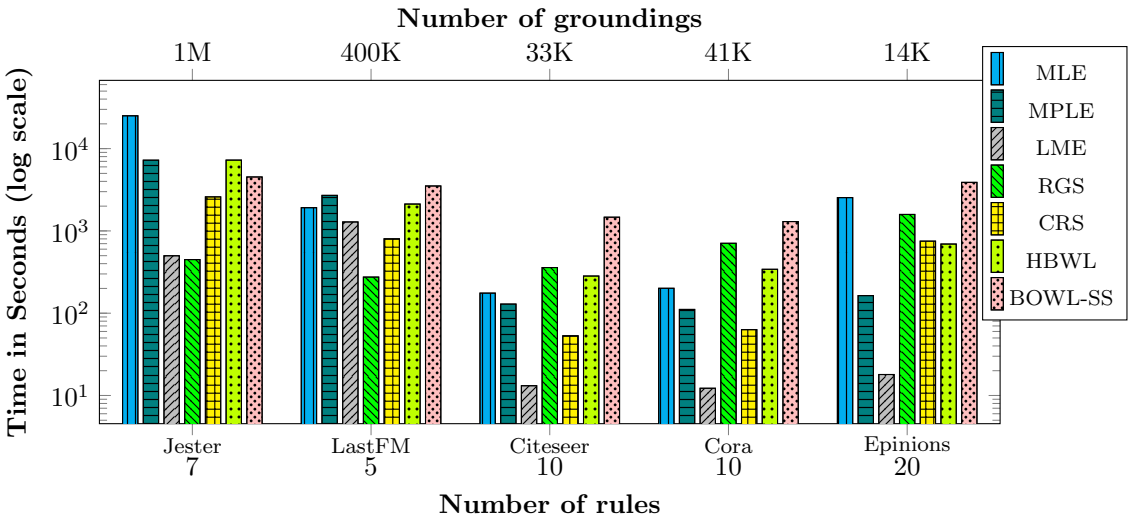
time taken to run Epinions is two times greater for MLE even though the number of groundings of Epinions is three times smaller. Overall MLE has the largest increase in runtime ($\sim$150 fold increase) from Cora to Jester dataset. The runtimes of search-based approaches also depend on the number of evaluations (or resources allocated) they are allowed. For a good approximation of the evaluation function the number of evaluation points required increases exponentially with number of rules in the model. Since I fix the maximum number of iterations to 50 for all models, it does not affect my approaches. Specifically, the runtimes of BOWL across datasets does not vary as much as other approaches. BOWL has the smallest increase in runtime ($\sim$3 fold) between the Cora and Jester datasets across all methods. This is because BOWL has a constant overhead for performing updates in the GP and retrieving the best next set of weight configurations. This also ensures bad weights that converge poorly are chosen less often (as they are likely to yield poor evaluation metric value), hence making it efficient and scalable with number of groundings. Overall search-based approaches can scale with a large number of groundings and produce the best results on the evaluation metric function.

Robustness

To address [Q4], I ran three sets of experiments: the first experiment is to check how robust search-based methods are w.r.t. different initialization, the second is to evaluate the robustness of CRS, HBWL, and BOWL to the hyperparameter A from the Dirichlet distribution, and the third experiment is to test the effects of choosing an acquisition function on the performance of BOWL.



(a) Groundings generated by different datasets.



(b) Time to learn vs. # of rules and groundings in datasets.

Figure 5.2: Analyzing the scalability of different approaches on the number of rules and groundings. With number of iterations fixed, search-based approaches scale better with both the number of rules and groundings.

Varied Initialization For the first experiment, I perform weight learning with all four search-based approaches using 30 random initializations and report the mean and standard deviation (std) of a metric per dataset in Table 5.3. Note that for this experiment I use UCB as the acquisition function in BOWL. Further, I use only one fold (of the eight folds) per dataset as I intend to measure the variance introduced by different initialization. In Table 5.3, the standard deviation is very small for Jester dataset and on other datasets except LastFM the standard deviation introduced by different folds is much higher than initializations indicating robustness to initialization. Finally, on the LastFM dataset, on all approaches except HBWL, a standard deviation larger than the standard deviation across folds obtained in Table 5.1. This is likely because HBWL explores more weight configurations using smart resource allocation. I can conclude that while the search-based approaches are reasonably robust to initialization, this can depend on the dataset and the number of weight configurations they are allowed to try.

Datasets	Varied initializations			
	RGS	CRS	HBWL	BOWL
Jester (MSE)	0.053 (0.001)	0.053 (0.001)	0.052 (0.001)	0.053 (0.001)
LastFM (MSE)	0.067 (0.008)	0.067 (0.006)	0.065 (0.002)	0.067 (0.006)
Citeseer (F1)	0.319 (0.007)	0.321 (0.007)	0.322 (0.008)	0.320 (0.007)
Cora (F1)	0.412 (0.006)	0.409 (0.005)	0.411 (0.005)	0.411 (0.005)
Epinions (F1)	0.714 (0.002)	0.713 (0.002)	0.714 (0.002)	0.716 (0.001)

Table 5.3: The table shows the mean (std) of the metrics obtained by running search-based approaches with varied initialization. The performance of BOWL is least affected by both initialization.

Impact of Hyperparameter A Here I evaluate the robustness of CRS, HBWL, and BOWL to the hyperparameter A in the Dirichlet distribution when sampling for the weight configurations. I chose four different values for $\mathrm{A} = \{10, 1, 0.1, 0.01\}$ and evaluated on one discrete (Citeseer) dataset and one continuous (Jester) dataset. For Citeseer, I use CA as the evalaution metric and I use MSE for the Jester dataset. I evaluate the three approaches, CRS, HBWL, and BOWL, that are impacted by A in PSL. Table 5.4 shows the metrics obtained for different values of A on both datasets and all methods. Here, the effect of A on the Citeseer datasest is minimal in all methods. This is likely because the CA function w.r.t. weights is reasonably flat and small changes in weights have minimal impact. Therefore as long as there is at least one sampled point in a region, it is sufficient to get an optimal value and thus all approaches seem robust. Next, when I consider the Jester dataset, the parameter A has a large impact on CRS followed by HBWL. For a value of $\mathrm{A} = 0.1$ CRS and HBWL perform the best and as it is increased to 10, both approaches produce the worst MSE value. This is because the space generated by the Dirichlet distribution using these parameter values is not representative of the true weight space for these two approaches. The impact of A is smaller on HBWL than CRS as HBWL explores more weight configurations via smart exploration. Finally, BOWL is robust to the parameter A as it uses GP to choose the next point.

Impact of Acquisition function in BOWL My third experiment measures the robustness of BOWL to different acquisition functions. In Table 5.5 I compare the performance of BOWL using four different acquisition functions (UCB, TS, PI, and EI) for all five datasets in PSL (one metric per dataset). On all datasets and metrics, BOWL is relatively robust to acquisition function and

Datasets	Methods	A = 10	A = 1	A = 0.1	A = 0.01
Citeseer (CA)	CRS	**0.844**	**0.844**	**0.843**	**0.844**
	HBWL	**0.843**	**0.844**	**0.843**	**0.843**
	BOWL	**0.844**	**0.844**	**0.845**	**0.843**
Jester (MSE)	CRS	0.064	0.054	**0.053**	0.056
	HBWL	0.061	0.053	**0.052**	0.054
	BOWL	**0.053**	**0.053**	**0.053**	**0.053**

Table 5.4: Performance of different search-based approaches by varying the A parameter in the Dirichlet distribution. The best metric value in every row is shown in bold.

performs similarly for all.

Finally, based on these experiments it can be observed that BOWL is the most robust to initialization, hyperparameter, and acquisition function and produces the best evaluation metric.

5.2 Online Inference

In many practical machine learning settings it is common for both the data and structure of the model to change incrementally over time. Incoming data may provide additional evidence or add to a set of predictive targets a system has to infer, and new information or a shifting context may alter a model's defining parameters or underlying structure. For instance, a new product review may change recommendations made for the reviewer and users who bought related products, or a recent event could impose constraints on a product's availability which were previously not considered

Datasets	Different acquisition functions			
	UCB	TS	PI	EI
Jester (MSE)	0.053	0.055	0.053	0.053
LastFM (MSE)	0.065	0.065	0.066	0.067
Citeseer (F1)	0.324	0.323	0.323	0.323
Cora (F1)	0.440	0.437	0.442	0.439
Epinions (F1)	0.711	0.712	0.713	0.713

Table 5.5: The table shows the effect of metrics obtained by using different acquisition function with BOWL. The performance of BOWL is unaffected by both acquisition function.

by a predictor. These are two of many examples of problems that are both online and require joint predictions.

Online inference is especially challenging for structured prediction settings Baki et al. [2007]. Structured prediction algorithms utilize the underlying relational properties of the data and problem domain to improve predictive performance and typically require collective (i.e., joint) inference over a probabilistic model. However, updates to the evidence and dependency structure of these methods can have cascading effects on the predictions that are expensive to perform. Updating inference is a long-standing problem in the machine learning community, and there is a vast body of literature on the topic for graphical models subject to structural updates Buntine [1991], Friedman and Goldszmidt [1997], Li et al. [2006], Acar et al. [2009], Sümer et al. [2011] and for dynamic models Murphy [2002], Nodelman et al. [2002].

In this section, I develop theory and methodologies for updating inference in an undirected graphical model with continuous valued random variables. My work in this section is most closely related to Pujara et al. (2015a), however my definition of online collective inference is much more general, supporting the introduction of new random variables and changes in the graphical model structure. Further, I analyze a different aspect of online collective inference by developing theory for performing exact inference on evolving models.

I define the problem of online collective inference using the framework of *templated graphical models* (TGMs) Koller and Friedman [2009]. In this framework, graphical models are specified by functions of relations and attributes of entities in a dataset, typically expressed as weighted logical rules. Then, *online collective inference* is the task of performing inference on a series of TGMs related by sequences of updates to the dataset and dependency structure.

My key contributions to this component of the model adaptability pillar include: 1) I define the problem of online collective inference using the framework of templated graphical models, 2) I propose principled approximations to updating an existing model, 3) I analyze the stability of MAP states of models subject to sequences of model updates, 4) I bound the loss incurred by performing approximate model updates, 5) I implement an online collective inference system using PSL, and 6) through experiments over three tasks, *online recommendation*, *online demand forecasting*, and *online model selection*, I show that my approximate online approach consistently provides a up to a 5 time speedup over an offline variant while, surprisingly, maintaining the quality of the predictions.

The work covered in this section was presented at the 2021 International Conference on

Machine Learning (ICML) Dickens et al. [2021].

5.2.1 Related Work

Online learning describes a class of machine learning methods where data arrives sequentially and the goal is to obtain the best predictor for future data using the most up-to-date information Shalev-Shwartz [2012]. Online machine learning in the independent and identically distributed (IID) setting has been studied extensively Mairal et al. [2010], Kushner and Yin [2003], Williams and Zipser [1989] and major advances have been made in creating efficient and scalable algorithms. Many online IID models can make updates to their predictor using the loss incurred from a single prediction Bottou [2010] or by updating a set of summary statistics Cesa-Bianchi and Lugosi [2006].

Updating graphical models is a long-standing problem in the machine learning community. There is substantial research in the area of updating Bayesian networks Buntine [1991], Friedman and Goldszmidt [1997], Li et al. [2006], both with evolving structure and updating parameters. Likewise, there has been much work in the area of dynamic and sequential modeling, such as dynamic and continuous time Bayesian Networks Murphy [2002], Nodelman et al. [2002] and hierarchical hidden Markov models Fine et al. [1998]. Adaptive inference is another related area that aims to make efficient updates to the inference result of a general graphical model defined over discrete valued variables Sümer et al. [2011].

The task of updating the MAP state of a TGM conditioned on evolving evidence was first considered in Pujara et al. (2015a). In this setting the graphical model has a fixed dependency structure, i.e., no variables or potentials are added or deleted, and the MAP state is found using a

technique which constrains the number of variables that may be updated to a predefined budget. The authors bound the *inference regret* induced by performing this type of approximate inference on an updated model. The definition of inference regret is defined as the normalized L_1 distance between a setting of the unobserved variables and the MAP state of a fixed model. My work differs significantly in that the templated graphical model is not fixed and inference is not constrained by a budget. My work also introduces a much more general notion of stability. The notion of collective stability London et al. [2013a, 2014] is related to the stability discussion in Section 5.2.4. Collective stability measures the change in the output of a structured predictor subject to updates to a set of evidence. However, again, it does not consider updates which add or delete variables or potentials defining the graphical model.

5.2.2 Online Collective Inference

A key challenge of online collective inference is expressing how two models are related and modified. To address this I leverage a general framework for defining probability distributions, referred to as *templated graphical models* (TGM) Koller and Friedman [2009]. Then, I formally define online collective inference.

Templated Graphical Models

A TGM encodes dependencies between relations and attributes of entities in a domain using functions called *template factors* with arguments referred to as *template variables*. Template

factors are commonly expressed as weighted logical rules, for example:

$$w : \textsc{Likes}(\mathtt{P}_1, \mathtt{P}_2) \rightarrow \textsc{Knows}(\mathtt{P}_1, \mathtt{P}_2) \tag{5.5}$$

This template factor represents the idea that entities who like each other will often know each other.

Every template variable is associated with a range, for instance, the range of LIKES is {LIKES(Alice, Bob), LIKES(Bob, Charlie)} and the range of KNOWS is {KNOWS(Alice, Bob), KNOWS(Bob, Charlie)}. These ranges are subsets of a provided dataset and define the random variables in the domain. Template factors are instantiated by realizing combinations of template variable instantiations. With the above templates and data, the resulting set of instantiated template factors is:

$$w : \textsc{Likes}(Alice, Bob) \rightarrow \textsc{Knows}(Alice, Bob)$$

$$w : \textsc{Likes}(Bob, Charlie) \rightarrow \textsc{Knows}(Bob, Charlie)$$

Given the instantiated set of template factors, a TGM defines a joint probability distribution over the instantiated template variables. More formally,

Definition 9 (Template Variables)**.** *A template variable V is a function of entity variables $V(e_1, \cdots, e_k)$, with a range denoted by $Val(V)$.*

Template variables are related by template factors, which, when instantiated, are referred to as *potentials*.

Definition 10 (Template Factors, Potentials)**.** *A template factor τ defines a mapping from a subset of a cartesian product of template variable ranges, $\Gamma \subseteq Val(V_1) \times \cdots \times Val(V_n)$, to $\mathbb{R}$. Given a*

tuple of random variables $R \in \Gamma$, I use $\phi(R)$ to denote an instantiated template factor, referred to as a potential.

An example potential is denoted as:

$$\phi(\text{LIKES}(Alice, Bob), \text{KNOWS}(Alice, Bob))$$

and is an arbitrary function that maps to $\mathbb{R}$. For the template factor defined earlier, the potential for the weighted rule can be logical (mapping to 0 or 1) or represent the distance to satisfaction. Generating a set of potentials by realizing every tuple of random variables in the union of the domains of the template factors is referred to as *grounding* (Section 5.2.3). A TGM is defined as a collection of one or more template factors, a set of potentials, and the set of random variables present in the dataset.

Definition 11 (Templated Graphical Model)**.** *Given a set of template factors T with a corresponding set of random variables Z and potentials $\Phi = \{\phi_1, \cdots, \phi_m\}$ a templated graphical model (TGM) defines the joint probability distribution* [7]

$$P(Z) = \frac{1}{\mathcal{Z}} \prod_{i=1}^{m} \phi_i(Z) \tag{5.6}$$

where $\mathcal{Z} = \int_z \prod_{i=1}^{m} \phi_i(Z = z)$ normalizes $P(Z)$. A TGM is denoted by the tuple $\mathbb{T} = (T, \Phi, P(Z), Z)$.

[8]

[7] Abuse of notation: $\phi_i(\cdot)$ is written as a function of Z when it actually is defined as a function of a tuple of random variables from Z.

[8] This definition assumes continuous valued potential functions. Mass functions are defined by replacing the integral in the definition by summation.

