DEEP LEARNING WITH OUTPUT SPACE CONSTRAINTS

YATIN NANDWANI

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI

APRIL 2024

©Indian Institute of Technology Delhi - 2024
All rights reserved.

DEEP LEARNING WITH OUTPUT SPACE CONSTRAINTS

by

YATIN NANDWANI

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of Doctor of Philosophy

to the

INDIAN INSTITUTE OF TECHNOLOGY DELHI

APRIL 2024

Certificate

This is to certify that the thesis titled **Deep Learning with Output Space Constraints** being submitted by **Mr Yatin Nandwani** for the award of **Doctor of Philosophy** in Department of Computer Science and Engineering is a record of bonafide work carried out by him under my guidance and supervision at the **Department of Computer Science and Engineering, Indian Institute of Technology Delhi**. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma unless otherwise stated explicitly. In particular, work done in chapter 3, chapter 4, chapter 5, chapter 6 and chapter 7 were done jointly with undergraduate students. In each case, the part done by the collaborators appeared in their respective bachelor's thesis.

Mausam
Professor
Department of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi- 110016

Parag Singla
Professor
Department of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi- 110016

Acknowledgements

I decided to leave my comfortable job and challenge myself by enrolling in the PhD program at the age of thirty-two. With two kids and so much responsibility, it was never an easy decision, filled with uncertainty, doubts and doubters! Yet, amidst the sceptics and the challenges, one unwavering presence stood by my side like a steadfast pillar: my beloved wife, Shagun. It is to her unwavering support that I dedicate this thesis, for without her, the very thought of seeking this academic pursuit at this stage of life would have been inconceivable.

With the responsibility of two kids, *Ziven* and *Zaina*, it could have been difficult for both Shagun and me to handle it all alone, especially during the initial years when Zaina was just an infant. Thanks to the invaluable assistance of our parents, sisters, and brothers-in-law, who played pivotal roles in tending to our children, I could get the needed time and focus for my research and coursework. During COVID, I often went to Shagun's sister's home, or sometimes my own sister's home, to study. This provided a much-needed break and a welcome change of place.

Despite the added responsibility that comes with children, witnessing their growth is a source of constant joy. Engaging in playtime and spending quality moments with them often served as a welcome distraction from my studies. Their presence proved to be a fantastic equalizer and an effective stress reliever.

This journey wouldn't have been as incredible as it turned out to be without my supervisors, *Mausam* and *Parag*. They are truly inspiring, excelling not only in their roles as researchers and educators but also as people deeply committed to their profession and society. Their strong sense of duty towards the academic community and humanity sets a remarkable example for all. Apart from our research endeavours, I cherished engaging in philosophical conversations with both of them on various topics, ranging from our societal responsibilities to the meaning of life and beyond.

I had the privilege of serving as one of the workflow chairs for AAAI'21 when Mausam was the program co-chair. His dedication and resilience were truly remarkable; even while battling COVID in the hospital, he had conference-related work in the back of his mind, displaying an unparalleled commitment to his responsibilities. This was the first time I

saw him not just as an advisor but also as a dedicated researcher. Besides this, I worked very closely with him to edit the video content for the NPTEL course on AI. This is when I realized that a lot of effort goes behind the scenes to cater to such a big course! We jointly presented a tutorial on neuro-symbolic AI at a workshop. The flow of the tutorial significantly influenced the narrative and the story that connects the different works presented in my thesis!

I thoroughly enjoyed attending Parag's lectures, particularly his course on Deep Learning, which was the first course I attended. He teaches with a lot of patience, and I can't imagine anyone else delivering the material better than him. Initially, our discussions revolved around research topics. I later learned about the A.I.N.A. (An Initiative for National Advancement) initiative that Parag has been running since 2012. This led to an opportunity to join one of the trips to Uttarakhand, where many volunteer students participated, accompanied by both Parag and me, along with our families. During this trip, I got to engage in philosophical discussions with Parag, which added depth to our interactions. Witnessing the dedication of the students involved in the initiative was truly inspiring and served as a motivating force for me.

Besides Mausam and Parag, I enjoyed my interactions with Prathosh and Rohan. Deep Learning was the first course that Prathosh taught at IIT Delhi, and I enjoyed the couple of discussions I had with him about the course content. I was one of the TAs for Rohan's AI course, and working closely with such a humble and down-to-earth person was a wonderful experience.

At NeurIPS'18, my very first international conference, I had the chance to interact with many researchers. One name that stands out is Alexender Rush (Shasha). My interactions with him proved crucial for my first work, which eventually got published at NeurIPS'19. His tutorial on dual decomposition and Lagrangian relaxation in NLP [Rush and Collins, 2012] provided a foundational understanding that laid the groundwork for my research.

My first publication at NeurIPS'19 introduced me to the research community focusing on deep learning with constraints. I had the opportunity to connect with Maneesh Singh and Vivek Srikumar, whose work overlapped with mine. This fruitful connection led to a joint internship with Vivek and Maneesh at Verisk, where I tackled the problem of identifying temporal relationships between events in documents. While the internship transitioned to a work-from-home format due to COVID, the experience was invaluable.

During my stint as AAAI'21 workflow chair, I got an opportunity to work closely with Prof. Kevin Leyton-Brown (he was the co-chair along with Mausam) and his students at UBC - Heddy, Chris, Neil. We ended up publishing our findings, and I remember writing the first draft of the technical section. One night before the deadline, the paper was in bad shape, and I was almost sure that we wouldn't be able to submit it. Kevin started

polishing the writing (actually, rewrote most of it) just 24 hours before the deadline, and the transformation of the paper was unbelievable! Thanks, Kevin, for the wonderful exposure!

One person who warrants a special mention here is Ankit Anand. I published my first paper with him, attended my first conference with him, and stayed at his home in Montreal countless times! Thanks, Ankit and Pallavi, for hosting me with smiles! What a nice way of giving back for the home food that we sometimes shared together ;-).

I shared most of my time in the lab with Keshav, who happens to be ten years younger than me! I always felt younger in his company! Thanks, Keshav, for the wonderful camaraderie we shared.

Before COVID, we used to play cricket on campus, and on many days, it was the primary motivation to come to college! Thanks to all the players who made it happen, especially Happy, who would happily give me a lift back home!

As we prepared for our job interviews, Prachi, Keshav, Arnab, Vishal, Kunal, and I decided to form a reading group. It turned out to be a fantastic decision! We met weekly, and each person took turns presenting a topic. Each of us had expertise in different areas – Kunal in extreme multi-label learning, Prachi and Keshav in NLP, Vishal in RL, and Arnab in VAEs. Learning from each other's expert knowledge turned our preparation into an exciting journey of learning. Thanks to everyone's enthusiasm and insights, our sessions were not only informative but also a lot of fun. Thanks, guys, for sharing your invaluable insights and making the process rewarding and enjoyable! I hope that our friendship continues and we stay in touch!