Any distribution can be expressed in the form (5.6) for some set of potentials Φ. Throughout this section I assume the random variables of a TGM, Z, are partitioned into observed random variables X and unobserved random variables Y. This partition defines a conditional distribution:

$$P(Y|X) = \frac{1}{\mathcal{Z}(X)} \prod_{i=1}^{m} \phi_i(Y, X) \tag{5.7}$$

where $\mathcal{Z}(X) = \int_y \prod_{i=1}^{m} \phi_i(Y = y, X)$. An equivalent way to express a TGM with a random variable partition and conditional distribution is $\mathbb{T} = (T, \Phi, P(Y|X), Y, X)$.

Online Collective Inference

TGMs provide a convenient means for systematically defining modifications to a graphical model, which are broken into fundamental steps called *model updates*.

Definition 12 (Model Update)**.** *Given the TGM $\mathbb{T} = (T, \Phi, P(Y|X), Y, X)$. I define a model update as one of the following:*

1. *Update the value of an observed variable $x_i \in X$*
2. *Add or delete a random variable $x_i \in X$ or $y_i \in Y$*
3. *Add or delete a template factor $\tau_i \in T$*

With TGMs, and model updates I define the task of *online collective inference*.

Definition 13 (Online Collective Inference)**.** *Let $\mathbb{T}_1 = (T_1, \Phi_1, P_1(Y_1|X_1), Y_1, X_1)$ be a TGM. Then apply a series of updates that converts T_1, X_1, and Y_1, to T_2, X_2, and Y_2. Online collective inference is the task of instantiating the potentials Φ_2 using every $\tau_i \in T_2$ to get $\mathbb{T}_2 =$*

$(T_2, \Phi_2, P(Y_2|X_2), Y_2, X_2)$ *and performing inference over the newly defined probability distribution* $P_2(Y_2|X_2)$.

A common inference task is obtaining a *maximum-a-posteriori (MAP) estimator* of the random variables. For a distribution $P(Y|X = \mathbf{x})$, a MAP estimator, $\mathbf{y}^*$, achieves the mode, i.e., $\mathbf{y}^* = \arg\max_{\mathbf{y}} P(Y = \mathbf{y}|X = \mathbf{x})$.

5.2.3 Model Instantiation

As mentioned in Section 5.2.2, a vital subproblem of online collective inference is generating potentials, i.e., grounding.

Definition 14 (Grounding)**.** *Let* $T = \{\tau_1, \cdots, \tau_s\}$ *and* $\Gamma_1, \cdots, \Gamma_s$ *be a set of template factors and corresponding domains. Grounding is the process of, for every* $\tau_i \in T$*, realizing all tuples of random variables* $(Z_1, \cdots, Z_{n^i}) \in \Gamma_i$*, and instantiating every potential* $\phi_{\hat{i}}(Z_1, \cdots, Z_{n^i})$.

Realizing every tuple of random variables in a domain, Γ, of a template factor, τ, is difficult as Γ can be large and a non-trivial subset of the full n-ary Cartesian product of the range of template variables associated with τ. Reductions to this set are generally performed, such as removing potentials that will not modify the optimal setting of the random variables, i.e., *trivial* potentials. There has been a considerable amount of research on identifying trivial potentials and designing scalable algorithms for grounding Richardson and Domingos [2006], Bach et al. [2017]. I expand upon these efforts to scale the grounding process in an orthogonal direction through online-specific improvements.

Context-Aware Grounding

Rather than re-perform the entire grounding process to instantiate an updated model, I introduce *context-aware grounding*. This class of methods leverage the practical observation that model updates will typically preserve much of the initial TGM's existing structure. Context-aware grounding makes a minimal edit, adding and deleting potentials from the initial potential set.

Definition 15 (Context-Aware Grounding)**.** *Let* $\mathbb{T}_1 = (T_1, \Phi_1, P(Z_1), Z_1)$ *and* $\mathbb{T}_2 = (T_2, \Phi_2, P(Z_2), Z_2)$*, be two TGMs. A context-aware grounding instantiates* Φ_2 *by grounding the set* $\Phi_+ = \Phi_2 \setminus \Phi_1$ *and removing* $\Phi_- = \Phi_1 \setminus \Phi_2$*, i.e.,* $\Phi_2 = \Phi_+ \cup (\Phi_1 \setminus \Phi_-)$*.*

I derive a tight upper bound on the number of new potentials generated by a context-aware grounding.

Theorem 6. *Let* $\mathbb{T} = (T, \Phi, P(Z), Z)$ *be a TGM, and* T_+*,* Z_+ *be the sets of template factors and random variables added after a series of model updates, respectively. Define a set of context template factors* $T_c \subseteq T_+ \cup T$ *such that each* $\tau_i \in T_c$ *is a function of at least one tuple of random variables containing some* $Z_j \in Z_+$*. Build the set of context variables for each* τ_i*, denoted as* Z_{τ_i}*, to be the complete set of variables* $Z \cup Z_+$ *if* $\tau_i \in T_+$*, or* Z_+*, otherwise. Then*

$$|\Phi_+| \leq \sum_{\tau_i \in \mathrm{T}_c} \sum_{j=1}^{|\tau_i|} \binom{|\tau_i|}{j} (|Z_{\tau_i}|^j \cdot |Z|^{|\tau_i|-j}) \tag{5.8}$$

where $|\tau_i|$ *is the number of template variables defining* τ_i*.*

Approximate Context-Aware Groundings

Theorem 6 shows that the number of potentials instantiated by a context-aware grounding will grow exponentially with respect to the number of context variables and context template factors. Thus, generating all of the new potentials can be a costly operation that may not be viable in all settings. I therefore propose *approximate context-aware grounding*:

Definition 16 (Approximate Context-Aware Grounding)**.** *Let Φ_+ be a set of potentials generated from a context-aware grounding. An approximate context-aware grounding is any process which generates a set of potentials $\tilde{\Phi}_+$ that is a proper subset of Φ_+, i.e., $\tilde{\Phi}_+ \subset \Phi_+$.*

An example of an approximate context-aware grounding is one that generates potentials which contain less than some threshold number of context variables, say κ. This is motivated by observing that this restricts the upper limit of the inner summation in the bound of Theorem 6. This results in a degree κ polynomial growth rate with respect to the number of context variables, context template factors, and random variables.

5.2.4 Stability and Regret of Online Inference

In this section I analyze the stability and regret of the MAP states of TGMs belonging to a log-concave and continuous exponential family Wainwright and Jordan [2008]. This implies potentials in (5.7) are constrained to the form: $\phi(\mathbf{y}, \mathbf{x}) = \exp(-\theta\psi(\mathbf{y}, \mathbf{x}))$ for some continuous convex function $\psi(\cdot)$ and a non-negative real valued parameter vector θ. The MAP inference

problem of a TGM in such a class of distributions with m potential functions is:

$$\max_{\mathbf{y}} P(Y = \mathbf{y}|X = \mathbf{x}) \equiv \min_{\mathbf{y}} \sum_{i=1}^{m} \theta_i \psi_i(\mathbf{y}, \mathbf{x}) \tag{5.9}$$

Further, I let $H(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{m} \theta_i \psi_i(\mathbf{y}, \mathbf{x})$ denote the *MAP objective*. Formally, for all TGMs in this section, I make the following assumption.

Assumption 1 *$P(Y|X)$ is a member of a log-concave exponential family.*

Stability

Broadly speaking, stability ensures that small changes to the input result in bounded variation in the output. A result of Assumption 1 is that the MAP objectives are convex, which follows from the definition of log-concavity. Specifically for stability analysis, an additional useful assumption is strong convexity. Strong convexity of a function ensures a unique minimizer and thus stability of MAP states simplifies to analyzing the distance between points rather than sets.

Definition 17 (Strong Convexity)**.** *$f : \mathbb{R}^n \to \mathbb{R}$ is α-strong convex iff for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and $g \in \partial f(\mathbf{x}_1)$* [9]

$$f(\mathbf{x}_2) - f(\mathbf{x}_1) \geq g^T(\mathbf{x}_2 - \mathbf{x}_1) + \frac{\alpha}{2}\|\mathbf{x}_2 - \mathbf{x}_1\|_2^2 \tag{5.10}$$

Strong convexity of the MAP objective can be ensured by squared L_2 regularization. This is a consequence of the fact that an $\alpha > 0$ parameterized squared L_2 regularizer is α-strong convex, and strong convexity is preserved when summed with convex functions.

Lemma 1 *Let $l = f + h$ where f is convex and h is α-strong convex. Then l is α-strong convex.* [10]

[9] $\partial f(\mathbf{x})$ here denotes the set of subgradients of f at $\mathbf{x}$.

[10] Proof available in Shalev-Shwartz [2012].

The strong convexity condition yields an upper bound on the distance a point is from a minimizer.

Lemma 2 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be α-strong convex. Suppose $\mathbf{x}^*$ is a minimizer of f, for all $\mathbf{x} \in \mathbb{R}^n$ and $g \in \partial f(\mathbf{x})$* †

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \frac{2}{\alpha}\|g\|_2$$

This lemma yields a bound on the distance of any setting of the unobserved random variables to the unique minimizer of an L_2 regularized MAP objective. I thus have a direction for analyzing the stability of MAP states by bounding the magnitude of gradients of MAP states after a sequence of model updates. My approach leverages an additional assumption bounding the rate of change of gradients in the MAP objective of the updated TGM, H_2, i.e, β-smoothness.

Definition 18 (β-Smoothness)**.** *$f : \mathbb{R}^n \to \mathbb{R}$ is β-smooth over $\Omega \subseteq \mathbb{R}^n$ iff for all $\mathbf{x}_1, \mathbf{x}_2 \in \Omega$*

$$\|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\|_2 \leq \beta\|\mathbf{x}_1 - \mathbf{x}_1\|_2 \tag{5.11}$$

Assumption 2 *The MAP objective $H(\mathbf{y}, \mathbf{x})$ of the distribution $P(Y|X)$, is β-smooth as a function of both arguments $\mathbf{y}$ and $\mathbf{x}$.*

To account for the complexity of a sequence of updates relating the models $\mathbb{T}_1$ and $\mathbb{T}_2$, I define the delta model as the set of potentials which are added and removed by a sequence of model updates.

Definition 19 (Delta Model)**.** *Given the TGMs $\mathbb{T}_1 = (T_1, \Phi_1, P(Z_1), Z_1)$ and $\mathbb{T}_2 = (T_2, \Phi_2, P(Z_1), Z_2)$. Define $\Phi_- = \{\exp(\theta\psi_i) | \phi_i = \exp(-\theta\psi_i) \in \Phi_1 \setminus \Phi_2\}$ and $\Phi_+ = \Phi_2 \setminus \Phi_1$. The potential set $\Phi_\Delta = \Phi_- \cup \Phi_+$ is a delta model with a corresponding MAP objective $H_\Delta = \sum_{\phi \in \Phi_\Delta} -\log(\phi_i(\mathbf{y}, \mathbf{x}))$.*

Applying the lemmas and assumptions introduced in this section, I derive the following bound on the distance between MAP states for two TGMs.

Theorem 7. *Let* $\mathbb{T}_1 = (T_1, \Phi_1, P_1(Y_1|X_1), Y_1, X_1)$ *and* $\mathbb{T}_2 = (T_2, \Phi_2, P(Y_2|X_2), Y_2, X_2)$ *be two TGMs defining the MAP objectives* H_1 *and* H_2*. Suppose* $P_1(Y_1|X_1)$ *and* $P_2(Y_2|X_2)$ *satisfy Assumption 1 and* $P_2(Y|X)$ *satisfies Assumption 2. Let* Φ_Δ *be the delta model relating* $\mathbb{T}_1$ *to* $\mathbb{T}_2$ *with MAP objective* H_Δ*. Denote the vectors of observed random variable values of* $\mathbb{T}_1$ *and* $\mathbb{T}_2$ *as* $\mathbf{x}_1 \in \mathbb{R}^{|X_1|}$ *and* $\mathbf{x}_2 \in \mathbb{R}^{|X_2|}$*, and MAP states* $\mathbf{y}_1^* = \arg\min_{\mathbf{y}} H_1(\mathbf{y}, \mathbf{x}_1)$ *and* $\mathbf{y}_2^* = \arg\min_{\mathbf{y}} H_2(\mathbf{y}, \mathbf{x}_2)$*. Let* $\tilde{\mathbf{y}}_1^*$ *and* $\tilde{\mathbf{x}}_1$ *be the vectors* $\mathbf{y}_1^*$ *and* $\mathbf{x}_1$*, such that values corresponding to deleted variables are removed, and values corresponding to new variables and changing partitions are added. Note, values for variables moving from the unobserved to the observed partition are the values from* $\mathbf{y}_1^*$*. Let* δ *to be the change in the observed variable values, i.e.,* $\delta = \|\tilde{\mathbf{x}}_1 - \mathbf{x}_2\|_2$*. Then*

$$\|\tilde{\mathbf{y}}_1^* - \mathbf{y}_2^*\|_2 \leq 2\frac{\beta}{\alpha}\delta + \frac{2}{\alpha}\|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2 \tag{5.12}$$

Proof. First note that

$$\delta = \|\tilde{\mathbf{x}}_1 - \mathbf{x}_2\|_2 = \left\| \begin{bmatrix} \tilde{\mathbf{y}}_1^* \\ \\ \tilde{\mathbf{x}}_1 \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{y}}_1^* \\ \\ \mathbf{x}_2 \end{bmatrix} \right\|_2$$

Thus β-Smoothness of H_2 implies

$$\begin{aligned}
\beta^2\delta^2 &\geq \|\nabla H_2(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1) - \nabla H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2^2 \\
&= \|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1) - \nabla_{\mathbf{y}} H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2^2 \\
&\quad + \|\nabla_{\mathbf{x}} H_2(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1) - \nabla_{\mathbf{x}} H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2^2 \\
&\geq \left(\|\nabla_{\mathbf{y}} H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2 - \|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2\right)^2 \\
\implies \beta\delta &\geq \|\nabla_{\mathbf{y}} H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2 - \|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2
\end{aligned}$$

Rearranging terms I have

$$\beta\delta + \|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2 \geq \|\nabla_{\mathbf{y}} H_2(\tilde{\mathbf{y}}_1^*, \mathbf{x}_2)\|_2$$

Lastly, applying the α-strong convexity bound, Lemma 2, on $\|\tilde{\mathbf{y}}_1^* - \mathbf{y}_2^*\|_2$ I have

$$\|\tilde{\mathbf{y}}_1^* - \mathbf{y}_2^*\|_2 \leq 2\frac{\beta}{\alpha}\delta + \frac{2}{\alpha}\|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2$$

□

Warm-Starts A direct application of the above theorem motivates the notion that a related TGM's MAP estimator can be used to seed inference in a state closer than random initialization. I define a *warm-start*, $\tilde{\mathbf{y}}_1^*$, for a TGM $\mathbb{T}_2$ as the augmented MAP state of a related TGM $\mathbb{T}_1$. A *cold-start*, $\mathbf{y}_{\text{cold}}$, is drawn uniformly at random over the domain of the random variable values. If the MAP objectives of the distributions defined by $\mathbb{T}_1$ and $\mathbb{T}_2$ satisfy the necessary assumptions for Theorem 7, then the bound in the theorem can be applied to prove that a warm-start is closer to $\mathbb{T}_2$'s MAP state than a cold-start in expectation. More formally, if it can be shown that a sequence of model

updates results in a change in observed variable values $\delta = \|\tilde{\mathbf{x}}_1 - \mathbf{x}_2\|_2$ and a delta model gradient $\|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2$ such that $2\frac{\beta}{\alpha}\delta + \frac{2}{\alpha}\|\nabla_{\mathbf{y}} H_\Delta(\tilde{\mathbf{y}}_1^*, \tilde{\mathbf{x}}_1)\|_2 \leq E\left[\|\mathbf{y}_{\text{cold}} - \mathbf{y}^*\|_2\right]$, then Theorem 7 yields $E\left[\|\tilde{\mathbf{y}}_1^* - \mathbf{y}_2^*\|_2\right] \leq E\left[\|\mathbf{y}_{\text{cold}} - \mathbf{y}^*\|_2\right]$.

Regret Bounds

Regret in online optimization is typically defined as the difference between the total loss incurred by two competing settings of the random variable values, called hypotheses Shalev-Shwartz [2012]. Typically in collective inference, the total loss of a hypothesis is defined as a function of the set of potentials. However, in this setting, it is possible for the set of potentials to be modified. For this reason it is important to explicitly specify the set of potentials functions over which the regret is being computed.