While working for AAAI'21, I formed a close collaboration with Dinesh Raghu. Little did I know that this partnership would extend far beyond the conference, with Dinesh eventually playing a pivotal role in getting me a position at the IBM Research Lab (IRL). Thanks, Dinesh, for an amazing collaboration! The interactions with my manager, Sachin, and other teammates at IRL have been incredibly stimulating. Thanks, Sachin, for offering me the opportunity and for also granting me the necessary time to complete my thesis! Special thanks to Gaurav, Mayank, and Kushal for discussing and helping with the additional experiment I did to incorporate one of the thesis reviewer's suggestions.

To write the abstract of this thesis in Hindi, I first used GPT-4. Sadly, it didn't do a good job. Fortunately, one of my uncles, Prof. Harish Sethi, who specializes in translating technical documents to Hindi, visited us on a Sunday and helped me with the translation. Thank you, uncle, for your timely help!

Finally, I would like to thank all my collaborators – Bhavesh, Abhishek, Aman, Ankesh, Mayank, Deepanshu, Vidit, and Rishabh. I spent a lot of time brainstorming, arguing, and discussing various topics with them. Thanks, guys, for the many sleepless nights close

to the conference deadlines! I wish you all the success and happiness in life. With your support and company, my PhD journey became really enjoyable. I hope that our paths will cross again!

In the end, I would once again like to thank –

Parag and Mausam, who guided me on my academic journey ...

Zaina, whose birth marked the journey of my re-discovery ...

Ziven, whose birth marked the journey of my parenthood ...

Shagun, who marked the journey of my partnership ...

My spiritual masters, who marked my inner journey of self-discovery ...

My parents, who marked my outer journey in this world ...

Abstract

Since the emergence of Deep Learning, the field of Artificial Intelligence has experienced remarkable progress across multiple domains, including Computer Vision and Natural Language Processing. Neural Networks have consistently achieved state-of-the-art performance across various tasks within these fields. The output of these neural networks is often symbolic, and there may exist known or unknown constraints between these symbols. These constraints may provide a natural way of representing domain knowledge related to the underlying task. We can categorize these constraints based on various dimensions, such as their representation (explicit vs implicit), knowledge (known vs unknown), hard vs soft constraints *etc.*. We use the first two dimensions of 'constraint representation' and 'knowledge of constraints' to categorize our work into three broad buckets of "explicitly known constraints", "unknown constraints with implicit representation", and "unknown constraints with explicit representation".

In the first part, 'Explicitly Known Constraints', we explore if neural models can be better trained using known symbolic domain knowledge expressed as hard–constraints on the input/output variables. When constraints are hard, most of the existing works focus on constrained inference [Rush and Collins, 2012, Ning et al., 2019]. Few works do exploit constraints during training [Hu et al., 2016a, Xu et al., 2018], however, their formulation is capable of handling only soft constraints. In contrast, we propose a primal-dual formulation that can potentially deal with hard constraints [Nandwani et al., 2019].

In the next two parts, we focus on automated learning of unknown constraints from the data. In many scenarios, it is difficult to encode our intuition explicitly in the form of a constraint, or we may miss certain constraints while enumerating them, warranting the need for their automated learning. Here we focus on combinatorial problems, such as sudoku, that involve satisfaction of the underlying hard constraints defining the problem. Based on the representation of constraints, we categorize existing approaches for learning of unknown constraints into two categories: *implicit* and *explicit*. Works in the former category propose purely neural models [Palm et al., 2018, Dong et al., 2019] for solving these tasks, representing the underlying constraints implicitly in their weights. In the second part of the thesis, 'Unknown Constraints with Implicit Representation', we identify

a couple of potential issues in such models and propose appropriate solutions. First, we identify the issue of solution multiplicity (one input having many correct solutions) while training neural models and propose appropriate loss functions and an RL based technique to optimize it[Nandwani et al., 2021]. Next, we observe that existing architectures, such as SATNet [Wang et al., 2019], Relational Recurrent Networks (RRN) [Palm et al., 2018], fail to generalize across the output space of variables (can not solve 16 x 16 sudoku after training on only 9 x 9 sudoku). In response, we design two neural architectures for output space invariance in combinatorial problems [Nandwani et al., 2022a].

The third part of the thesis, 'Explicit and Unknown Constraints', focuses on the *explicit* representation of constraints, *e.g.*, linear inequalities over output variables as in an Integer Linear Program (ILP) that can express many combinatorial problems. One shortcoming of the existing approaches [Paulus et al., 2021, Wilder et al., 2019] is that they need to solve the ILP in every learning iteration, thereby considerably slowing down the training process. In the third part, we first propose a solver-free framework for scalable learning of constraints in an ILP [Nandwani et al., 2022b]. Finally, we take a completely different view of constraints as an explanation for the output of a black-box neural model. Specifically, we focus on explaining the output of Tensor–Factorization based models for the task of Knowledge Base Completion. We propose a post-hoc explanation engine that produces constraints in the form of horn clauses as an explanation. These constraints may not always hold for all the samples in the data, thereby making them 'soft', unlike all our previous works where it is not allowed to violate the constraints, making them 'hard'.

In the end, we conclude by summarizing our contributions and proposing directions for future work that builds on the techniques and methods introduced in this work.

सार

डीप लर्निंग के उद्भव और विकास के कारण आर्टिफिशियल इंटेलिजेंस के कंप्यूटर विजन और प्राकृतिक भाषा संसाधन (नेचुरल लैंग्वेज प्रोसेसिंग) जैसे विविध क्षेत्रों में उल्लेखनीय प्रगति देखी गई है। न्यूरल नेटवर्क्स ने इन क्षेत्रों के भीतर विभिन्न कार्यों में लगातार अत्याधुनिक (स्टेट–ऑफ–द–आर्ट) प्रगति की है। इन न्यूरल नेटवर्क्स का आउटपुट अक्सर प्रतीकात्मक होता है, और इन प्रतीकों के बीच ज्ञात या अज्ञात कॉन्सट्रेंट्स हो सकते हैं। ये कंस्ट्रेन्ट्स अंतर्निहित कार्य से संबंधित क्षेत्र–विशेष के ज्ञान को दर्शाने का एक स्वाभाविक तरीका प्रदान कर सकते हैं। हम इन कंस्ट्रेन्ट्स को विभिन्न आयामों के आधार पर वर्गीकृत कर सकते हैं। जैसे, उनका रिप्रेजेंटेशन (एक्सप्लिसिट बनाम इम्प्लिसिट), ज्ञान (ज्ञात बनाम अज्ञात), हार्ड बनाम सॉफ्ट कंस्ट्रेन्ट्स आदि। इस शोध अध्ययन को पहले दो आयामों – 'कंस्ट्रेन्ट रिप्रेजेंटेशन' और 'कंस्ट्रेन्ट्स की नॉलेज' – का उपयोग करके तीन मुख्य वर्गों में विभाजित किया गया है। ये हैं – "एक्सप्लिसिटली ज्ञात कंस्ट्रेन्ट्स", "अज्ञात कंस्ट्रेन्ट्स विद इम्प्लिसिट रिप्रेजेंटेशन"; और "अज्ञात कंस्ट्रेन्ट्स विद एक्सप्लिसिट रिप्रेजेंटेशन"।