Definition 20 (Regret)**.** *Let Φ be a set of potential functions, $\mathcal{L}(\mathbf{x}; \Phi) : \mathbb{R}^n \rightarrow \mathbb{R}$ be a loss function parameterized by Φ, and $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$. The regret with respect to the two competing hypothesis, $\mathbf{x}_1$, and $\mathbf{x}_2$, and Φ is defined as*

$$\textit{Regret}(\mathbf{x}_1, \mathbf{x}_2; \boldsymbol{\Phi}) = \mathcal{L}(\mathbf{x}_1; \boldsymbol{\Phi}) - \mathcal{L}(\mathbf{x}_2; \boldsymbol{\Phi}) \tag{5.13}$$

For MAP inference, the loss function is the MAP objective, i.e., $\text{Regret}_H(\mathbf{y}_1, \mathbf{y}_2) = H(\mathbf{y}_1, \mathbf{x}) - H(\mathbf{y}_2, \mathbf{x})$.

I are specifically interested in bounding the MAP regret relative to MAP estimators for two distinct TGMs defined over the same set of random variables. To do this I reformulate the problem into that of analyzing the stability of MAP states of TGMs subject to model updates. I

then apply Theorem 7 together with an added assumption bounding the rate of change of the MAP objective, namely L-Lipschitz continuity.

Definition 21 (L-Lipschitz Continuity)**.** *$f : \mathbb{R}^n \to \mathbb{R}$ is L-Lipschitz continuous over $\Omega \subseteq \mathbb{R}^n$ iff for all $\mathbf{x}_1, \mathbf{x}_1 \in \Omega$*

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|_2 \tag{5.14}$$

Assumption 3 *The MAP objective $H(\mathbf{y}, \mathbf{x})$, of the distribution $P(Y|X)$, is L-Lipschitz as a function of $\mathbf{y}$ for any $\mathbf{x}$.*

Theorem 8. *Let $\mathbb{T}_1$ and $\mathbb{T}_2$ be two TGMs defined over the same random variables. Further, suppose the distributions defined by $\mathbb{T}_1$ and $\mathbb{T}_2$, $P_1(Y|X)$ and $P_2(Y|X)$, satisfy Assumption 1, and $P_2(Y|X)$ satisfies Assumption 2 and Assumption 3. Let Φ_Δ be the delta model relating $\mathbb{T}_1$ to $\mathbb{T}_2$ with MAP objective H_Δ. Then, let $\mathbf{y}_1^*$ and $\mathbf{y}_2^*$ be the MAP states of $\mathbb{T}_1$ and $\mathbb{T}_2$, respectively.*

$$\textit{Regret}_{H_2}(\mathbf{y}_1^*, \mathbf{y}_2^*) \leq 2\frac{L}{\alpha}\|\nabla_{\mathbf{y}} H_\Delta(\mathbf{y}_1^*, \mathbf{x})\|_2 \tag{5.15}$$

Proof. As $H_2(\mathbf{y}, \mathbf{x})$ is L-Lipschitz continuous,

$$\|H_2(\mathbf{y}_1^*, \mathbf{x}) - H_2(\mathbf{y}_2^*, \mathbf{x})\| \leq L\|\mathbf{y}_1^* - \mathbf{y}_1^*\|_2 \tag{5.16}$$

Then, since $\mathbf{y}_2^* = \arg\min_{\mathbf{y}} H_2(\mathbf{y}, \mathbf{x})$

$$H_2(\mathbf{y}_1^*, \mathbf{x}) - H_2(\mathbf{y}_2^*, \mathbf{x}) \leq L\|\mathbf{y}_1^* - \mathbf{y}_1^*\|_2 \tag{5.17}$$

Next, noting that $\delta = 0$ and applying Theorem 7 yields

$$H_2(\mathbf{y}_1^*, \mathbf{x}) - H_2(\mathbf{y}_2^*, \mathbf{x}) \leq 2\frac{L}{\alpha}\|\nabla_{\mathbf{y}} H_\Delta(\mathbf{y}_1^*, \mathbf{x})\|_2 \tag{5.18}$$

□

5.2.5 Empirical Evaluation

In this section, I evaluate the performance of an online collective inference system implemented in PSL (*online PSL*), and empirically validate the theory introduced in this section. I answer the following questions: Q1) Are model updates performed via context-aware groundings faster than grounding a new model? Q2) How does approximate context-aware grounding affect the runtime and performance of online collective inference? Q3) How well do the theoretical bounds introduced in Section 5.2.4 predict the movement of the MAP states? Q4) In practice, are warm-starts typically closer to the updated model's MAP state than cold-starts?

Datasets and Models

I explore these questions on three online collective inference tasks: *online recommendation*, *online demand forecasting*, and *online model selection* [11]. These tasks showcase how online collective inference can be applied in practical settings, while demonstrating common model update patterns. The datasets for the tasks are as follows:

MovieLens – MovieLens is a movie recommendation dataset containing approximately 1M timestamped ratings made by 6K users on 4K movies Harper and Konstan [2015]. Although often used as an offline recommendation dataset, timestamps in the data allow us to turn this dataset into an online recommendation problem using the following procedure. 10 splits are uniformly sampled, each using 70% of the original data. Each split is then partitioned into 20 time steps, where the first time step contains one third of the split's data and the rest are evenly partitioned.

[11] Data and code will be made available upon final submission.

Two variants of the dataset are created for the online recommendation task: *MovieLens-Fixed* and *MovieLens-TimeSeries*. In *MovieLens-Fixed*, all ratings are always present as either observations or unknowns. At each time step, previously unknown ratings are observed. In *MovieLens-TimeSeries*, ratings are added incrementally. At each time step, 20% of the unknowns from the previous time step are observed and all ratings in the new time step are unknown.

BikeShare – *BikeShare* is a dataset that contains information for 650k trips between 70 stations by customers of the bicycle sharing service, Bay Area Bike Share Bay Area Bike Share [2016]. The task for this dataset is to predict the demand for bikes at each station. The data is divided into 10 overlapping splits, where each split contains one third of the original data. Splits are partitioned into 20 time steps, where the first time step contains one third of the split's data and the rest are evenly partitioned. At each time step, the demand for the previous time step becomes fully observed and the demand for the next time is added as unknown.

Epinions – *Epinions* is a trust prediction dataset with 2k users with 8.5k directed links representing whether one user trusts the other. The data is divided into 8 splits with the trust links partitioned into observed and unknown sets following the same procedure as Bach et al. (2017). Each split is then partitioned into 10 time steps, where the first time step contains the full model and each subsequent time step adds and removes templates from the model.

Methods

Across all tasks and data variants, I evaluate the performance of three methods for executing model updates and performing MAP inference. First, **Offline** is a standard PSL implementation

that executes model updates by grounding the full model at each time step. Then, **Regret-Free** and **Regretful** are online PSL implementations executing model updates via context-aware grounding and approximate context-aware grounding, respectively. The regretful models are grounded using the example described in Section 5.2.3 with $\kappa = 1$.

Performance and Regret Analysis

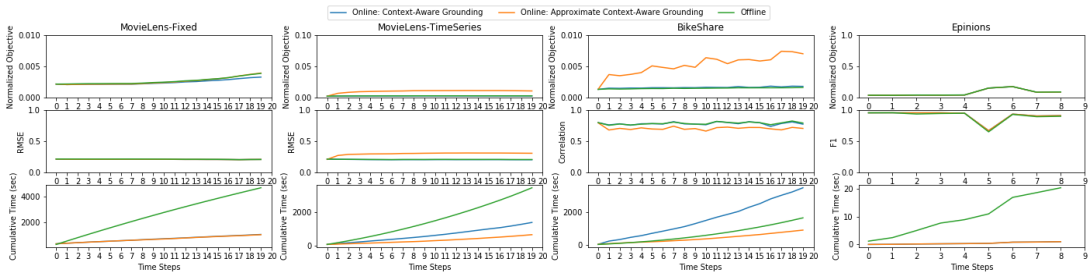


Figure 5.3: Comparison of normalized MAP inference objective (top), domain-specific evaluation metrics (middle), and runtime (bottom) of the three systems.

To address questions Q1 and Q2, I compare the performance of my online and offline methods across each dataset. Figure 5.3 shows at each time step the cumulative time in seconds to obtain a MAP prediction, a problem-specific evaluation metric, and the normalized MAP inference objective. Because the MAP inference problem is strongly convex, the Online PSL implementation employing exact context-aware grounding yields the same predictions as the offline method.

In the MovieLens-Fixed and Epinions setting no variables are added or deleted, thus there is no significant difference between the time, evaluation, and MAP objective performance of the exact and approximate online models. In Epinions, potentials are all initially grounded and then at each timestep a subset is removed to obtain the desired model. Removed potentials are cached and

added back to the model after inference is completed. Because the grounding cost after the initial timestep in the MovieLens-Fixed and Epinions setting is virtually eliminated, the online methods are able to complete over 4 times faster than the offline method in MovieLens-Fixed, and over 20 times faster in Epinions without incurring any regret.

Next I examine the results of the MovieLens-TimeSeries and BikeShare datasets in Figure 5.3. Both of these datasets operate similarly, where unknowns in the previous time step become observed as well as unknowns being introduced in the new time step. In both problems, the regretful online method performs $2 - 5$ times faster than the regret-free online method, but this comes at the cost of an inferior MAP inference objective and evaluation score.

Stability Analysis

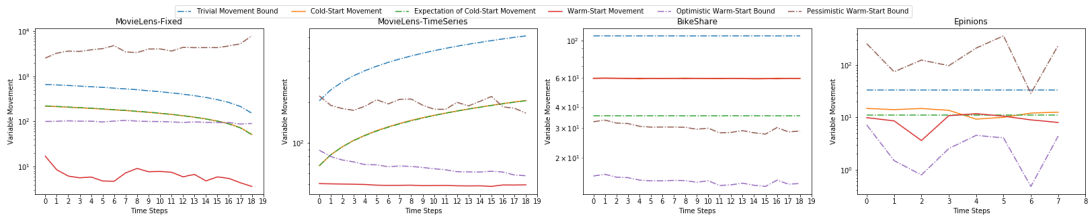


Figure 5.4: Comparison of warm-start and cold-start variable movements plotted with various theoretical and empirically estimated bounds.

To address question Q3 and Q4, I estimate model complexity parameters of the TGMs instantiated by PSL and measure the movement of variables with warm-start and cold-start initializations. Figure 5.4 shows the measured variable movement on all problems. Also shown in the figure are two empirical estimates of the warm-start bound derived in Section 5.2.4, one pessimistic and one optimistic. As optimization is performed, β and α estimates are created by sampling the

rate of change in the gradients. Both the optimistic and pessimistic warm-start bounds take β to be the maximum observed rate of change of gradient. Then, the α estimate is the average of the rates in the optimistic bound and the average between 0.1, the regularization parameter of the model, and the minimum observed rate for the pessimistic bound. I also plot the expectation of the cold-start variable movement which I assume to be the expected distance between two randomly sampled points in a hypercube Anderssen et al. [1976]. Moreover, a trivial upper bound on the possible movement considering only the $[0, 1]$ variable box constraints enforced by PSL is plotted.

The expected and actual cold-start movements are aligned in all experiments except BikeShare. This behavior suggests that the assumption that the distance between a random initialization and the MAP state of a TGM instantiated by PSL is reasonable. The BikeShare exception can be explained by the fact that truth values in this setting are mostly near zero.

Next, in the MovieLens-Fixed, MovieLens-TimeSeries, and Epinions experiments, the cold-start movement of the variables is significantly larger than that of the warm-start. This matches the intuition of the settings, as there are many unobserved variables shared between time steps that are already optimized for a subset of the potentials in the updated model. Then, as expected, the exception of this behavior is in the BikeShare experiments. Cold-start and warm-start initializations are in fact equivalent in BikeShare because all unobserved variables for a time step are introduced by the model update.

In all experiments, the optimistic warm-start bound closely follows the actual warm-start movement, verifying theoretical predictions of Theorem 7 and hence Theorem 8. Furthermore, the optimistic warm-start bound consistently falls below the expected cold-start movement. The

pessimistic bound on the other hand typically overestimates the warm-start variable movement by a large-margin, even predicting more movement than the trivial bound in some cases. The optimistic and pessimistic warm-start bounds do drop below the actual warm-start movement in both the BikeShare and Epinions experiments. This can be explained by the empirical nature of this experiment, the true α and β and parameters of this model are only estimated via a sampling.

5.3 Conclusions and Future Work

In this chapter, I show how SRL systems can be enhanced to handle models that adapt to different datasets and changes within a dataset. Weight learning allows SRL models to specialize to specific domains and datasets through learning weights for template rules, I specifically show how search-based approaches are effective on a wide range of models. Additionally, I show how existing models can be updated with new data through online collective inference.

Some future directions for weight learning revolve around learning weights without fully grounding the model. There are a variety of approaches for avoiding full grounding [Sarkhel et al., 2015, 2016] that would be interesting to integrate into my approach. Further, performance of Gaussian processes are highly dependent on the kernel function used. Therefore, in one of my search-based approaches (*Bayesian optimization for weight learning (*BOWL*)*) that use Gaussian process, an exploration of different kernels for BOWL could further improve the performance of my method.

Future directions for online collective inference are to introduce methods for reducing

the size of the instantiated graphical model by summarizing or forgetting some of the model. This could further improve the speed up over offline methods and reduce the memory requirements of the system. An orthogonal approach to forgetting that could also result in runtime improvements is implementing approximate inference techniques.

Chapter 6

Usability

Usability refers to the ease with which a user can interact with an SRL system. Non-SRL machine learning frameworks (e.g., scikit-learn, Tensorflow, and PyTorch) have a straightforward and simple pipeline: flatten the data into a single homogeneous feature table, fit on labeled training features, and then predict on test features. Unfortunately, SRL is more complicated because of its collective and transductive nature. Structure in rules and data must be specified, observed data must be provided for both training and testing, and debugging predictions requires looking at other related predictions. Creating SRL tools that allows users familiar with machine learning, but not necessarily SRL, is a substantial roadblock to making SRL practical. In this chapter, I discuss my work in making SRL frameworks that are accessible to all machine learners[1].

[1]The work presented in this chapter was presented at the 2020 ACM Conference on Recommender Systems (RecSys) Rodden et al. [2020].

6.1 Standard Relational Language Interface

When choosing to implement a SRL model, there are several widely-used languages available including but not limited to: Markov Logic Networks (MLNs) Richardson and Domingos [2006], ProbLog Raedt et al. [2007], PSL Bach et al. [2017], and Probabilistic Circuits Choi et al. [2020]. Each of these languages has several implementation available where the implementations include different levels of support and features, vary in scalability, and are written in different programming languages. The large set of options may seem enticing, but from a developer's perspective the abundance is overwhelming. Additionally, each of these languages and implementations use different terminology, concepts, and semantics for handling data and handling logical rules. A developer wishing to implement the same SRL model in two different frameworks may find the task to be trivial or impossible depending on the model and frameworks. This large degree of variability between frameworks hinders SRL research and can make SRL hostile to new comers.

However at its core SRL is about adding structure to predictions, and SRL frameworks can be described in four components: data, model, grounding, and computation. To start, an SRL framework needs a description of its data and the structure associated with it; for example the framework will need to know about what logical predicates to expect and their arguments. Then, the framework needs to know about the structure of the problem itself; most SRL frameworks represent this structure using logical rules. Finally, the SRL framework needs to know what computation needs to be done over the data and model, e.g., MAP or marginal inference.

In this section, I introduce the Standard Relational Language Interface (SRLi). SRLi

is a generic SRL framework written in Python that decomposes the SRL pipeline into the above components and allows users to build SRL models that are general and agnostic to the underlying engine. A general representation of the SRL pipelines allows users to easily invoke different computational engines which may use algorithms that perform better on specific datasets or have different scaling properties. I demonstrate the power of SRLi by creating a varied collection of SRL models that run (unmodified) over several different existing SRL frameworks. Additionally, SRLi allows modelers to create new SRL engines by reusing components of existing SRL engines. I demonstrate this ability be creating multiple new SRL engines in SRLi by utilizing shared components.