पहले भाग, अर्थात "एक्सप्लिसिटली ज्ञात कंस्ट्रेन्ट्स" में, हम यह पता लगाते हैं कि क्या न्यूरल मॉडल्स को ज्ञात सिंबोलिक डोमेन नॉलेज का उपयोग करके बेहतर तरीके से प्रशिक्षित किया जा सकता है? इस सिंबोलिक डोमेन नॉलेज को हम इनपुट/आउटपुट वैरिएबल्स पर हार्ड कंस्ट्रेन्ट्स के रूप में व्यक्त करते हैं। जब कंस्ट्रेन्ट्स हार्ड होते हैं, तब अधिकांशतः कार्य कंस्ट्रेन्ड इंफरेंस पर केंद्रित होते हैं। कुछ कार्य, प्रशिक्षण के दौरान कंस्ट्रेन्ट्स का लाभ उठाते हैं, हालांकि, उनका फॉर्मूलेशन केवल सॉफ्ट कंस्ट्रेन्ट्स का उपयोग करने में सक्षम है। इसके स्थान पर, हमने एक प्राइमल–डुअल फॉर्मूलेशन को प्रस्तावित किया है जो संभवतः प्रशिक्षण के दौरान हार्ड कंस्ट्रेन्ट्स को भली प्रकार से डील करने में सक्षम हो सकता है।

अगले दो भागों में, डेटा से अज्ञात कंस्ट्रेन्ट्स को मॉडल के द्वारा स्वतः सीखने पर ध्यान केंद्रित किया गया है। कई परिदृश्यों में, हमारी इंट्यूशन को कंस्ट्रेन्ट के रूप में स्पष्टतः एन्कोड करना कठिन होता है या हम उनका उल्लेख करते समय कुछ कंस्ट्रेन्ट्स भूल सकते हैं। इस कारण मॉडल को डेटा से कंस्ट्रेन्ट्स को स्वतः सीखने की आवश्यकता होती है। हम यहां सुडोकु जैसे कॉम्बिनेटोरियल प्रॉब्लम्स पर ध्यान केंद्रित करते हैं, जिसमें प्रॉब्लम को परिभाषित करने वाले हार्ड कंस्ट्रेन्ट्स को मॉडल के द्वारा सीखना होता है। कंस्ट्रेन्ट्स के रिप्रेजेंटेशन के आधार पर, अज्ञात कंस्ट्रेन्ट्स को सीखने के लिए हमने वर्तमान दृष्टिकोणों को दो श्रेणियों में वर्गीकृत किया है — इम्प्लिसिट और एक्सप्लिसिट। इम्प्लिसिट श्रेणी के काम, कॉम्बिनेटोरियल प्रॉब्लम्स को हल करने के लिए विशुद्ध रूप से न्यूरल मॉडल्स का प्रस्ताव देते हैं, जो अपने वेट्स में इम्प्लिसिट तरीके से कंस्ट्रेन्ट्स को रिप्रेजेंट करते हैं। शोध–प्रबंध के दूसरे भाग — 'अज्ञात कंस्ट्रेन्ट्स विद इम्प्लिसिट रिप्रेजेंटेशन'— में हमने इन मॉडलों में कुछ संभावित मुद्दों की पहचान की है और उनके समुचित समाधान प्रस्तावित किए हैं। इस प्रक्रिया में हमने सबसे पहले न्यूरल मॉडल्स को प्रशिक्षित करते समय 'सॉल्यूशन मल्टीप्लिसिटी' (एक इनपुट के कई सही आउटपुट होना) की समस्या की पहचान की है। फिर, इसे हल करने के लिए हमने एक उचित लॉस फंक्शन

निर्मित किया है और उसे ऑप्टिमाइज़ करने के लिए एक आर.एल. आधारित तकनीक को प्रस्तावित किया है। इसके बाद, हमने यह ध्यान दिया कि रिलेशनल रिकरेंट नेटवर्क्स (आर.आर.एन.) जैसे मौजूदा आर्किटेक्चर्स, आउटपुट स्पेस में जनरलाइज़ करने में सार्थक सिद्ध नहीं होते हैं (जैसे केवल 9×9 सुडोकु पर प्रशिक्षण के बाद 16×16 सुडोकु हल नहीं कर सकते)। इसलिए, इसके स्थान पर, हमने कॉम्बिनेटोरियल प्रॉब्लम्स में आउटपुट स्पेस इन्वेरियन्स के लिए दो न्यूरल आर्किटेक्चर्स को डिज़ाइन किया है।

शोध–प्रबंध का तीसरा भाग, 'एक्सप्लिसिट और अननोन कंस्ट्रेन्ट्स', कंस्ट्रेन्ट्स के एक्सप्लिसिट रिप्रेजें–टेशन पर केंद्रित है। उदाहरण के तौर पर, इंटीजर लीनियर प्रोग्राम (आई.एल.पी.) की लीनियर इनइक्वालिटीज़, जो कई कॉम्बिनेटोरियल प्रॉब्लम्स को व्यक्त कर सकती हैं। वर्तमान दृष्टिकोणों की एक कमी यह है कि उन्हें प्रशिक्षण की हर इटरेशन में आई.एल.पी. को हल करना पड़ता है, जिससे प्रशिक्षण प्रक्रिया में काफी समय लगता है। तीसरे भाग में, हमने आई.एल.पी. के कंस्ट्रेन्ट्स को स्केलेबल तरीके से सीखने के लिए एक सॉल्वर–फ्री फ्रेमवर्क को प्रस्तावित किया है। अंत में, हम कंस्ट्रेन्ट्स को एक नए नजरिए से देखते हैं जिसमें उनका प्रयोग ब्लैक–बॉक्स न्यूरल मॉडल के आउटपुट को समझाने के लिए किया जाता है। विशेष रूप से, हमने नॉलेज बेस कंप्लीशन के कार्य के लिए टेंसर–फैक्टराइजेशन आधारित मॉडल्स के आउटपुट को समझाने पर ध्यान केंद्रित किया है। हमने एक पोस्ट–हॉक एक्सप्लेनेशन इंजन को प्रस्तावित किया है, जो हॉर्न क्लॉजेस के रूप में कंस्ट्रेन्ट्स को एक्सप्लेनेशन के तौर पर उत्पन्न करता है। ध्यान देने योग्य बात यह है कि एक्सप्लेनेशन के लिए प्रयोग किए गए कंस्ट्रेन्ट्स, सभी डेटा सैंपल्स के लिए हमेशा मान्य नहीं हो सकते हैं। इसलिए वे सॉफ्ट कंस्ट्रेन्ट्स हैं, जबकि इस शोध अध्ययन के अन्य सभी कार्यों में हार्ड कंस्ट्रेन्ट्स का प्रयोग किया गया है।

अंत में, हमने अपने कार्य के रूप में किए गए योगदानों का उपसंहार प्रस्तुत किया है और इस दिशा में भविष्य में होने वाले उन शोध–कार्यों के लिए दिशा–निर्देश प्रस्तावित किए हैं, जो इस अध्ययन में प्रस्तुत तकनीकों और विधियों पर आधारित होंगे।

Contents

Certificate i

Acknowledgements iii

Abstract vii

Abstract in Hindi ix

List of Figures xviii

List of Tables xxi

I Prologue 1

1 Introduction 3

1.1 Constraints in Machine Learning 5

1.1.1 Benefits of Imposing Constraints 5

1.1.2 Variables in the Constraint Expression 7

1.1.3 Representation of Constraints 8

1.1.4 Known vs Unknown Constraints 9

1.1.5 Hard Vs Soft Constraints 9

1.1.6 Methods for Combining ML Models and Constraints 9

1.2 Contributions 11

1.2.1 Known Constraints 11

1.2.2 Unknown Constraints 13

1.2.3 Implicit Representation of Unknown Constraints 14

1.2.4 Explicit Representation of Unknown Constraints 17

1.3 Thesis Outline 19

2 Related Work and Background **21**
2.1 Explicitly Known Constraints . . . 21
2.1.1 Constrained Inference . . . 21
2.1.2 Constrained Training . . . 24
2.2 Unknown Constraints . . . 26
2.2.1 Neural Solvers with Implicit Constraint Representation . . . 27
2.2.2 Symbolic Solvers with Explicit Constraint Representation . . . 31

II Explicitly Known Constraints **33**

3 Learning with Known Constraints **35**
3.1 Introduction . . . 35
3.2 Related Work . . . 38
3.3 Constrained Learning of Neural Models . . . 39
3.3.1 A Lagrangian-based Formulation . . . 40
3.3.2 Constraint Language for Discrete Output Spaces . . . 41
3.4 Training . . . 42
3.5 Experiments . . . 44
3.5.1 Semantic Role Labeling (SRL) . . . 44
3.5.2 Named Entity Recognition (NER) . . . 46
3.5.3 Fine Grained Entity Typing . . . 48
3.6 Conclusion . . . 50

III Unknown Constraints with Implicit Representation **51**

4 1oML: One of Many Learning **53**
4.1 Introduction . . . 54
4.2 Background and Related Work . . . 56
4.3 Theory and Algorithm . . . 58
4.3.1 Problem Definition . . . 58
4.3.2 Objective Function . . . 59
4.3.3 Greedy Formulation: MinLoss . . . 63
4.3.4 Reinforcement Learning Formulation: SelectR . . . 64
4.3.5 Training Algorithm . . . 66
4.4 Experiments with Non-autoregressive Models . . . 67
4.4.1 Datasets and Prediction Networks . . . 68

4.4.2 Baselines and Evaluation Metric . . . 69
4.4.3 Results and Discussion . . . 70
4.5 Experiments with Auto-regressive Models . . . 73
4.5.1 Prediction Network . . . 73
4.5.2 Results and Discussion . . . 74
4.6 Conclusion . . . 75

5 Neural Models for Output-Space Invariance in Combinatorial Problems 77
5.1 Introduction . . . 78
5.2 Related Work and Background . . . 79
5.2.1 Generalization of GNNs across Graph Sizes . . . 81
5.3 Preliminaries and Problem Definition . . . 82
5.4 Models Description . . . 84
5.4.1 Binary Model . . . 84
5.4.2 Multi-valued Model . . . 88
5.4.3 Relative Comparison . . . 92
5.5 Experiments . . . 93
5.5.1 Task Description and Datasets . . . 93
5.5.2 Experimental Setup & Baselines . . . 96
5.5.3 Results and Discussion . . . 98
5.6 Conclusion . . . 100

IV Explicit and Unknown Constraints 101

6 Scalable Learning of Constraints in Neural ILP Architectures 103
6.1 Introduction . . . 104
6.2 Related Works . . . 106
6.3 Differentiable ILP Loss . . . 108
6.3.1 Background and Task Description . . . 108
6.3.2 A Solver-free Framework . . . 109
6.3.3 Negative Sampling . . . 112
6.4 Experiments . . . 113
6.4.1 Symbolic and Visual Sudoku . . . 114
6.4.2 Random constraints . . . 116
6.4.3 Knapsack from Sentence Descriptions . . . 117
6.4.4 Keypoint Matching . . . 118
6.5 Discussion . . . 120

6.6 Conclusion . . . 121

7 OxKBC: Outcome Explanation for Knowledge Base Completion 123
7.1 Introduction . . . 124
7.2 Related Work . . . 125
7.3 OxKBC: The Outcome Explanation Engine . . . 127
7.3.1 Language for generating Explanation Paths . . . 127
7.3.2 Templates: Aggregation of similar Explanation Paths . . . 128
7.3.3 Selection Module . . . 130
7.4 Experiments . . . 133
7.4.1 Internal Evaluation . . . 134
7.4.2 User Evaluation . . . 135
7.4.3 Faithfulness of Explanations . . . 138
7.5 Discussion . . . 139
7.6 Conclusions . . . 140

V Epilogue 141

8 Conclusion and Future Work 143
8.1 Conclusion and Summary . . . 143
8.2 Future Work . . . 145
8.2.1 Combining Known and Unknown Constraints . . . 145
8.2.2 Solution Multiplicity . . . 146
8.2.3 Output Space Invariance . . . 147
8.2.4 Neuro–symbolic Architectures . . . 147