My key contributions to this component of the usability pillar include: 1) decomposing of SRL into four independent components, 2) creating of a general SRL framework based on the above four components, 3) implementing existing SRL frameworks in SRLi, 4) implementing new SRL frameworks in SRLi that utilize shared components.

6.1.1 The Components of SRL

At its core, SRL is about adding structure to predictions. Different SRL frameworks implement the integration of structure and prediction in very different ways. SRLi is an attempt to define the main steps of an SRL pipeline and decouple them as much as possible. Decoupling these steps allows SRLi to make them independent, generalizable, and exchangeable. SRLi refers to module that implements all the steps of the SRL pipeline as an *engine*. Throughout this section, I will compare SRLi to three of the most popular SRL frameworks: ProbLog (based on Prolog) Raedt

et al. [2007] Tuffy (an MLN implementation) Niu et al. [2011], and Probabilistic Soft Logic Bach et al. [2017].

Data

In classical machine learning, data is typically represented as a single table of numeric features. Features that do not fit into this paradigm (like categorical features) are flattened, converted to numeric values, and treated the same as the other features. However, SRL treats its data very differently. Much like a relational database, data in SRL is represented as multiple dependent tables.

SRLi abstracts data into structures it refers to as *relations*. Like "predicates" from first-order logic or "relations" from relational algebra and relational database theory, SRLi relations have a unique name and arguments. For example, the relation HASLABEL(`User`, `Label`) may be used to represent categorical predictions of a user. This relation has the name HASLABEL and argument `User` and `Label`. Each argument is associated with a named domain that denotes what values may substitute for each argument. These domains may be specified by the user, and when they are not specified SRLi will infer them from the data and model.

In addition to managing data, SRLi relations need to capture two features widely used in SRL languages: data constraints and priors.

Data constraints Constraints that are applied directly to a single relation, as opposed to general constraints that apply across multiple instances of relations, are referred to as *data constraints*. For example, consider a relation, HASLABEL(`User`, `Label`), that predicts a label for a user from the

possible labels: $\{1, 2, 3\}$. I may want to enforce a constraint on the model that only one label can be predicted for each user. Different SRL frameworks represent and implement this constraint in very different ways. ProbLog uses a logical rule with a special operator called an *annotated disjunction*:

```
1/3 :: hasLabel(User, 1) ; 1/3 :: hasLabel(User, 2) ; 1/3 :: hasLabel(User, 3) .
```

PSL represents this constraint using an arithmetic rule with a special "summation" syntax:

```
HasLabel(User, +Label) = 1.0 .
```

Tuffy uses a special syntax when defining logical predicates:

```
HasLabel(user, label!)
```

SRLi takes an approach similar to Tuffy and represents data constraints as an attribute of a predicate. This allows SRLi (unlike PSL and ProbLog) to make a distinction between constraints that operate over the data and constraints that operate over the entire model. This method of defining properties of SRLi data can be seen as similar to data description language (DDL) used in SQL.

```
srli.relation.Relation('HasLabel',
                       arity = 2, variable_types = ['User', 'Label'],
                       sum_constraint = srli.relation.SumConstraint('='))
```

Priors Priors, most often implemented as "negative priors", allow models to add in a bias or regularizer to each relation. In the most simple form, a negative prior enforces a belief that in the absence of evidence, predictions should favor false / 0. For example, consider a relation, HASLINK(NodeA, NodeB), that predicts if a link exists between two nodes in a graph. Given no evidence we typically want predict that no link exists between the two nodes, but still allow a small

chance that links can exist without additional evidence (for example, we may want to set the link probability without evidence to be a small value such as 0.01). ProbLog can use a set of rules with an intermediate predicate that allows false values a small chance to swap to true:

```
hasLink(NodeA, NodeB) :- hasLink_intermediate(NodeA, NodeB) .
0.001 :: hasLink(NodeA, NodeB) :- \+hasLink_intermediate(NodeA, NodeB) .
```

Tuffy uses a simple logical rule:

```
0.001 !HasLink(nodeA, nodeB)
```

Similar to Tuffy, PSL also uses a simple logical rule:

```
0.001: !HasLink(NodeA, NodeB)
```

As in the case of the data constraint, SRLi represents a negative prior as an attribute of the relation itself, which allows SRLi to separate details of the data from details of the model.

```
srli.relation.Relation('HasLink',
                       arity = 2, variable_types = ['Node', 'Node'],
                       negative_prior_weight = 0.01)
```

Model

As discussed in Section 2.1, SRL frameworks typically use weighted logical rules to represent the core of their models. However, different SRL frameworks use different conventions, syntax, semantics, and representations of probability. For example, ProbLog is based on horn clauses Lloyd [1987]; Markov Logic Networks (MLNs) Richardson and Domingos [2006] allow both universal and existential quantification and is not restricted to horn clauses; and PSL only

uses universal quantification and requires logical rules to reduce to a 1-DNF Kononenko and Kukar [2007], but also allows for non-logical templates in the form of inequalities of linear combinations.

SRLi uses first-order logic with the same restrictions as PSL, but limits rules to purely logical rules. Choosing this subset of logic allows SRLi to represent a wide range of models, while still being able to translate models into languages like MLN which use a wider subset of first-order logic.

Grounding

Grounding is a key component in any SRL system and is discussed in detail in Section 2.3. Despite being critical to SRL performance, the grounding component of SRL systems is often overlooked and under optimized. At its simplest (and slowest), grounding can be implemented with just nested loops. But as Section 3.2 proves, grounding systems can be very complex and have a very large impact on performance.

To provide the best grounding possible, SRLi utilizes the grounding interface provided by PSL. Because of SRLi's general representation for data, rules, and ground rules, any SRL framework implemented in SRLi gets trivial access to state-of-the-art grounding.

Computation

There are several different types of computation done in SRL frameworks. The two most common are inference (discussed in Section 2.4) and weight learning (discussed in Section 2.4.1). SRLi's primary contribution here is developing a clean interface and tools for future methods to

implement their own inference and weight learning methods. However, SRLi also provides several inference and weight learning implementations (discussed in Section 6.1.2 and Section 6.1.3).

6.1.2 Recreating Existing SRL Frameworks

SRLi can be used to recreate the most popular SRL frameworks. Here, I will show how SRLi can be used to recreate ProbLog, Tuffy, and PSL. These three different frameworks have vastly different syntaxes, semantics, and implementations. The ability to recreate them all speaks to the flexibility of SRLi.

ProbLog

The latest version of ProbLog (ProbLog 2 [2]) is implemented in Python. This allows for easier integration into SRLi, which is also written in Python.

Tuffy (MLN)

Implementing a Tuffy engine in SRLi is complicated by the fact that Tuffy does not have a Python interface. Additionally, Tuffy has a required dependency on a PostgreSQL database that require specific settings and modules. To meet all these requirements while still providing a safe and flexible framework for users, I implement the Tuffy SRLi engine as a Docker container. Docker is an open-source tool that allows for software to be run in reproducible and sand boxed lightweight containers.

[2] `https://github.com/ML-KULeuven/problog`

PSL

PSL is written in Java, but provides a Python interface (`pslpython`[3]) which makes implementing a SRLi PSL engine straightforward. Additionally, PSL is used as the default grounding component in SRLi. PSL's grounding API is also accessed through the same Python interface.

6.1.3 Implementing Custom SRL Engines

A key part of SRLi is the independence of the different components. Making the different parts of the SRL pipeline independent from their implementation allows these components to be shared. Here, I create several custom SRLi engines that utilize a mix of components from different engines and custom components.

MLN Native

MLNs were one of the first popular SRL representations, with the most used implementations being Alchemy Kok et al. [2009] and Tuffy Niu et al. [2011]. Implementing MLNs in SRLi can be done simply by using the common grounding component provided by SRLi and a custom implementation of the base MaxWalkSat algorithm (the default inference method used in Tuffy). The inference algorithm implemented in this engine is simple when compared to the mature implementation used by Tuffy. However, this engine benefits from advanced grounding techniques that Tuffy cannot use (since Tuffy tightly couples its implementation of the different SRL components).

[3] `https://pypi.org/project/pslpython/`

MLN PySat

Building off of the native MLN engine, this engine swaps out the MaxWalkSat inference component for PySat Ignatiev et al. [2018], a popular SAT solver for Python.

Non-Collective ProbLog

The non-collective ProbLog engine (ProbLog NC) builds off of the previous ProbLog engine, but attempts to address scaling issues often seen in ProbLog. Because ProbLog is built on top of Prolog, it shares some of the same scalability issues. For example, both Prolog and ProbLog has difficulty working with logical rules that contain cycles (which are quite common in many SRL problems). To help address this issue, ProbLog implements memoization Fierens et al. [2015]. However, larger models can still create issues for ProbLog.

To address scaling issues in ProbLog, a common technique is to reduce to scope of a ProbLog program to as little as one random variable at a time Tsamoura et al. [2020]. Typically, this is done in an ad-hoc manner for each dataset. However because SRLi has full information about the data and model, it can automatically reduce the scope of ProbLog programs. First, SRLi can pre-ground the entire ProbLog program using SRLi's common grounding component. Then, this engine can iterate through each random variable and compute its Markov blanket (the set of all variables the target variables is linked to through ground rules). Now this engine can create ProbLog programs that are scoped to deal with only one Markov blanket at a time. One iteration is complete after all variables/blankets have been checked. The engine continues until it reaches the maximum

number of iterations or until convergence.

6.1.4 A Complete SRLi Model

Below is the full SRLi implementation of the CITESEER model used in Section 6.1.5.

```
link = srli.relation.Relation('Link',
                              arity = 2, variable_types = ['Paper', 'Paper'])
hascat = srli.relation.Relation('HasCat',
                                arity = 2, variable_types = ['Paper', 'Label'],
                                negative_prior_weight = 0.001,
                                sum_constraint = srli.relation.Relation.SumConstraint())


rules = [
    # Neighbor has cat => has cat.
    "HasCat(A, C) & Link(A, B) & (A != B) -> HasCat(B, C)",
    "HasCat(A, C) & Link(B, A) & (A != B) -> HasCat(B, C)",
    # Per category rules.
    "HasCat(A, '1') & Link(A, B) -> HasCat(B, '1')",
    "HasCat(A, '2') & Link(A, B) -> HasCat(B, '2')",
    "HasCat(A, '3') & Link(A, B) -> HasCat(B, '3')",
    "HasCat(A, '4') & Link(A, B) -> HasCat(B, '4')",
    "HasCat(A, '5') & Link(A, B) -> HasCat(B, '5')",
    "HasCat(A, '6') & Link(A, B) -> HasCat(B, '6')",
    "HasCat(A, '7') & Link(A, B) -> HasCat(B, '7')",
]

link.add_observed_data(path = os.path.join(DATA_DIR, 'eval', 'link_obs.txt'))
hascat.add_observed_data(path = os.path.join(DATA_DIR, 'eval', 'hasCat_obs.txt'))
hascat.add_unobserved_data(path = os.path.join(DATA_DIR, 'eval', 'hasCat_targets.txt'))
hascat.add_truth_data(path = os.path.join(DATA_DIR, 'eval', 'hasCat_truth.txt'))

engine = srli.engine.psl.engine.PSL(relations = [link, hascat], rules = rules)
results = engine.solve()

evaluation = srli.evaluation.CategoricalAccuracy(hascat)
print("Accuracy: ", evaluation.evaluate(results))
```

Engine	Parent SRL Framework	Class
ProbLog	ProbLog	srli.engine.problog.engine.ProbLog
Tuffy	Tuffy	srli.engine.tuffy.docker.Tuffy
PSL	PSL	srli.engine.psl.engine.PSL
MLN Native	Custom	srli.engine.mln.native.NativeMLN
MLN PySat	Custom	srli.engine.mln.pysat.PySATMLN
Non-Collective ProbLog	ProbLog	srli.engine.problog.noncollective.NonCollectiveProbLog

Table 6.1: SRL engines provided by SRLi.

Note how this example uses SRLi's PSL engine. However, because the model and data are independent of the computational engine, using a different engine only requires a change to one line. Table 6.1 shows the details of all the engines discussed in this section.

6.1.5 Empirical Evaluation

To evaluate the effectiveness of SRLi, I run the different SRLi engines over several known relational problems[4]. The datasets (detailed in Table 6.2) are from the official PSL examples repository[5], and were used without modifications to the model or dataset. To ingest the examples, a SRLi connector was written that can process PSL configuration file. All experiments were run on the same machine with 128 GB of RAM and 20 threads clocked at 3.1 GHz.

First, I examine SRLi's ability to run the engines described in this section over the

[4]All code for these SRLi experiments is available at `https://github.com/eriq-augustine/srli-experiments`.

[5]Models and data are available at `https://github.com/linqs/psl-examples`.

Dataset	Ground Rules	Task	Source
CITESEER	36K	CC	Bach et al. [2017]
CORA	41K	CC	Bach et al. [2017]
DDI	1M	LP	Sridhar et al. [2016]
EPINIONS	14K	LP	Huang et al. [2013]
ER	3M	CC	Bhattacharya and Getoor [2007]
JESTER	1M	REC	Bach et al. [2017]
KGI	90K	LP	Pujara et al. [2013]
LASTFM	1.4M	REC	Kouki et al. [2015]
SMOKERS	21	CC	Niu et al. [2011]
STANCE	2K	CC	Sridhar et al. [2015]
YELP	500K	REC	Kouki et al. [2015]

Table 6.2: Details of datasets and models used for evaluation. The generic task performed on each dataset include collective classification (CC), link prediction (LP), and recommendation (REC).

Dataset	MLN Native	MLN PySAT	PSL	ProbLog	ProbLog NC	Tuffy
CITESEER	✓	✓	✓	TIMEOUT	✓	✓
CORA	✓	✓	✓	TIMEOUT	✓	✓
DDI	✓	✓	✓	TIMEOUT	TIMEOUT	✓
EPINIONS	✓	✓	✓	TIMEOUT	✓	✓
ER	✓	✓	✓	TIMEOUT	✓	TIMEOUT
JESTER	✓	✓	✓	TIMEOUT	TIMEOUT	TIMEOUT
KGI	TIMEOUT	✓	✓	TIMEOUT	✓	MEMORY
LASTFM	TIMEOUT	✓	✓	TIMEOUT	✓	TIMEOUT
SMOKERS	✓	✓	✓	✓	✓	✓
STANCE	✓	✓	✓	TIMEOUT	✓	✓
YELP	TIMEOUT	✓	✓	TIMEOUT	✓	TIMEOUT

Table 6.3: The result of running multiple SRLi engines. A check mark (✓) means that the engine ran without issue, TIMEOUT indicates that the engine was killed after two hours, and MEMORY indicates that the engine ran out of memory.

provided datasets. Table 6.3 shows the result of running each engine. A check mark (✓) indicates that the engine ran without issue, TIMEOUT indicates that the engine was killed after two hours, and MEMORY indicates that the engine ran out of memory.

Most engines are able to run on most unmodified datasets without issue. As all these datasets and models are provided by PSL, it is not surprising that the PSL engine runs all datasets without issue. The ProbLog engine sees the most issues, and can only run the SMOKERS dataset.

This result is expected as the provided datasets are not partitioned into smaller subgraphs. However, the non-collective ProbLog engine (ProbLog NC) is able to run on all but two of the larger datasets (DDI and JESTER). Tuffy times out on many of the large datasets. MLN Native, however, is able to demonstrate the power of SRLi's common grounding component by successfully running two more datasets than Tuffy. The MLN PySat engine is able to go even further and run all datasets.

To assess the quality of results that the different SRLi engines output, I can look at the output from engines on datasets that did not time out (CITESEER, CORA, EPINIONS, and STANCE-CREATEDEBATE). Table 6.4 shows results for all the engines (except ProbLog) averaged over five different splits. The mean of the evaluation metric used in the original publication for each respective dataset along with the standard deviation is provided. The runtimes for these experiments are listed in Table 6.5. The best performing engines for each dataset (determined using a Student's T-test with a $p = 0.01$) are in bold. These results do not include weight learning, and therefore represent a worst-case scenario of how each engine can perform without any dataset-specific tuning. The PSL engine tends to perform the best, and is often tied with one of the available MLN engines.