Bibliography 179

VI Appendices 181

A Appendix to Chapter 3 183
A.1 Experimental Details . . . 183
A.1.1 Semantic Role Labeling (SRL) . . . 183
A.1.2 Named Entity Recognition (NER) . . . 184
A.1.3 Fine Grained Entity Typing . . . 184

B Appendix to Chapter 4 **187**
B.1 Experimental Details . . . 187
B.1.1 Datasets and Prediction Networks . . . 187
B.1.2 Results and Discussions . . . 193

C Appendix to Chapter 5 **195**
C.1 Models Description . . . 195
C.1.1 Alternate encoding scheme in Binarized Model . . . 195
C.1.2 Experimental Details . . . 196

D Appendix to Chapter 6 **199**
D.1 Mean and Std Err. over multiple runs . . . 199

E Appendix to Chapter 7 **201**
E.1 Related Works . . . 201
E.1.1 Comparison of TF based KBC Models with Path based Approaches 201
E.1.2 Pedagogical Approach . . . 201
E.2 Experiment Details . . . 202
E.3 Mechanical Turk Experiment Details . . . 203

List of Publications **205**

Biography **207**

List of Figures

1.1 A schematic diagram placing our different works in a 2×2 grid with one dimension representing if the constraints are *Known* vs *Unknown* and the other dimension differentiating based on how the constraints are represented: *Explicit* or *Implicit*. 12

1.2 A neuro-symbolic pipeline where the input is an image of a sudoku puzzle and the output is the corresponding solved sudoku. The 'perception model' decodes the image in each cell into one of the nine classes and the 'reasoner' discovers the unknown constraints while training and fills each cell with a digit between 1 and 9. 14

1.3 Solution Multiplicity: A simple sudoku input with only 4 empty cells and its corresponding solutions. 17

1.4 Neural-ILP Architecture: The reasoning component is represented as an optimization problem, which can be solved either by a black-box solver such as Gurobi [Gurobi Optimization, LLC, 2022], or by following steps of a specific algorithm such as cutting-planes [Gomory, 2010] 18

2.1 Known constraints: categorization of related works on the basis of how they formulate the constraints (hard vs soft) and when do they enforce the constraints (training or inference). 22

3.1 NER: Comparison of different training techniques. B: Baseline; CL: Constrained Learning; SCL: Semi-supervised Constrained Learning; CI: Constrained Inference 48

4.1 Decision Boundary learnt by logistic regression guided by MinLoss. Green vertical line at $\mathbf{x} = 0$ is the initial decision boundary and black vertical line at $\mathbf{x} = -0.55$ is the decision boundary at convergence. 64

4.2 Flow-diagram for our RL Framework 65

4.3 Left: Accuracy vs size of query's solution set (with 95% confidence interval). Right: #givens and #datapoints vs size of query's solution set 71
4.4 Fraction of training samples for which arg max of G_ϕ probability distribution is different from the target closest to model prediction. For I-ExpLR, this fraction is 0% 72

5.1 An example Futoshiki Puzzle of size 3×3 and the corresponding graphs. A value of -1 indicates an unassigned variable. Black and red edges are Constraint and Relation edges respectively. The digits $5, 7, 1$ in square boxes represent a random 3-permutation of $k_{\max}$, used in multi-valued model for initialization of node embeddings. 84
5.2 Hidden State Update at time-step t: We take a toy graph with 3 nodes, $\{a, b, c\}$, and 2 edge-types, $\{q, r\}$, to illustrate edge-dependent message passing and hidden state update of node b. First, messages along the four edges are calculated as an edge-type dependent function of the sender and receiver hidden state using f_q and f_r. Next, the incoming messages are aggregated by edge-type (*e.g.*, using attention based mechanism or simple averaging), and the outputs are concatenated to obtain the final message, $m_{t-1}[b]$. The hidden state of node b is updated by function g which takes the previous hidden state $h^{t-1}(b)$, the incoming message $m_{t-1}[b]$, and the initial embedding of the node as its input. 87
5.3 Resource: Futoshiki 99

6.1 An illustration of our framework. Figure on the left shows 4 ground truth constraints that need to be learnt. Blue dots are the only feasible integral points w.r.t. the 4 constraints. Shaded area containing only the dot with green border is the feasible region after we add the cost constraint (dashed line). Figure on the right shows an intermediate scenario while learning. The green-bordered dot (positive) is outside the intermediate 4^{th} constraint and the red dot (negative) is inside the intermediate 3^{rd} constraint. Positive and negative losses encourage the 4^{th} and the 3^{rd} hyperplanes to move in the direction shown by the green and red dotted arrows respectively. . . 111

7.1 Architecture of our Selection Module. The underlying two-layer MLP is same when predicting score for every template. 132

B.1 8-Queens query along with its two possible solution.[1] 188

E.1 Human Intelligence Task on Amazon Mechanical Turk 204

List of Tables

3.1 $g(\mathbf{w})$, $g_1(\mathbf{w})$ and $g_2(\mathbf{w})$ are soft value functions for C, C_1 and C_2, respectively. 42
3.2 Effect of constrained learning on SRL, with and without constrained inference (CI). 46
3.3 Constraints used in NER task 46
3.4 F–score for different models (mean ± stdev), along with average number of constraint violations 47
3.5 TypeNet: MAP Scores (in %) and # of constraint violations for different training sizes. 50

4.1 Statistics of datasets. 'Train', 'Test' and task names are abbreviated. Devset similar to test. 69
4.2 Mean test accuracy (±standard error) over three runs for multiplicity aware methods compared with baselines. *OS*: test queries with only one solution, *MS*: queries with more than one solution. 70
4.3 Comparing the accuracy of an autoregressive encoder-decoder with a non-autoregressive encoder. For easy reference, we copy from Table 4.2 the mean accuracy obtained by RRN. *OS*: test queries with only one solution, *MS*: queries with more than one solution. 75

5.1 Dataset details 93
5.2 Test Data statistics for all three tasks 94
5.3 Range for p for given k and n for GCP data generation 94
5.4 Futoshiki: Mean (Std. dev.) of Board and Pointwise accuracy on different board sizes. MV and BIN correspond to Multi-valued Model and Binarized Model, respectively. 97
5.5 GCP: Mean (Std. dev.) of coloring and pointwise accuracy on graphs with different chromatic number. 97
5.6 Sudoku: Mean (Std. dev.) of board and pointwise accuracy on different board-sizes. Both models trained on 9×9 puzzles 98