6.2 VMI-PSL: Visual Model Inspector for Probabilistic Soft Logic

One of the appeals of PSL models is that they are intuitive to create and make it easy to include background knowledge and complex data interactions. However, these complex interactions can sometimes make PSL models difficult to inspect, debug, and understand. While there has been significant work in explaining SRL models Symeonidis et al. [2009], Friedrich and Zanker [2011],

Dataset	Metric	MLN Native	MLN PySAT	PSL	ProbLog NC	Tuffy
CITESEER	Accuracy	**0.23 ± 0.01**	0.16 ± 0.01	**0.24 ± 0.01**	0.16 ± 0.01	0.22 ± 0.01
CORA	Accuracy	**0.30 ± 0.02**	0.13 ± 0.01	**0.33 ± 0.02**	0.17 ± 0.02	0.28 ± 0.02
EPINIONS	AuPRC	0.92 ± 0.01	0.92 ± 0.01	**0.98 ± 0.01**	0.92 ± 0.01	0.95 ± 0.01
STANCE-CREATEDEBATE	RMSE	0.72 ± 0.04	0.73 ± 0.03	**0.60 ± 0.02**	0.68 ± 0.03	**0.59 ± 0.11**

Table 6.4: The mean quality metric and standard deviation of different SRLi methods run over several established relational datasets averaged over 5 splits. The presented metric is the metric used in the original publication. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold.

Dataset	MLN Native	MLN PySAT	PSL	ProbLog NC	Tuffy
CITESEER	1042 ± 40	110 ± 23	**7 ± 1**	5444 ± 3341	35 ± 2
CORA	1550 ± 61	325 ± 32	**8 ± 1**	1992 ± 116	50 ± 4
EPINIONS	151 ± 16	13 ± 1	**11 ± 1**	256 ± 23	892 ± 137
STANCE-CREATEDEBATE	5 ± 2	**2 ± 1**	4 ± 4	199 ± 132	7 ± 2

Table 6.5: The mean runtime and standard deviation of different SRLi methods run over several established relational datasets averaged over 5 splits. The best results (determined using a Student's T-test with a $p = 0.01$) are in bold.

Bostandjiev et al. [2012], Nguyen et al. [2013], Verbert et al. [2013], Parra et al. [2014], Berkovsky et al. [2017], Kouki et al. [2017b], Nunes and Jannach [2017], Sato et al. [2018], Andjelkovic et al. [2019], there has been far less work in debugging them.

In this section, I present my inspection tool for PSL: Visual Model Inspector for Probabilistic Soft Logic (VMI-PSL), and demonstrate its use on hybrid recommender systems Kouki et al. [2015]. VMI-PSL provides an easy-to-use web-based inspector for PSL models that requires no additional setup, uses data tables and aggregate statistics to display information about a model in a scalable way, and uses contextual information to display only relevant portions of the model.

To understand how a PSL model can be inspected, I must first discuss the basic structure of a PSL model. A PSL model is constructed by instantiating weighted logical rules with data, these instantiations are referred to as *ground rules*. The components in a ground rule are referred to as *ground atoms*. Ground atoms assume a continuous *truth value* in the range $[0, 1]$, and ground rules take on a degree of *satisfaction* in the range $[0, 1]$.

My key contributions to this component of the usability pillar include: 1) I augment the PSL framework to output the necessary information for model inspection, 2) I create a new visual inspection tool for PSL models, VMI-PSL, [6] 3) I provide sample workflows to show how VMI-PSL can be used with a recommender system model. To the best of my knowledge, VMI-PSL is the first generic visual model inspector for any SRL framework.

The work covered in this section was presented at the 2020 ACM Conference on Recommender Systems (RecSys) Rodden et al. [2020].

[6] Code available at `https://github.com/linqs/psl-vmi`.

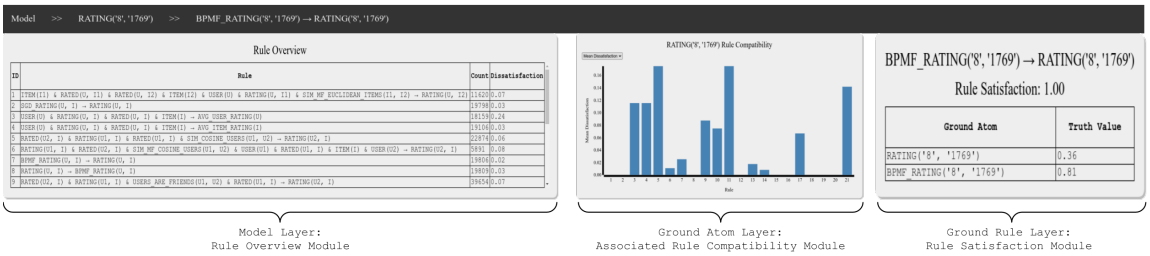


Figure 6.1: The three different context layers in VMI-PSL, along with a module from each layer. Here, the ground atom RATING('8', '1769') is the context for the `ground atom` layer and the ground rule BPMF_RATING('8', '1769') → RATING('8', '1769') is the context for the `ground rule` layer.

6.2.1 System Overview

VMI-PSL is designed to mimic the workflow of an experienced PSL user as they manually inspect a PSL model. Information is organized into into three different *context layers*, each containing *modules* that automatically activate when a user selects an item. These layers are the `model` layer, the `ground atom` layer, and the `ground rule` layer. The information displayed starts general at the `model` layer, but then becomes more detailed as the user drills down into more specific context layers. Figure 6.1 shows a module from each of the three layers on a hybrid recommender system modelKouki et al. [2015].

The `model` layer contains information about the model as a whole, and contains five modules: 1. `Rule Overview` – a table of all the rules used in the model along with each rule's weight, count, total dissatisfaction, and mean dissatisfaction. 2. `Violated Constraints` – a table of all violated hard constraints. 3. `Rule Compatibility` – a bar graph containing all the rules, the display can be toggled between total satisfaction, mean satisfaction, total dissatisfaction, and mean dissatisfaction. 4. `Ground Rule Count` – a bar graph showing the number of ground

rules instantiated from each rule. 5. `Truth Table` – a table that shows every ground truth value and its corresponding prediction.

When a ground atom is selected, the `ground atom` layer is activated with two modules: 1. `Associated Rule Compatibility` – a bar graph similar to the `model` layer's `Rule Compatibility` module, however only ground rules containing the selected ground atom are aggregated. 2. `Associated Ground Rules` – a table that shows every ground rule that contains the selected ground atom along with the dissatisfaction of the ground rule.

Lastly, the `ground rule` layer contains a single module: 1. `Rule Satisfaction` – a table that shows the total satisfaction of the ground rule along with the truth value of each ground atom involved in that ground rule.

6.2.2 Example Workflows

VMI-PSL is supported in all PSL interfaces, and no additional setup is required. To use VMI-PSL, a user must first run their existing PSL model with the additional `--visualization` option. PSL generates a JSON file that contains all the necessary information for model inspection. The user must then go to `https://linqs.github.io/psl-vmi` and select the generated JSON file when prompted. Finally, VMI-PSL rebuilds the model from the JSON file and displays all the relevant information.

Figure 6.2 shows three different workflows of VMI-PSL on the HyPER hybrid recommender modelKouki et al. [2015]. In the *Debug* workflow, VMI-PSL is being used to answer the question: Why is my model giving incorrect results? A common cause of erroneous models is a

Figure 6.2: Three different workflows showcasing different use cases of VMI-PSL: debugging an erroneous model, analyzing the performance of components of a working model, and explaining specific predictions.

rule unintentionally generating zero ground rules. This can be caused by corrupted data files or incorrectly written rules. The workflow for this use case is simple, the `Rule Overview`, `Rule Compatibility`, and `Ground Rule Count` modules will all show zeroes in their respective metrics for that rule. The *Analysis* workflow answers the question: What components of my model can be improved? To answer this question, the first step is to select an atom with an incorrect prediction from the `Truth Table` module. Then, the `Ground Atom` layer is activated and the `Associated Rule Compatibility` can be used to find the rules that most contribute to the incorrect prediction. Finally, the *Explainability* workflow answers the question: What causes a specific prediction? The first step is looking up the specific prediction in the `Truth Table` module. Once the prediction is selected, the `Associated Ground Rules` module can be sorted in ascending order by dissatisfaction to find the specific ground rules that contribute most to this prediction. Selecting each ground rule will open the `Rule Satisfaction` module where the individual contribution of each ground atom in the selected rule can be observed. These explanations are found in the same way as Kouki et al. [2019].

6.3 Conclusions and Future Work

In this chapter, I discuss two tools I created to make SRL more usable to those in the SRL community and more accessible to machine learners outside of the SRL community. The first tool I discuss, SRLi, is a framework that serves as a standard interface for relational languages, with the ability to convert relational models into various existing frameworks as well as the ability to create new SRL engines using shared and custom components. This common interface will allow the SRL community to coalesce behind a common language, and lowers the bar for others wishing to implement SRL models. The second tool, VMI-PSL, is a tool for inspecting and debugging PSL models.

Future directions in SRLi falls into two categories. First, SRLi's support for existing frameworks can be extended to other relational models, Probabilistic Circuits Choi et al. [2020] being one of the most obvious choices. Next, SRLi's functionality can be extended by allowing models from common machine learning frameworks, such as scikit-learn, Tensorflow, and PyTorch, to be integrated directly into SRLi models. This integration would be an easy way for newcomers to augment an existing model with relational information.

A promising future direction for VMI-PSL is to attach VMI-PSL to an online inference engine for PSL, making it possible to see the effects of changing rules or atoms without needing to re-run PSL. Additionally, making VMI-PSL compatible with SRLi allows for VMI-PSL to automatically support a wider range of SRL engines.

Chapter 7

Conclusions and Future Work

In this dissertation, I have discussed the four pillars that are central in making practical SRL systems that can be used by both SRL experts and those new to the SRL community: *scalability*, which gives SRL systems the ability be used with large, complex, and realworld models; *expressivity*, which lets SRL systems represent many different problems and data patterns; *model adaptability*, which allows SRL systems to handle models that change; and *usability*, which makes SRL systems accessible for users with varying degrees of expertise with machine learning and SRL.

My key contributions are: 1) identifying the roadblocks that prevent SRL from becoming a more widely-used paradigm of machine learning, 2) advancing the bounds of SRL scalability for both grounding and inference, 3) integrating neural models with symbolic reasoning, 4) extending existing SRL models to incorporate new and changing data, and 5) creating a standardized framework for working with existing and custom SRL engines.

Each chapter covering a pillar of practical SRL systems contains directions for future work specific to that pillar, however here I would like to highlight a broader vision of the future. Machine learning is powerful, and we see it become a larger and larger part of everyday life. We have reached a point of diminishing returns when it comes to the size and amount of training data that can be encoded in a simple, "flat" machine learning model. Instead, it it time for us to make models that are smarter and work directly with curated knowledge. Looking to the future, we need systems that can integrate the powerful models we see around us everyday with human intention. We need models that can make use of common sense from a knowledge base, that can learn from tiny amounts of data, that can integrate guardrails and safety constraints, that can explain its predictions in an easily interpretable fashion, and that can codify societal desires in every prediction. SRL provides a mean of creating these model by incorporating curated knowledge (whether that be constraints, domain knowledge, social graphs, or large knowledge bases) into machine learning. The groundwork I have laid in this dissertation serves as a starting point for building these integrated systems that can be used effectively by everyone in the machine learning community.

Bibliography

Umut Acar, Alexander Ihler, Ramgopal Mettu, and Ozgur Sumer. Adaptive updates for map configurations with applications to bioinformatics. In *IEEE Workshop on Statistical Signal Processing (SSP)*, 2009.

Somak Aditya, Yezhou Yang, and Chitta Baral. Explicit reasoning over end-to-end neural architectures for visual question answering. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018a.

Somak Aditya, Yezhou Yang, Chitta Baral, and Yiannis Aloimonos. Combining knowledge and reasoning through probabilistic soft logic for image puzzle solving. In *Uncertainty in Artificial Intelligence (UAI)*, pages 238–248, 2018b.

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

Alok Aggarwal and Jeffrey S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.

Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *ML*, 92:91–132, 2013.

Sultan S. Al-Qahtani, Ellis E. Eghan, and Juergen Rilling. Recovering semantic traceability links between apis and security vulnerabilities: An ontological modeling approach. In *ICST*, 2017.

Marco Alberti, Elena Bellodi, Giuseppe Cota, Fabrizio Riguzzi, and Riccardo Zese. cplint on swish: Probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11:47–64, 2017.

Afnan Alhazmi, Tom Blount, and George Konstantinidis. Forbackbench: A benchmark for chasing vs. query-rewriting. *International Conference on Very Large Data Bases (VLDB)*, 15(8):1519–1532, 2022.

Duhai Alshukaili, Alvaro A. A. Fernandes, and Norman W. Paton. Structuring linked data search results using probabilistic soft logic. In *ISWC*, pages 3–19, 2016. doi: 10.1007/978-3-319-46523-4_1.

R. Anderssen, R. Brent, D. Daley, and P. Morgan. Concerning $\int_0^1 \cdots \int_0^1 (x_k^2 + \cdots + x_k^2)^{1/2} dx_1 \cdots dx_k$ and a taylor series method. *SIAM Journal on Applied Mathematics*, 30(1):22–30, 1976.

Ivana Andjelkovic, Denis Parra, and John O'Donovan. Moodplay: interactive music recommendation based on artists' mood similarity. *International Journal of Human-Computer Studies*, 121: 142–159, 2019.

Eriq Augustine and Lise Getoor. A comparison of bottom-up approaches to grounding for templated markov random fields. In *Conference on Machine Learning and Systems (MLSys)*, 2018.

Eriq Augustine and Lise Getoor. Collective grounding: Applying database techniques to grounding templated models. In *International Conference on Very Large Data Bases (VLDB)*, Vancouver, BC, USA, 2023.

Eriq Augustine, Theodoros Rekatsinas, and Lise Getoor. Tractable probabilistic reasoning through effective grounding. In *ICML workshop on TPM*, 2019.

Stephen Bach, Bert Huang, and Lise Getoor. Unifying local consistency and max sat relaxations for scalable inference with rounding guarantees. In *AIStats*, pages 46–55, 2015.

Stephen H. Bach, Matthias Broecheler, Lise Getoor, and Dianne P. O'Leary. Scaling MPE Inference for Constrained Continuous Markov Random Fields with Consensus Optimization. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

Stephen H. Bach, Bert Huang, Ben London, and Lise Getoor. Hinge-loss markov random fields: Convex inference for structured prediction. In *Uncertainty in Artificial Intelligence (UAI)*, Bellevue, WA, USA, 2013. AUAI.

Stephen H. 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. In *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, 2005.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Gökhan Baki, Thomas Hofmann, Bernhard Schölkopf, Alexander Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data*. The MIT Press, 2007.

Bay Area Bike Share. OPEN DATA, 2016. URL `http://www.bayareabikeshare.com/open-data`.

Islam Beltagy and Raymond Mooney. Efficient markov logic inference for natural language semantics. In *International Workshop on Statistical Relational AI (StarAI)*, 2014.

Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond Mooney. Montague meets Markov: Deep semantics with probabilistic logical form. In *SEM*, 2013.

Islam Beltagy, Katrin Erk, and Raymond Mooney. Probabilistic soft logic for semantic textual similarity. In *Association for Computational Linguistics (ACL)*, pages 1210–1219. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-1114.

Adán L. Benito and Joshua D. Reiss. Intelligent multitrack reverberation based on hinge-loss Markov random fields. In *Semantic Audio*, 2017.

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

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

Shlomo Berkovsky, Ronnie Taib, and Dan Conway. How to recommend? user trust factors in movie recommender systems. In *Intelligent User Interfaces (IUI)*, Limassol, Cyprus, 2017. ACM.

Jonathan W. Berry, Kina Kincher-Winoto, Cynthia A. Phillips, Eriq Augustine, and Lise Getoor. Entity resolution at large scale: Benchmarking and algorithmics. Technical report, Sandia National Laboratories, Albuquerque, NM, USA, 2018.

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. *arXiv*, 2017.

Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1:1–36, 2007.

Mustafa Bilgic, Lise Geetor, and Louis Licamela. D-dupe: An interactive tool for entity resolution

in social networks. In *IEEE Symposium On Visual Analytics Science And Technology (VAST)*, Baltimore, MD, USA, 2005. IEEE.

Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *International Conference on Machine Learning (ICML)*, 2017.

Svetlin Bostandjiev, John O'Donovan, and Tobias Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *ACM Conference on Recommender Systems (RecSys)*, Dublin, Ireland, 2012. ACM.

Leon Bottou. Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics (COMPSTAT)*, 2010.

Leon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018.

George Box, Gwilym Jenkins, Gregory Reinsel, and Greta Ljung. *Time series analysis: Forecasting and control*. John Wiley & Sons, 2015.

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*, 3(1):1–122, 2010.

Timothy Van Bremen, Anton Dries, and Jean Christoph Jung. Ontology-mediated queries over probabilistic data via probabilistic logic programming. In *International Conference on Information and Knowledge Management (CIKM)*, Beijing, China, 2019. ACM.

Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2010.

Guy Van Den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2178–2185. AAAI Press, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-363.

David Buchman and David Poole. Negative probabilities in probabilistic logic programs. *International Journal of Approximate Reasoning (IJAR)*, 83:43–59, 2017a.

David Buchman and David Poole. Why rules are complex: Real-valued probabilistic logic programs are not fully expressive. In *Uncertainty in Artificial Intelligence (UAI)*, 2017b.

Hung Hai Bui, Tuyen N. Huynh, and Sebastian Riedel. Automorphism groups of graphical models and lifted variational inference. In *UAI*, 2013.

Wray Buntine. Theory refinement on bayesian networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1991.

Luis M De Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning (IJAR)*, 51(7):785–799, 2010.

Rose Catherine and William Cohen. Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *ACM Conference on Recommender Systems (RecSys)*, Boston, MA, USA, 2016. ACM.

Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. *International Conference on Very Large Data Bases (VLDB)*, 87:1–4, 1987.

Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder for english. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv*, 2019.

Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing (STOC)*, Boulder, Colorado, USA, 1977. ACM.

Melisachew Wudage Chekol, Giuseppe Pirrò, Joerg Schoenfisch, and Heiner Stuckenschmidt. Tecore: Temporal conflict resolution in knowledge graphs. *International Conference on Very Large Data Bases (VLDB)*, 10:Iss–12, 2017.

H. Chen, W. Ku, H. Wang, L. Tang, and M. Sun. Linkprobe: Probabilistic inference on large-scale social networks. In *IEEE International Conference on Data Engineering (ICDE)*, 2013.

H. Chen, W. Ku, H. Wang, L. Tang, and M. Sun. Scaling up markov logic probabilistic inference for social graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 29(2):433–445, 2017.

Po-Ta Chen, Feng Chen, and Zhen Qian. Road traffic congestion monitoring in social media with hinge-loss markov random fields. In *ICDM*, pages 80–89. IEEE Computer Society, 2014. doi: 10.1109/ICDM.2014.139.

Junho Choi, Chang Choi, Eunji Lee, and Pankoo Kim. *Markov Logic Network Based Social Relation Inference for Personalized Social Search*, pages 195–202. Springer-Verlag, 2015.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, Oct 2020. URL `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`.

Li Chou, Somdeb Sarkhel, Nicholas Ruozzi, and Vibhav Gogate. On parameter tying by quantization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3241–3247, 2016.

Petr Cintula, Christian Fermüller, and Carles Noguera. *Handbook of Mathematical Fuzzy Logic*. College Publications, 2015.

Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv*, 2015.

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

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Symposium on Theory of Computing*, 1971.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning (ICML)*, 2016.

Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

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

Mayukh Das, Yuqing Wu, Tushar Khot, Kristian Kersting, and Sriraam Natarajan. Scaling lifted probabilistic inference and learning via graph databases. In *SIAM International Conference on Data Mining (SDM)*, pages 738–746, 2016.

Mayukh Das, Devendra Singh Dhami, Gautam Kunapuli, Kristian Kersting, and Sriraam Natarajan. Fast relational probabilistic inference and learning: Approximate counting via hypergraphs. In *AAAI*, 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.

Artur S. 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.

Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *AI*, 171(2-3): 73–106, 2007.

Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *UAI*, 2012.

Lingjia Deng and Janyce Wiebe. Joint prediction for entity/event-level sentiment analysis using probabilistic soft logic models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 179–189. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1018.

Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, Twin Cities, Minnesota, USA, 2011.

Charles Dickens, Rishika Singh, and Lise Getoor. Hyperfair: A soft approach to integrating fairness criteria. In *RecSys Workshop on Responsible Recommendation (FAccTRec)*, Rio de Janeiro, Brazil, 2020.

Charles Dickens*, Eriq Augustine*, Connor Pryor, and Lise Getoor. Negative weights in hinge-loss markov random fields. In *Workshop on Tractable Probabilistic Modeling (TPM)*, Virtual, 2021.

Charles Dickens, Connor Pryor*, Eriq Augustine, Alex Miller, and Lise Getoor. Context-aware

online collective inference for templated graphical models. In *International Conference on Machine Learning (ICML)*, Virtual, 2021.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *International Conference on Machine Learning and Applications (ICMLA)*, 2017.

Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, Williston, VT, USA, 1st edition, 2009.

Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Igor Douven and Hans Rott. From probabilities to categorical beliefs: Going beyond toy models. *Journal of Logic and Computation*, 28(6):1099–1124, 2018.

Javid Ebrahimi, Dejing Dou, and Daniel Lowd. Weakly supervised tweet stance classification by relational bootstrapping. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1012–1017. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1105.

Varun Embar, Golnoosh Farnadi, Jay Pujara, and Lise Getoor. Aligning product categories using anchor products. In *KBC*, 2018.

Varun Embar, Sriram Srinivasan, and Lise Getoor. Tractable marginal inference for hinge-loss

markov random fields. In *ICML Workshop on Tractable Probabilistic Modeling (TPM)*, Long Beach, CA, USA, 2019. TPM.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research (JAIR)*, 61:1–64, 2018.

Khan Mohammad Al Farabi, Somdeb Sarkhel, and Deepak Venugopal. Efficient weight learning in high-dimensional untied mlns. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Golnoosh Farnadi, Stephen H. Bach, Marie-Francine Moens, Lise Getoor, and Martine De Cock. Soft quantification in statistical relational learning. *Machine Learning*, 2017.

Golnoosh Farnadi, Pigi Kouki, Spencer K. Thompson, Sriram Srinivasan, and Lise Getoor. A fairness-aware hybrid recommender system. In *RecSys Workshop on Responsible Recommendation (FATREC)*, Vancouver, Canada, 2018a. ACM.

Golnoosh Farnadi, Jie Tang, Martine De Cock, and Marie-Francine Moens. User profiling through deep multimodal fusion. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 171–179, 2018b.

Golnoosh Farnadi, Behrouz Babaki, and Lise Getoor. A declarative approach to fairness in relational domains. *IEEE Data Engineering Bulletin*, 42(3):36–48, 2019.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon,

Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning (ML)*, 32:41–62, 1998.

Nir Friedman and Moises Goldszmidt. Sequential update of bayesian network structure. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1997.

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 99, pages 1300–1309, 1999.

Gerhard Friedrich and Markus Zanker. A taxonomy for generating explanations in recommender systems. *AI Magazine*, 32(3):90–98, 2011.

Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2 edition, 2008.

Thomas Geier and Susanne Biundo. Approximate online inference for dynamic markov logic networks. In *ICTAI*, pages 764–768, 2011.

Lise Getoor and Benjamin Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Deepanway Ghosal, Navonil Majumder, Alexander Gelbukh, Rada Mihalcea, and Soujanya Poria.

COSMIC: COmmonSense knowledge for eMotion identification in conversations. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Michael Robert Glass and Ken Barker. Focused grounding for markov logic networks. In *FLAIRS*, 2012.

Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2008.

Vibhav Gogate and Pedro M. Domingos. Exploiting logical structure in lifted probabilistic inference. In *AAAI Workshop on StarAI*, 2010.

Vibhav Gogate, Abhay Kumar Jha, and Deepak Venugopal. Advances in lifted importance sampling. In *AAAI*, 2012.

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *IR*, 4:133–151, 2001.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems (NeurIPS*, 2014.

Eric Gribkoff and Dan Suciu. Slimshot: In-database probabilistic inference for knowledge bases. *International Conference on Very Large Data Bases (VLDB)*, 9(7):552–563, 2016.

Mourad Gridach, Hatem Haddad, and Hala Mulki. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In *ACL NUT*, pages 21–30, 2017.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

Jan Van Haaren, Guy Van den Broeck, Wannes Meert, and Jesse Davis. Lifted generative learning of markov logic networks. *ML*, 103:27–55, 2016.

Fabian Hadiji and Kristian Kersting. Reduce and re-lift: Bootstrapped lifted likelihood maximization for MAP. In *AAAI*, 2013.

Petr Hájek, Lluís Godo, and Francesc Esteva. Fuzzy logic and probability. In *Uncertainty in Artificial Intelligence (UAI)*, 1995.

Alon Y. Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

Maxwell Harper and Joseph Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4), 2015.

Matthew Harrison-Trainor, Wesley Holliday, and Thomas Icard III. Preferential structures for comparative probabilistic reasoning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Thomas Hartman. On functions representable as a difference of convex functions. *Pacific Journal of Mathematics*, 9(3):707–713, 1959.

Alexander L Hayes. srlearn: A python library for gradient-boosted statistical relational models. In *International Workshop on Statistical Relational AI (StarAI)*, 2020.

Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

Eduard Hovy. Generating natural language under pragmatic constraints. *Journal of Pragmatics*, 11 (6):689–719, 1987.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

Bert Huang, Angelika Kimmig, Lise Getoor, and Jennifer Golbeck. A flexible framework for probabilistic models of social trust. In *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP)*, Washington, DC, USA, 2013. Springer-Verlag.

Xuedong Huang, Yasuo Ariki, and Mervyn Jack. *Hidden Markov Models for Speech Recognition*. Columbia University Press, 1990.

Tuyen N. Huynh and Raymond J. Mooney. Max-margin weight learning for markov logic networks. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 564–579, 2009.

Tuyen N. Huynh and Raymond J. Mooney. Online Max-margin Weight Learning with Markov Logic Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010a.

Tuyen N. Huynh and Raymond J. Mooney. Online max-margin weight learning with markov logic networks. In *International Workshop on Statistical Relational AI (StarAI)*, pages 32–37, 2010b.

Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi: 10.1007/978-3-319-94144-8_26. URL `https://doi.org/10.1007/978-3-319-94144-8_26`.

M. M. Islam, K. Mohammad Al Farabi, S. Sarkhel, and D. Venugopal. Scaling up inference in mlns with spark. In *Big Data*, 2018.

TS Jayram, Phokion G Kolaitis, and Erik Vee. The containment problem for real conjunctive queries with inequalities. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Chicago, IL, USA, 2006a. ACM.

TS Jayram, Phokion G Kolaitis, and Erik Vee. The containment problem for real conjunctive queries with inequalities. In *International Conference on Management of Data (SIGMOD)*, pages 80–89, 2006b.

Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine learning*, 77(1):27–59, 2009.

Kristen Johnson and Dan Goldwasser. "all I know about politics is what I read in twitter": Weakly supervised models for extracting politicians' stances from twitter. In *COLING*, 2016a.

Kristen Johnson and Dan Goldwasser. Identifying stance by analyzing political discourse on twitter. In *EMNLP workshop on NLP and CSS*, 2016b.

Kristen Johnson, I-Ta Lee, and Dan Goldwasser. Ideological phrase indicators for classification of political discourse framing on twitter. In *ACL workshop on NLP and CSS*, 2017.

Donald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

Andreas Kamilaris and Francesc X. Prenafeta-Boldu. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture (CEA)*, 147:70–90, 2018.

Fisnik Kastrati and Guido Moerkotte. Optimization of disjunctive predicates for main memory column stores. In *International Conference on Management of Data (SIGMOD)*, pages 731–744. ACM, 2017.

Henry Kautz, Bart Selman, and Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. *Satisfiability Problem: Theory and Applications*, 35:573–586, 1996.

Seyed Mehran Kazemi and David Poole. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *KR*, pages 561–564. AAAI Press, 2016.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. The multiple-choice knapsack problem. In *Knapsack Problems*, pages 317–347. Springer, Catonsville, MD, USA, 2004.

Kristian Kersting. Lifted probabilistic inference. In *ECAI*, pages 33–38. IOS Press, 2012.

Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In *UAI*, 2009.

Kristian Kersting, Martin Mladenov, and Pavel Tokmakov. Relational linear programming. *AI*, 244: 188–216, 2017.

Mahmoud Abo Khamis, Phokion G Kolaitis, Hung Q Ngo, and Dan Suciu. Bag query containment and information theory. *ACM Transactions on Database Systems (TODS)*, 46(3):1–39, 2021.

Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning Markov logic networks via functional gradient boosting. In *IEEE International Conference on Data Mining (ICDM)*, 2011.

Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. Markov logic networks for natural language question answering. In *Association for Computational Linguistics (ACL)*, 2015a.

Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. Exploring Markov logic networks for question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015b.

Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. Lifted graphical models: a survey. *Machine Learning*, 99:1–45, 2015. doi: 10.1007/s10994-014-5443-2.

Angelika Kimmig, Alex Memory, Renée J. Miller, and Lise Getoor. A collective, probabilistic approach to schema mapping using diverse noisy evidence. *IEEE Transactions on Knowledge and Data Engineering*, 31(8):1426–1439, 2019.

Diederik Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Erich Klement, Radko Mesiar, and Endre Pap. *Triangular Norms*. Springer, 2000.

George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic - Theory and Applications*. Prentice Hall, 1995.

Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *ICML*, pages 441–448, 2005.

Stanley Kok, Parag Singla, Matthew Richardson, Pedro Domingos, Marc Sumner, and Hoifung Poon. The alchemy system for statistical relational ai: User manual, 2007.

Stanley Kok, Marc Sumner, Matthew Richardson, Parag Singla, Hoifung Poon, Daniel Lowd, Jue Wang, and Pedro Domingos. The alchemy system for statistical relational ai. Technical report, Department of Computer Science and Engineering, University of Washington, 2009.

Phokion G Kolaitis and Moshe Y Vardi. Conjunctive-query containment and constraint satisfaction.

In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Seattle, WA, USA, 1998. ACM.

Phokion G Kolaitis and Moshe Y Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences (JCSS)*, 61(2):302–332, 2000.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

Igor Kononenko and Matjaz Kukar. *Machine learning and data mining*. Horwood Publishing, 2007.

Pigi Kouki, Shobeir Fakhraei, James Foulds, Magdalini Eirinaki, and Lise Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *RecSys*. ACM, 2015.

Pigi Kouki, Jay Pujara, Christopher Marcum, Laura M. Koehly, and Lise Getoor. Collective entity resolution in familial networks. In *ICDM*, 2017a.

Pigi Kouki, James Schaffer, Jay Pujara, John O'Donovan, and Lise Getoor. User preferences for hybrid explanations. In *ACM Conference on Recommender Systems (RecSys)*, Como, Italy, 2017b. ACM.

Pigi Kouki, Jay Pujara, Christopher Marcum, Laura Koehly, and Lise Getoor. Collective entity resolution in multi-relational familial networks. *KAIS*, pages 1–35, 2018.

Pigi Kouki, James Schaffer, Jay Pujara, John Odonovan, and Lise Getoor. Personalized explanations

for hybrid recommender systems. In *Intelligent User Interfaces (IUI)*, Los Angeles, CA, USA, 2019. ACM.

Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision (IJCV)*, 123(1):32–73, 2017.

Aviral Kumar, Sunita Sarawagi, and Ujjwal Jain. Trainable calibration measures for neural networks from kernel mean embeddings. In *International Conference on Machine Learning (ICML)*, 2018.

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.

Harold Kushner and George Yin. *Stochastic approximation and recursive algorithms and applications*. Springer, 2003.

Ondrej Kuželka. Complex markov logic networks: Expressivity and liftability. In *Uncertainty in Artificial Intelligence (UAI)*, 2020.

Sarasi Lalithsena, Sujan Perera, Pavan Kapanipathi, and Amit P. Sheth. Domain-specific hierarchical subgraph extraction: A recommendation use case. *Big Data*, pages 666–675, 2017a. doi: 10.1109/BigData.2017.8257982.

Sarasi Lalithsena, Sujan Perera, Pavan Kapanipathi, and Amit P. Sheth. Domain-specific hierarchical subgraph extraction: A recommendation use case. In *BigData*, 2017b.