5.7 Sudoku: Mean (Std. dev.) of board and pointwise accuracy of models fine-tuned on 24 board-size 98

6.1 Board accuracy and training time for different board sizes of symbolic and visual sudoku (for CombOptNet, "-" denotes time-out after 12 hours) . . 115

6.2 Mean of the vector accuracy ($\mathbf{M}_\Theta(x) = \mathbf{y}^*$) and training time over the 10 random datasets in each setting of Random constraints. Number of learnable constraints is twice the number of ground truth constraints. See section D.1 for std. err. over the 10 runs. 117

6.3 Mean accuracy and train-time over 10 runs for various knapsack datasets with different number of items in each instance. For $N = 25, 30$, "-" represents timeout of 12 hours and we evaluate using the latest snapshot of the model obtained within 12 hours of total training. See section D.1 for std. err. over the 10 runs. 118

6.4 Keypoint Matching: Number of train and test samples for datasets with different keypoints 119

6.5 Average point-wise accuracy and training times over 3 runs with different random seeds for varying # of keypoints. Neural + CI denotes ILP inference with known constraints over the cost learnt by neural model. See section D.1 for std. err. over the 3 runs. 120

7.1 Input Features for a prediction $r(s, o)$ from i^{th} template Ti 131

7.2 Micro-F1 score of OxKBC (mean ± std deviation) 134

7.3 OxKBC vs. rule mining under different scenarios: **Overall:** when either of the two systems give an explanation; **Both:** When both give an explanation; **OxKBC:** When only OxKBC gives an explanation (explained above); Similarly for 'Rules'. When only one of the systems produces an explanation (last two rows), we ask workers whether it is better or worse than having no explanation. If it is selected as being worse, we mark the other system, producing 'no explanation', as better. 136

7.4 OxKBC was preferred to Rule Mining for first two facts, rule mining won in third fact, and none of the explanations were accepted in the last example. 137

7.5 OxKBC explanations from N-GT@1 when the top prediction is correct in real world. 137

7.6 OxKBC explanations for samples from N-GT@1 when the predictions are actually incorrect. 138

7.7 MRR over 100 samples when the probability of the supporting fact in the explanation is reduced. Control MRR reports the MRR when probability of any other related fact in the KB is reduced. 138

A.1 Best hyper-parameters in SRL experiments for different training sizes. . . 183
A.2 Best hyper-parameters in NER for different training sizes in both scenarios: CL and SCL. 184
A.3 Best hyper-parameters in Typenet for different training sizes in both scenarios: CL and SCL. 184

B.1 Seed wise gains of SelectR over MinLoss across different random seeds and experiments 193

C.1 Hyperparameters for different models and tasks 196
C.2 Training cost of different models in terms of number of epochs, gradient updates and clock time 197

D.1 Mean ± std err of the vector accuracy ($\mathbf{M}_\Theta(x) = \mathbf{y}^*$) and training time over the 10 random datasets in each setting of Random constraints. Number of learnable constraints is twice the number of ground truth constraints. CombOpt: CombOptNet 200
D.2 Mean ± std err. of accuracy and train-time over 10 runs for various knapsack datasets with different number of items in each instance. For $N = 25, 30$, *TO* represents timeout of 12 hours and we evaluate using the latest snapshot of the model obtained within 12 hours of total training. CombOpt: CombOptNet 200
D.3 Mean ± std error of point-wise accuracy and training times over 3 runs with random seeds for varying number of keypoints. Neural+CI denotes ILP inference with known constraints over the cost learnt by neural model. 200

E.1 Hyper-parameters used for various settings 202

Part I

Prologue

Chapter 1

Introduction

Over the last decade, deep learning has revolutionized the field of Artificial Intelligence. Deep neural models have already achieved state-of-the-art performance in many tasks in computer vision, such as image classification [Deng et al., 2009], object detection [Zhao et al., 2018], image segmentation [Minaee et al., 2022], and many Natural Language Processing (NLP) tasks such as Semantic Role Labeling (SRL) [Carreras and Màrquez, 2005], Named Entity Recognition (NER) [Tjong Kim Sang and De Meulder, 2003], Dialog Systems [Bordes et al., 2017], Question Answering [Yang et al., 2018, Kwiatkowski et al., 2019, Karpukhin et al., 2020], Summarization [See et al., 2017, Zhang et al., 2020], machine translation [Vaswani et al., 2017, Wu et al., 2016, Sutskever et al., 2014] Knowledge Base Completion (KBC) [Wang et al., 2015, Jain et al., 2018], and many more.

We may sometimes have some prior domain knowledge about the underlying task. However, in most cases, the model is trained using only the input-output pairs in the training data without considering the expert knowledge. For example, consider the task of Fine-Grained Entity Typing [Murty et al., 2018] posed as a multi-label classification problem where each class is a fine-grained entity type, *e.g.*, PERSON, ARTIST, MUSICIAN. Now, one may expect that if an entity is tagged as MUSICIAN, it must also be tagged as an ARTIST and as a PERSON. However, this is not always the case with the output of a trained model, especially when the training data is less. We observe such discrepancies in the output even when we use general purpose Large Language Models (LLMs) that rely only on the input instructions to understand the domain and the task.

In many cases, prior domain knowledge as above can be succinctly expressed in the form of an explicit constraint over output symbols, *e.g.*, $\mathbb{1}\{\mathbf{y}_{MUSICIAN} = 1\} \implies \mathbb{1}\{\mathbf{y}_{ARTIST} = 1\}$ and $\mathbb{1}\{\mathbf{y}_{MUSICIAN} = 1\} \implies \mathbb{1}\{\mathbf{y}_{PERSON} = 1\}$. Here, the first constraint encodes that if variable $\mathbf{y}_{MUSICIAN}$, corresponding to label MUSICIAN, is assigned a value of 1, then variable $\mathbf{y}_{ARTIST}$, corresponding to the label ARTIST, should also be assigned a value of 1.

Enforcing such constraints can presumably regularize a deep learning model and improve its performance.