Luís C. Lamb, Artur S. 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 *International Joint Conference on Artificial Intelligence, (IJCAI)*, 2020.

Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

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.

Hannes Leitgeb. *The stability of belief: How rational belief coheres with probability*. Oxford University Press, 2017.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *arXiv*, 2018a.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18:1–52, 2018b.

Longfei Li, Yong Zhao, Dongmei Jiang, Yanning Zhang, Fengna Wang, Isabel Gonzalez, Enescu

Valentin, and Hichem Sahli. Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2013.

Wei Li, Peter Van Beek, and Pascal Poupart. Performing incremental bayesian inference by dynamic model counting. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2006.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2017.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Association for Computational Linguistics (ACL)*, 2017.

Baoding Lin. *Uncertainty theory: An introduction to its axiomatic foundations*. Springer-Verlag, 2004.

Thomas Lipp and Stephen Boyd. Variations and extensions of the convex-concave procedure. *Optimization and Engineering*, 17:263–287, 2016.

Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis (MedIA)*, 42:60–88, 2017.

Juntao Liu, Caihua Wu, and Wenyu Liu. Bayesian probabilistic matrix factorization with social relations and item contents for recommendation. *Decision Support Systems*, 55(3):838–850, 2013.

Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision (IJCV)*, 128(2):261–318, 2020.

Shulin Liu, Kang Liu, Shizhu He, and Jun Zhao. A probabilistic soft logic based approach to exploiting latent and global information in event classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

Daniel James Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, 2008.

John W Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Germany, 1987.

Ben London, Bert Huang, Benjamin Taskar, and Lise Getoor. Pac-bayes generalization bounds for randomized structured prediction. In *NeurIPS Workshop on Perturbation, Optimization and Statistics*, 2013a.

Ben London, Sameh Khamis, Stephen H. Bach, Bert Huang, Lise Getoor, and Larry Davis. Collective activity detection using hinge-loss markov random fields. In *CVPR SPTLI*, 2013b.

Ben London, Bert Huang, Benjamin Taskar, and Lise Getoor. Pac-bayesian collective stability. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, 2014.

Daniel Lowd and Pedro Domingos. Efficient weight learning for markov logic networks. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 200–211, 2007.

Stefan Lüdtke, Max Schröder, Sebastian Bader, Kristian Kersting, and Thomas Kirste. Lifted filtering via exchangeable decomposition. In *ECAI*, 2018.

Stefan Lüdtke, Alejandro Molina, Kristian Kersting, and Thomas Kirste. Gaussian lifted marginal filtering. In *KI: Advances in Artificial Intelligence*, pages 230–243. Springer, 2019.

Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *ACM International Conference on Web Search and Data Mining (WSDM)*, Hong Kong, 2011. ACM.

Sara Magliacane, Philip Stutz, Paul Groth, and Abraham Bernstein. foxpsl: A fast, optimized and extended psl implementation. *IJAR*, 67:111 – 121, 2015. doi: https://doi.org/10.1016/j.ijar.2015.05.012.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research (JMLR)*, 12(1):19–60, 2010.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021a.

Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate inference for neural probabilistic logic programming. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2021b.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

George Marsaglia. Choosing a point from the surface of a sphere. *Ann. Math. Statist.*, 43(2): 645–646, 1972.

Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José A. Castellanos, and Arnaud Doucet. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *AR*, 27(2):93–103, 2009.

Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional

data sets with application to reference matching. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 169–178. ACM, 2000. doi: 10.1145/347090.347123.

Wannes Meert and Joost Vennekens. Inhibited effects in cp-logic. In *European Workshop on Probabilistic Graphical Models*, 2014.

Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime Carbonell. Towards semi-supervised learning for deep semantic role labeling. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *arXiv*, abs/2001.05566, 2020.

Martin Mladenov, Babak Ahmadi, and Kristian Kersting. Lifted linear programming. In *AIStats*, 2012.

Martin Mladenov, Amir Globerson, and Kristian Kersting. Lifted message passing as reparametrization of graphical models. In *UAI*, 2014a.

Martin Mladenov, Kristian Kersting, and Amir Globerson. Efficient lifting of MAP LP relaxations using k-locality. In *AIStats*, 2014b.

Martin Mladenov, Leonard Kleinhans, and Kristian Kersting. Lifted inference for convex quadratic programs. In *AAAI*, 2017.

Jonas Mockus. On Bayesian Methods for Seeking the Extremum. In *IFIP Technical Conference*, 1974.

Jonas Mockus. On bayesian methods for seeking the extremum and their application. In *IFIP Congress*, 1977.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. In *TGO*, 1978.

Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *JLP*, 19-20:629 – 679, 1994.

Mervin E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, 1959.

Kevin Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

Galileo Mark Namata, Brian Staats, Lise Getoor, and Ben Shneiderman. A dual-view approach to interactive network visualization. In *ACM Conference on Information and Knowledge Management (CIKM)*, Lisbon, Portugal, 2007. ACM.

Yatin Nandwani, Abhishek Pathak, and Parag Singla. A primal dual formulation for deep learning with constraints. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Yatin Nandwani, Deepanshu Jindal, and Parag Singla. Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. *arXiv*, 2020.

Aniruddh Nath and Pedro Domingos. Efficient belief propagation for utility maximization and repeated inference. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010a.

Aniruddh Nath and Pedro Domingos. Efficient lifting for online probabilistic inference. In *International Workshop on Statistical Relational AI (StarAI)*, 2010b.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996.

Howard B. Newcombe and James M. Kennedy. Record linkage: Making maximum use of the discriminating power of identifying information. *ACM*, 5:563–566, 1962. doi: 10.1145/368996.369026.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Tien T Nguyen, Daniel Kluver, Ting-Yu Wang, Pik-Mai Hui, Michael D Ekstrand, Martijn C Willemsen, and John Riedl. Rating support interfaces to improve user experience and recommender accuracy. In *ACM Conference on Recommender Systems (RecSys)*, Hong Kong, 2013. ACM.

Nils J Nilsson. Probabilistic logic. *Artificial intelligence (AI)*, 28(1):71–87, 1986.

Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference

in markov logic networks using an rdbms. *International Conference on Very Large Data Bases (VLDB)*, 4:373–384, 2011.

Uri Nodelman, Christian Shelton, and Daphne Koller. Continuous time bayesian networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2002.

Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *International Workshop on Statistical Relational AI (StarAI)*, pages 37–42. AAAI Press, 2013.

Sebastian Nowozin and Christoph H Lampert. *Structured Learning and Prediction in Computer Vision*. Now Publishers Inc., 2011.

Ingrid Nunes and Dietmar Jannach. A systematic review and taxonomy of explanations in decision support and recommender systems. *User Modeling and User-Adapted Interaction (UMUAI)*, 27 (3-5):393–444, 2017.

George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *International Conference on Very Large Data Bases (VLDB)*, 9:684–695, 2016.

Terence J. Parr and Russell W. Quong. Antlr: A predicated-ll (k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

Denis Parra, Peter Brusilovsky, and Christoph Trattner. See what you want to see: visual user-driven

approach for hybrid recommendation. In *Intelligent User Interfaces (IUI)*, Haifa, Israel, 2014. ACM.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR workshop*, 2017.

Emmanouil Platanios, Hoifung Poon, Tom M. Mitchell, and Eric J. Horvitz. Estimating accuracy from unlabeled data: A probabilistic logic approach. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 4361–4370. Curran Associates, Inc., 2017a.

Emmanouil Platanios, Hoifung Poon, Tom M. Mitchell, and Eric J. Horvitz. Estimating accuracy from unlabeled data: A probabilistic logic approach. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017b.

Emmanouil Antonios Platanios, Hoifung Poon, Tom M. Mitchell, and Eric Joel Horvitz. Estimating accuracy from unlabeled data: A probabilistic logic approach. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017c.

David Poole. First-order probabilistic inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2nd edition, 2017.

Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deter-

ministic dependencies. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 6, pages 458–463, 2006.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in Artificial Intelligence (UAI)*. IEEE, 2011.

Soujanya Poria, Erik Cambria, Devamanyu Hazarika, Navonil Majumder, Amir Zadeh, and Louis-Philippe Morency. Context-dependent sentiment analysis in user-generated videos. In *Association for Computational Linguistics (ACL)*, 2017.

Soujanya Poria, Navonil Majumder, Rada Mihalcea, and Eduard H. Hovy. Emotion recognition in conversation: Research challenges, datasets, and recent advances. *IEEE Access*, 7:100943–100953, 2019.

Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria E. Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, 51(5):92:1–92:36, 2019.

Connor Pryor, Charles Dickens*, Eriq Augustine*, Alon Albalak, William Yang Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. *arXiv*, 2022.

Jay Pujara. Probabilistic models for scalable knowledge graph construction. phd, University of Maryland, College Park, 2016.

Jay Pujara and Lise Getoor. Generic statistical relational entity resolution in knowledge graphs. In *International Workshop on Statistical Relational AI (StarAI)*. IJCAI, 2016.

Jay Pujara, Hui Miao, Lise Getoor, and William W. Cohen. Knowledge graph identification. In *International Semantic Web Conference (ISWC)*, 2013.

Jay Pujara, Ben London, and Lise Getoor. Budgeted online collective inference. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015a.

Jay Pujara, Ben London, Lise Getoor, and William Cohen. Online inference for knowledge graph construction. In *International Workshop on Statistical Relational AI (StarAI)*, 2015b.

Meng Qu and Jian Tang. Probabilistic logic neural networks for reasoning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Luc De Raedt and Kristian Kersting. Statistical relational learning. In *Encyclopedia of Machine Learning*. Springer, 2011.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, New York, NY, USA, 2011.

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *MLJ*, 62(1-2):107–136, 2006.

Sebastian Riedel. Improving the accuracy and efficiency of map inference for markov logic. In *Uncertainty in Artificial Intelligence (UAI)*, pages 468–475, 2008.

John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Aaron Rodden, Tarun Salh, Eriq Augustine, and Lise Getoor. Vmi-psl: Visual model inspector for probabilistic soft logic. In *ACM Conference on Recommender Systems (RecSys)*, Virtual, 2020. ACM. This is a demo track paper.

Marco Rospocher. An ontology-driven probabilistic soft logic approach to improve nlp entity annotation. In *International Semantic Web Conference (ISWC)*, 2018.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiV*, abs/1609.04747, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Mehdi Samadi, Partha Pratim Talukdar, Manuela M. Veloso, and Manuel Blum. Claimeval: Integrated and flexible framework for claim evaluation using credibility of sources. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Yatin P Saraiya. *Subtree-Elimination Algorithms in Deductive Databases*. Stanford University, Stanford, CA, USA, 1991.

Somdeb Sarkhel, Deepak Venugopal, Parag Singla, and Vibhav Gogate. Lifted MAP inference for Markov logic networks. In *AIStats*, 2014.

Somdeb Sarkhel, Parag Singla, and Vibhav Gogate. Fast lifted map inference via partitioning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

Somdeb Sarkhel, Deepak Venugopal, Tuan Anh Pham, Parag Singla, and Vibhav Gogate. Scalable training of Markov logic networks using approximate counting. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Masahiro Sato, Budrul Ahsan, Koki Nagatani, Takashi Sonoda, Qian Zhang, and Tomoko Ohkuma. Explaining recommendations using contexts. In *Intelligent User Interfaces (IUI)*, Tokyo, Japan, 2018. ACM.

Taisuke Sato and Yoshitaka Kameya. PRISM: A symbolic-statistical modeling language. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

Julian J. Schlöder and Raquel Fernández. Clarifying intentions in dialogue: A corpus study. In *Computational Semantics (IWCS)*, 2015.

Joerg Schoenfisch and Heiner Stuckenschmidt. Analyzing real-world sparql queries and ontology-based data access in the context of probabilistic data. *International Journal of Approximate Reasoning (IJAR)*, 90:374–388, 2017.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Timos K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13 (1):23–52, 1988.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29:93–106, 2008.

Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. In *UAI*, 2009.

Luciano Serafini and Artur S. d'Avila Garcez. Learning and reasoning with logic tensor networks. In *Italian Association for Artificial Intelligence (AI*IA)*, 2016.

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.

Hossam Sharara, Awalin Sopan, Galileo Namata, Lise Getoor, and Lisa Singh. G-pare: A visual analytic tool for comparative analysis of uncertain graphs. In *IEEE Conference on Visual Analytics Science and Technology (VAST)*, Providence, RI, USA, 2011. IEEE.

Jude Shavlik and Sriraam Natarajan. Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009a.

Jude Shavlik and Sriraam Natarajan. Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009b.

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.

Parag Singla and Pedro Domingos. Discriminative training of markov logic networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 868–873, 2005.

Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 488–493, 2006.

Parag Singla and Pedro M. Domingos. Lifted first-order belief propagation. In *AAAI*, 2008.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

Sirawit Sopchoke, Ken ichi Fukui, and Masayuki Numao. Ilp recommender system: Explainable and more. In *International Conference on Inductive Logic Programming (ILP)*, Plovdiv, Bulgaria, 2019. White Rose.

Dhanya Sridhar, James Foulds, Marilyn Walker, Bert Huang, and Lise Getoor. Joint models of disagreement and stance in online debate. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Beijing, China, 2015. ACL.

Dhanya Sridhar, Shobeir Fakhraei, and Lise Getoor. A probabilistic approach for collective similarity-based drug-drug interaction prediction. *Bioinformatics*, 32(20):3175–3182, 2016.

Dhanya Sridhar, Jay Pujara, and Lise Getoor. Scalable probabilistic causal structure discovery. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimiza-

tion in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. information theory. *IEEE Transactions on*, 2012.

Sriram Srinivasan, Behrouz Babaki, Golnoosh Farnadi, and Lise Getoor. Lifted hinge-loss markov random fields. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019a.

Sriram Srinivasan, Nikhil S Rao, Karthik Subbaian, and Lise Getoor. Identifying facet mismatches in search via micrographs. In *CIKM*, 2019b.

Sriram Srinivasan, Eriq Augustine, and Lise Getoor. Tandem inference: An out-of-core streaming algorithm for very large-scale relational inference. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020a.

Sriram Srinivasan, Golnoosh Farnadi, and Lise Getoor. Bowl: Bayesian optimization for weight learning in probabilistic soft logic. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020b.

Sriram Srinivasan, Charles Dickens, Eriq Augustine, Golnoosh Farnadi, and Lise Getoor. A taxonomy of weight learning methods for statistical relational learning. *Machine Learning*, 2021.

Özgür Sümer, Umut Acar, Alexander Ihler, and Ramgopal Mettu. Adaptive exact inference in graphical models. *Journal of Machine Learning Research (JMLR)*, 12:3147–3186, 2011.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *FTML*, 4(4): 267–373, 2012.

Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Moviexplain: a recommender system with explanations. In *ACM Conference on Recommender Systems (RecSys)*, New York, NY, USA, 2009. ACM.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna Estrach, Dumitru Erhan, Ian Goodfellow, and Robert Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2003.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2004.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *International Conference on Machine Learning (ICML)*, 2005.

Benjamin Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *UAI*, 2002.

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.

Jason Ting. Visualizing the uncertainties of probabilistic soft logic with a graphical model, 2019.

Sabina Tomkins, Jay Pujara, and Lise Getoor. Disambiguating energy disaggregation: A collective probabilistic approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Sabina Tomkins, Golnoosh Farnadi, Brian Amanatullah, Lise Getoor, and Steven Minton. The impact of environmental stressors on human trafficking. In *IEEE International Conference on Data Mining (ICDM)*, pages 507–516, 2018a.

Sabina Tomkins, Lise Getoor, Yunfei Chen, and Yi Zhang. A socio-linguistic model for cyberbullying detection. In *ASONAM*, 2018b.

Sabina Tomkins, Steve Isley, Ben London, and Lise Getoor. Sustainability at scale: Bridging the intention-behavior gap with sustainable recommendations. In *ACM Conference on Recommender Systems (RecSys)*, Vancouver, BC, Canada, 2018c. ACM.

Efthymia Tsamoura, Víctor Gutiérrez-Basulto, and Angelika Kimmig. Beyond the grounding bottleneck: Datalog techniques for inference in probabilistic logic programs. In *AAAI Conference on Artificial Intelligence (AAAI)*, New York City, NY, USA, 2020. AAAI.