In the above discussion, we assumed that the constraints are explicitly known. But this may not always be the case. There are scenarios where the constraints are unknown but form the core of the underlying reasoning task, such as combinatorial puzzles or Constraint Satisfaction problems (CSPs) with unknown constraints. Recently, there has been a renewed interest in the AI community for creating deep learning models for solving such symbolic reasoning tasks requiring higher level inferences, such as neural theorem proving [Rocktäschel et al., 2015, Evans and Grefenstette, 2018, Minervini et al., 2020], playing blocks world [Dong et al., 2019], travelling salesman problem [Karalias and Loukas, 2020]. One advantage of neural models for such tasks is that they can potentially result in a unified, end–to–end trainable representation for integrated AI systems that combine perceptual and high–level reasoning. One such high–level reasoning task is solving combinatorial problems that are essentially Constraint Satisfaction Problems, albeit with unknown constraints, *e.g.*, combinatorial puzzles like sudoku [Wang et al., 2019, Palm et al., 2018]. Here, the underlying task is governed by some fixed constraints that may not be known to us. The objective of the training is to learn to solve the task using many input instances of the puzzle and their corresponding solution. There are broadly two approaches to achieve this: either learn a parameterized function that directly maps an input puzzle to its output or explicitly learn the *hard* constraints governing the task and use them with a solver to find the output. Here, we call the constraints '*hard*' because violating them is not allowed and will result in an incorrect solution, *e.g.*, violating any constraint in sudoku results in an incorrect solution. This is in contrast to '*soft*' constraints where one is allowed to violate the constraint but pay the penalty for violations, *e.g.*, a constraint $BornIn(s, o) \implies CitizenOf(s, o), \forall\ s, o$, which indicates that if a person, s, is born in the country o, he must be o's citizen, is likely, but may not always hold.

The focus of this work is *Deep Learning with Constraints*. We will discuss both the cases of known as well as unknown constraints. But before we dive into our contributions, we first discuss different aspects of constraints in machine learning: What are the benefits of imposing them? What are the variables involved in the constraint expression? Are the constraints represented *explicitly* or *implicitly*? Are the constraints *known*, or *unknown* but *learnable*? What are the different ways of combining constraints with neural models? Each of these questions gives us a dimension to categorize our contributions. In particular, we will use the dimension of *representation* (explicit vs implicit) and *knowledge* (known vs unknown) to place our contributions in the three buckets, *viz.*, *known* constraints with *explicit* representation, *unknown* constraints with *implicit* representation, and *unknown* constraints with *explicit* representation.

1.1 Constraints in Machine Learning

The use of constraints in machine learning predates the modern deep learning era. For example, Druck et al. [2011] presents a detailed tutorial on exploiting rich prior knowledge while training non-neural models for various NLP tasks. Constrained Conditional Models (CCM) [Roth and Yih, 2004, Wang et al., 2010] have been quite popular for enforcing constraints on top of a learnt model and have shown significant improvements for many NLP tasks [Punyakanok et al., 2005, 2008, Chang et al., 2013]. For deep learning models, Borghesi et al. [2020] published a survey on improving them using domain knowledge expressed as constraints. Recently, Large Language Models (LLMs) have gained considerable popularity and made cutting–edge AI available to the masses. They have become the de facto choice for solving many tasks. Constraints on the LLMs' output can ensure that the generated text adheres to user–specific requirements while being unbiased and legally–compliant. Constraints can play a crucial role in harnessing the power of LLMs for various applications while mitigating potential risks and ensuring responsible AI deployment. Jiang et al. [2023] recently released a benchmark to access LLMs' capability to satisfy constraints expressed in the instructions.

This section first describes various benefits of imposing constraints on a machine learning model, followed by different aspects of the constraints that can be used to categorize them.

1.1.1 Benefits of Imposing Constraints

1. **A tool to inject domain knowledge:** As discussed earlier, one may express domain knowledge through task–specific constraints. Enforcing them on the model's output is one of the ways to exploit domain knowledge to improve the model's performance. For example, Geng et al. [2023] propose grammar-constrained decoding on an LLM's output for structured NLP tasks; Wang et al. [2024] propose constrained decoding as a way of injecting updated knowledge (also known as knowledge editing) into an LLM; [Bogoychev and Chen, 2023] enforce terminology constraints on the output of an LLM for domain–specific translations.

2. **A tool for reasoning:** Many reasoning problems can be expressed as a Constraint Satisfaction Problem (CSP) or as an Integer Liner Program (ILP). Here the constraints govern the task definition and act as an important tool for representing the problem. For example, consider the task of visual–sudoku where input is an image of a sudoku board and output is the corresponding two dimensional grid with a digit filled in each of its cell, satisfying the constraints of the puzzle. To solve

this task, a neural model may decode the image of the input sudoku board into a 2D tensor of digits. Now, when constraints of sudoku are assumed to be known, then a CSP or an ILP solver can be used for reasoning over the decoded digits. However, when the constraints are unknown, we can simultaneously learn the linear constraints of sudoku along with a neural model for decoding the sudoku image. An ILP solver, such as Gurobi [Gurobi Optimization, LLC, 2022] can then be used with the learnt constraints and the decoded digits to solve the puzzle [Nandwani et al., 2022b]. Alternatively, we may design a neural model for symbolic reasoning and an architecture based on the structure of the constraints may lead to better generalization. For example, using a Graph Neural Network (GNN) in conjunction with a CNN leads to better performance than a pure CNN for visual–sudoku.

3. **Performance gain in low data regime:** Enforcing constraints while training the model, especially under a low data regime, helps improve performance as the constraints provide an additional signal to guide the training process. For example, Xu et al. [2018] show improved performance in image classification task on MNIST ([LeCun and Cortes, 2010]) and Fashion MNIST ([Xiao et al., 2017]) under low data regime; Mehta et al. [2018] improve performance of SRL task by imposing constraints from the syntactic parse trees.

4. **Exploiting unlabeled data (semi-supervision):** Formulation of task–specific constraints often depends only on the input and its corresponding output and does not require the ground truth labels for its evaluation. Hence, the additional constraint term provides a natural way of exploiting the unlabeled data as it does not depend upon the ground truth labels.

5. **Domain adaptation:** When adopting a model to a different domain or task, *e.g.*, when using Resnet-152 or its variant for medical images instead of regular images, it becomes imperative to regularize or constrain the model output with additional constraints, *e.g.*, size or shape constraints in image segmentation [Oktay et al., 2018].

6. **Safe, robust and interpretable AI:** When AI models are used in critical applications such as autonomous vehicles, enforcing safety constraints becomes important [Wen et al., 2020, HasanzadeZonuzy et al., 2021]. Constraints can also prevent LLMs from generating harmful, offensive or biased content [Gallegos et al., 2023]. Making the models' output satisfy both safety and commonsense constraints is one of the essential steps towards instilling users' trust in them. Agrell et al. [2018] also argue that imposing "phenomenological constraints" which guide the phenomena generat-

ing the data, aids in making the model more transparent by avoiding overfitting, especially when the model extrapolates outside the range of given training data.