Kosetsu Tsukuda and Masataka Goto. Dualdiv: diversifying items and explanation styles in explainable hybrid recommendation. In *ACM Conference on Recommender Systems (RecSys)*, Copenhagen, Denmark, 2019. ACM.

Guy Van den Broeck, Kristian Kersting, Sriraam Natarajan, and David Poole. *An Introduction to Lifted Probabilistic Inference*. MIT Press, Cambridge, Massachusetts, 2021.

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. CP-Logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009.

D Venugopal, S Sarkhel, and V Gogate. Magician: Scalable inference and learning in markov logic using approximate symmetries. Technical report, UofM, Memphis, 2016.

Deepak Venugopal and Vibhav Gogate. On lifting the gibbs sampling algorithm. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

Deepak Venugopal and Vibhav Gogate. Evidence-based clustering for scalable inference in markov logic. In *ECML*, 2014.

Katrien Verbert, Denis Parra, Peter Brusilovsky, and Erik Duval. Visualizing recommendations to support exploration, transparency and controllability. In *Intelligent User Interfaces (IUI)*, Santa Monica, CA, USA, 2013. ACM.

Martin Wainwright and Michael Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning (FTML)*, 1(1 - 2):1–305, 2008.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning (ICML)*, 2019.

Wei-Chung Wang and Lun-Wei Ku. Identifying chinese lexical inference using probabilistic soft logic. In *ASONAM*, 2016a.

Wei-Chung Wang and Lun-Wei Ku. Identifying chinese lexical inference using probabilistic soft logic. In *ASONAM*, pages 737–743. IEEE Press, 2016b.

Wei-Chung Wang and Lun-Wei Ku. Enabling transitivity for lexical inference on chinese verbs using probabilistic soft logic. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2017.

William Yang Wang, Kathryn Mazaitis, Ni Lao, and William W Cohen. Efficient inference and learning in a large knowledge base. *ML*, 100:101–126, 2015.

Yan Wang, Jiayu Zhang, Jun Ma, Shaojun Wang, and Jing Xiao. Contextualized emotion recognition in conversation as sequence tagging. In *Special Interest Group on Discourse and Dialogue (SIGdial)*, 2020.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. Bayesian optimization in a billion dimensions via random embeddings. *JAIR*, 55(1):361–387, 2016.

Robert West, Hristo S. Paskov, Jure Leskovec, and Christopher Potts. Exploiting social network structure for person-to-person sentiment analysis. In *TACL*, 2014.

Ronald Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning (ICML)*, 2018.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Visual relationship detection with internal and external linguistic knowledge distillation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Alan L. Yuille and Anand Rangarajan. The convex-concave procedure. *Neural Computation*, 15(4): 915–936, 2003.

Qingchen Zhang, Laurence T. Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018a.

Yue Zhang, Arti Ramesh, Jennifer Golbeck, Dhanya Sridhar, and Lise Getoor. A structured approach to understanding recovery and relapse in aa. In *WWW*, 2018b.

Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020a.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *Transactions on Knowledge and Data Engineering (TKDE)*, 2020b.

Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Stephan Zheng, Yang Song, Thomas Leung, and Ian J. Goodfellow. Improving the robustness of deep neural networks via stability training. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Appendix A

Appendix

A.1 Collective Grounding

A.1.1 Grounding Pseudocode

Where $CreateBaseQuery$ is a function that takes in a single rule, and creates the rule's base query (as discussed in Section 3.2.1) as well as a mapping between the variables used in the rule and columns of the query; and $ExecuteQuery$ is a function that executes a query on a database and returns the results; and $InstantiateGroundRules$ takes in results from a database query, a set of rules, and a mapping that maps variables used in the rules to columns of the database query; and returns ground rules created by instantiating each rule using the query results.

Where $GenerateCandidates$ is fully defined in Algorithm 2;
$SearchCandidates$ is a function that explores that candidate search space as described in Sec-

Algorithm 9: The independent grounding process. This process takes as input a collection of template, T and a database containing both observed data X and unobserved data Y, and outputs a set of ground rules generated by instantiating the rules using the database. Each rule is processed individually using the base grounding query for each rule.

```
1 Function IndependentGrounding:
      Input: rules = T, database = (X, Y)
      Result: groundRules = {}
2
3     foreach rule in rules do
4         (query, variableMap) ← CreateBaseQuery(rule)
5         results ← ExecuteQuery(database, query)
6         groundRules ← groundRules ∪
7             InstantiateGroundRules(results, {rule}, variableMap)
```

Algorithm 10: The collective grounding process. This process takes as input a collection of template, T and a database containing both observed data X and unobserved data Y, and outputs a set of ground rules generated by instantiating the rules using the database. Candidates are generated from each rule. Then, these candidates are examined together to fine the best grounding program that collectively minimizes the joint workload of grounding.

```
Function CollectiveGrounding:
    Hyperparameters: searchBudget, searchType
    Input: rules = T, database = (X, Y)
    Result: groundRules = {}

    candidates ← {}

    foreach rule in rules do
        ruleCandidates ← GenerateCandidates(rule)
        bestRuleCandidates ←
          SearchCandidates(ruleCandidates, searchBudget, searchType)
        candidates ← candidates ∪ bestRuleCandidates

    candidateRuleMapping ← CreateCandidateRuleMapping(candidates, rules)
    groundingProgram ← ConstructGroundingProgram(candidateRuleMapping)

    foreach (query, rules, variableMap) in groundingProgram do
        results ← ExecuteQuery(database, query)
        groundRules ← groundRules ∪
          InstantiateGroundRules(results, rules, variableMap)
```

tion 11; $CreateCandidateRuleMapping$ is a function that maps candidates to the rules they satisfy as described in Section 11; and $ConstructGroundingProgram$ is a function that computes grounding plans (as defined by Equation 2.3) using the procedure described in Section 11. Note that the $CollectiveGrounding$ function has the same parameter and return signature as the $IndependentGrounding$ function.

The pseudocode in this section for $CollectiveGrounding$ provides a synchronous implementation that favors clarity over performance. The implementation used in the experiments in Section 3.2.2 (and included in the accompanying code[1]) is an asynchronous version that interweaves candidate generation with the budgeted search to avoid expanding large search spaces.

A.1.2 Full LASTFM Model

Figure A.1 shows the full LASTFM model used in Kouki et al. [2015]. The weight was omitted for all rules as weights are learned on a per-split basis.

A.1.3 Full DDI Model

Figure A.2 shows the full DDI model used in Sridhar et al. [2016]. The weight was omitted for all rules as weights are learned on a per-split basis. Rules ending with a period are unweighted (hard constraints).

[1] Available at `https://github.com/eriq-augustine/srli`.

Similarities like Pearson, Cosine, and Adjusted Cosine Similarity between items.

Rated(U, I1) ∧ Rated(U, I2) ∧ Rating(U, I1) ∧ SimPearsonItems(I1, I2) → Rating(U, I2)

Rated(U, I1) ∧ Rated(U, I2) ∧ Rating(U, I1) ∧ SimCosineItems(I1, I2) → Rating(U, I2)

Rated(U, I1) ∧ Rated(U, I2) ∧ Rating(U, I1) ∧ SimAdjcosItems(I1, I2) → Rating(U, I2)

Similarities like Pearson and Cosine Similarity between users.

Rated(U1, I) ∧ Rated(U2, I) ∧ Rating(U1, I) ∧ SimPearsonUsers(U1, U2) → Rating(U2, I)

Rated(U1, I) ∧ Rated(U2, I) ∧ Rating(U1, I) ∧ SimCosineUsers(U1, U2) → Rating(U2, I)

Other low dimension space similarities like Matrix Factorization Cosine and Euclidean Similarity between users.

User(U1) ∧ User(U2) ∧ Item(I) ∧ Rating(U1, I) ∧ Rated(U1, I) ∧ Rated(U2, I) ∧ SimMFCosineUsers(U1, U2) → Rating(U2, I)

User(U1) ∧ User(U2) ∧ Item(I) ∧ Rating(U1, I) ∧ Rated(U1, I) ∧ Rated(U2, I) ∧ SimMFEuclideanUsers(U1, U2) → Rating(U2, I)

Other low dimension space similarities like Matrix Factorization Cosine and Euclidean Similarity between items.

User(U) ∧ Item(I1) ∧ Item(I2) ∧ Rating(U, I1) ∧ Rated(U, I1) ∧ Rated(U, I2) ∧ SimMFCosineItems(I1, I2) → Rating(U, I2)

User(U) ∧ Item(I1) ∧ Item(I2) ∧ Rating(U, I1) ∧ Rated(U, I1) ∧ Rated(U, I2) ∧ SimMFEuclideanItems(I1, I2) → Rating(U, I2)

Predictions by different other methods like SGD, Item based Pearson methods, and BPMF methods.

SGDRating(U, I) → Rating(U, I)

Rating(U, I) → SGDRating(U, I)

ItemPearsonRating(U, I) → Rating(U, I)

Rating(U, I) → ItemPearsonRating(U, I)

BPMFRating(U, I) → Rating(U, I)

Rating(U, I) → BPMFRating(U, I)

Average prior of User Rating and Item ratings.

User(U) ∧ Item(I) ∧ Rated(U, I) ∧ AvgUserRating(U) → Rating(U, I)

User(U) ∧ Item(I) ∧ Rated(U, I) ∧ Rating(U, I) → AvgUserRating(U)

User(U) ∧ Item(I) ∧ Rated(U, I) ∧ AvgItemRating(I) → Rating(U, I)

User(U) ∧ Item(I) ∧ Rated(U, I) ∧ Rating(U, I) → AvgItemRating(I)

Social rule of friendship influencing ratings.

Rated(U1, I) ∧ Rated(U2, I) ∧ Friends(U1, U2) ∧ Rating(U1, I) → Rating(U2, I)

Content rule by Jaccard similarity.

Rated(U, I1) ∧ Rated(U, I2) ∧ Rating(U, I1) ∧ SimContentItemsJaccard(I1, I2) → Rating(U, I2)

Figure A.1: The full LastFM model used in Kouki et al. [2015].

```
# Similarity based rules.
ATCSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
SideEffectSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
GOSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
ligandSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
chemicalSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
seqSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
distSimilarity(D1, D2) ∧ interacts(D1, D3) ∧ validInteraction(D1, D3) ∧ validInteraction(D2, D3) → interacts(D2, D3)^2
# Symmetry.
interacts(D1, D2) = interacts(D2, D1).
# Negative prior.
validInteraction(D1, D2) → ¬interacts(D1, D2)^2
```

Figure A.2: The full DDI model used in Sridhar et al. [2016].

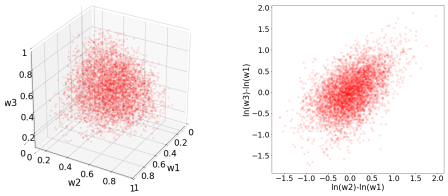
A.2 Weight Learning Sampling

A.2.1 Surface of a Unit Hypersphere as Search Space

Here, I discuss the effectiveness of the surface of a unit hypersphere as the search space in weight learning. To show that this is a reasonable space to sample weights from I need to show: first, the surface of the hypersphere is complete and non-redundant and second, if two weight configurations in OS are represented by the same weight configurations on the hypersphere then the solution obtained for $\mathbf{y}$ by both the weight configurations are the same. Every weight configuration given in OS can be easily projected on to the hypersphere in the following way:

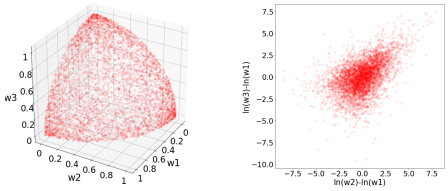
$$\mathcal{H}(\mathbf{w}) = \frac{\mathbf{w}}{||\mathbf{w}||_2} \tag{A.1}$$

$\mathrm{A} = 10, 10, 10$



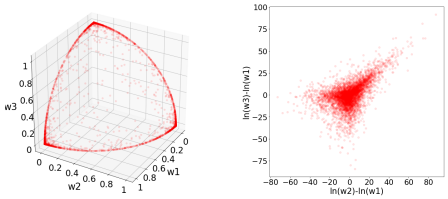
(a) In OS (b) In SS

$\mathrm{A} = 1, 1, 1$



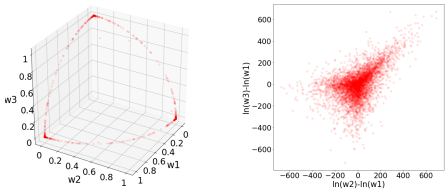
(c) In OS (d) In SS

$\mathrm{A} = 0.1, 0.1, 0.1$



(e) In OS (f) In SS

$\mathrm{A} = 0.01, 0.01, 0.01$



(g) In OS (h) In SS

Figure A.3: Visualization of Dirichlet distribution with different A of a model with three rules. Visualization shown both in OS and SS.

where $||\mathbf{w}||_2$ is the L-2 norm of the vector.

Theorem 9. *Every weight configuration* $\mathbf{w}$ *in OS has an equivalent weight configuration* $\tilde{\mathbf{w}}$ *on the hypersphere such that* $\arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \tilde{\mathbf{w}})$.

Proof. From Theorem 2 I know that weights are scale invariant when performing MAP inference and this implies that the magnitude of the vector generated by a weight configuration does not affect the solution obtained when performing inference in PSL and MLNs. This implies that all r-dimensional unit vectors in the positive quadrant is sufficient to represent all possible weight configuration for MAP inference in PSL and MLNs. All unit vectors can be represented using the surface of a hypersphere. Therefore, surface of an r-dimensional hypersphere removes redundancies of OS and is complete. □

Theorem 10. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *such that the projection* $\mathcal{H}$ *of the weight configurations are equal, i.e.,* $\mathcal{H}(\mathbf{w}_1) = \mathcal{H}(\mathbf{w}_2)$*, then the optimization solution obtained for* $\mathbf{y}$ *by performing inference is the same.*

Proof. This is easy to show as the definition of the projection defined in Equation A.1 is very similar to the projection defined in Theorem 2. The weights are simply rescaled and therefore, by definition and from Theorem 2, it is clear to see that if $\mathcal{H}(\mathbf{w}_1) = \mathcal{H}(\mathbf{w}_2)$ then $\arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_1) = \arg\max_{\mathbf{y}} \hat{s} \cdot \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2)$. □

This shows that the projection on to the surface of a unit hypersphere has properties similar to SS. However, these two are not entirely equivalent as the grounding factor κ plays an

important role in the actual impact of weights at the time of inference. While SS is ideal and distances between weights are preserved even after adjusting for grounding, the distances are not preserved after adjusting for grounding.

Theorem 11. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *and their grounding adjusted weights* $\kappa \cdot \mathbf{w}_1$ *and* $\kappa \cdot \mathbf{w}_2$ *(an element-wise dot), the distance between the two weights before and after grounding adjustment are not the same, i.e.,* $||\mathbf{w}_1 - \mathbf{w}_2|| \neq \kappa \cdot \mathbf{w}_1 - \kappa \cdot \mathbf{w}_2$.

Therefore, I can conclude that while the surface of a hypersphere does not have all the properties of SS it is a reasonable approximation of SS and weights can be samples from the hypersphere.

Density of Samples Generated by Projecting Points from Probability Simplex

While sampling from a hypersphere with different densities can be tricky one easy way to do this is to sample from a Dirichlet distribution which samples from the probability simplex and project the values on to the sphere. The Dirichlet distribution accepts the hyperparameter $\mathrm{A} \in \mathbb{R}^{+r}$ which controls the spread of the distribution. It is easy to see that every point on a probability simplex can be uniquely projected on to a hypershpere. Here in Figure A.3 I visualize the distribution generated by using different values of A for a model with three rules. I vary the value of alpha form all 10s to all 0.001 and plot the projection on to a sphere which is in OS on the left and SS (which is 2-dimensional space) on the right. I can observe that as the value of A is reduced the samples in OS are getting concentrated to the poles and in SS the samples spread wider

and choosing larger ratios. I can choose this value based on my application. For instance when the task is to perform rule pruning, it is more desirable for the ratio of weights to be more extreme and hence choose a small value for A and if the task is to fine tuning weights without making large changes, then a higher value of A might be more suitable.