7. **Better generalization through regularization:** State-of-the-art neural models have millions of parameters, *e.g.* BERT ([Devlin et al., 2019]) has 340 million parameters; image classification model Resnet 152 ([He et al., 2016]) has around 60.2 million parameters. Optimizing only the supervised loss function, *e.g.*, cross entropy *w.r.t.* ground truth labels, often leads to over-fitting on the training data. Regularizing the weights of a machine learning model is a common practice to avoid overfitting, and many techniques, like L_1/L_2 norm regularization, are ubiquitous tools to avoid overfitting. These regularization methods encoding important inductive biases are nothing but soft constraints, albeit directly on the network parameters instead of the network output [Ravi et al., 2018].

1.1.2 Variables in the Constraint Expression

One can view constraints as means for representing our inductive biases (or priors) over the two types of variables involved in any ML model: the model's parameters and the model's output/input. Depending on the variables and the functions involved in the constraint expression, we broadly classify them into the following categories:

1. **Constraints over Model Parameters:** In this category, constraints are represented directly in terms of the weights/parameters of the underlying model. Methods to avoid overfitting, such as L1/L2 regularization, max–norm and unit–norm constraints fall under this category [Ng, 2004, Srebro and Shraibman, 2005, Salimans and Kingma, 2016]. Typically, these constraints are designed by ML experts to solve the learnability issue. Such constraints are often inspired by the underlying mathematical properties, *e.g.*, enforcing recurrent matrices to have eigenvalues close to one solves exploding and vanishing gradient issues while training a Recurrent Neural Network (RNN) [Mikolov et al., 2015]. Similarly, to improve learnability in CNNs, Tai et al. [2016] and Liu et al. [2015] study low rank regularization and sparsity constraints respectively.

2. **Constraints over Model's Output/Input:** In this category, constraints are represented over the model's output and/or input. Such constraints are often inspired by the underlying domain knowledge for a specific task and are best provided by either a domain expert or commonsense. Even if we train a network in oblivion to such constraints, we expect not to violate them. For example, in entity typing and

relation extraction task, we expect the head and the tail entities for a given relation to be of a specific type, *e.g.*, relation *'WorksAt'* should have type *'PERSON'* of the head entity and type *'ORG'* of the tail entity [Pan and Srikumar, 2018].

3. **Constraints over Auxiliary Variables:** At times, it is easier to express the constraints over the auxiliary variables that are computed as some function of the model's output. While the constraint is designed by domain experts, the function to compute the auxiliary variables is designed by ML experts. For example, in a pose–conditional person image generation task, one may expect the generated and the ground truth image to have the same location of different body parts. Hu et al. [2018] use a human body–part parsing neural model [Gong et al., 2017] to segment both the generated image and the ground truth image and enforce the two segmentation masks to be close to each other. In this case, the two segmentation masks are the auxiliary variables computed using a separate neural function (body–part parsing model).

1.1.3 Representation of Constraints

There are broadly two ways to represent constraints: either as an explicit formula (Boolean or arithmetic), or implicitly as a black-box function that results in an assignment satisfying constraints. Below we briefly describe each of them:

1. **Explicit representation:** This method is the most natural way of representing the constraints. Here, a constraint is represented as either a Boolean expression over the output symbols [Hu et al., 2016a, Rocktäschel et al., 2015, Guo et al., 2016, Diligenti et al., 2017b], or as arithmetic expression over output variables, *e.g.*, linear constraints in (Binary) Integer Linear Programs [Roth and Yih, 2005, Ning et al., 2018, Han et al., 2019, Breitfeller et al., 2021]. Such explicitly represented constraints can be used as an input to an appropriate solver, *e.g.*, Gurobi[Gurobi Optimization, LLC, 2022], and result in an output satisfying the constraints.

2. **Implicit representation:** We call a constraint representation implicit when we do not express them in the language of any solver, such as a Boolean or arithmetic expressions. Instead, we learn a neural model that maps input directly to the constraint satisfying output. For example, Palm et al. [2018] propose Recurrent Relational Networks (RRN) that can be used to solve sudoku puzzles. One can think of such neural models as representing the constraints implicitly in their weights.

1.1.4 Known vs Unknown Constraints

When constraints represent the domain knowledge, either from an ML expert or a domain expert, it is assumed to be known apriori. However, experts may not always be able to express all the constraints, and thus one may wish to learn them from the data. Pan et al. [2020] attempts to learn the underlying fixed constraints governing the output as linear inequalities and enforce them on model's predictions. Besides, when constraints define the underlying reasoning task, such as playing sudoku, the objective of training a model is to discover the fixed but unknown constraints. Neural solvers, such as RRN [Palm et al., 2018], Neural Logic Machines (NLM) [Dong et al., 2019] represent the unknown constraints implicitly whereas methods like rule mining [Galárraga et al., 2013], Inductive Logic Programming (ILP) [Zeng et al., 2014, Raedt et al., 2008, Lavrac and Raedt, 1995] intend to learn explicit constraints in the form of horn clauses ($A \implies B$).

1.1.5 Hard Vs Soft Constraints

Constraints are hard when we are not allowed to violate them. Doing so results in an incorrect answer, *e.g.*, if any of the underlying constraints of the sudoku puzzle is violated, the corresponding solution won't be correct. On the other hand, constraints are soft when we are allowed to violate them, but we pay some penalty for their violation.

1.1.6 Methods for Combining ML Models and Constraints

1.1.6.1 Methods for Enforcing Known Constraints

Below we enumerate a few methods to enforce constraints on the output of an ML model. The first three methods result in a 'constraint–aware' model whose predictions are expected to satisfy constraints. On the other hand, 'constrained inference' makes use of an external module that processes the model's prediction and results in an output that satisfies the constraints. Of course, one can always combine the different methods, resulting in a more robust system.

1. **By Design:** One of the earliest approaches to inject domain knowledge into a neural network has been to design the architecture such that the constraints are automatically enforced. Methods under this category do not perform any constrained training, but constraint the network architecture so that the underlying constraints are not violated [Towell and Shavlik, 1994, Amos and Kolter, 2017, Li and Srikumar, 2019, Frerix et al., 2019, Lutter et al., 2019, Casado and Martínez-Rubio, 2019, Mahabal et al., 2019, Wang et al., 2019].

2. **Data Augmentation:** In this approach, additional training data is generated based on the information available through domain knowledge [Rocktäschel et al., 2015, Vo et al., 2019, Mollaysa et al., 2019, Bjerrum, 2017]. For example, in the task of Knowledge Base Completion, Rocktäschel et al. [2015] insert new facts deduced from the existing facts and known constraints. This approach has been quite popular in training computer vision models where additional data is generated by transforming an image while preserving its label [Perez and Wang, 2017]. The constraint here is that the label of the original and the transformed image should be the same. These label–preserving transformations are often guided by underlying phenomenological constraints of the data generation process.

3. **Constrained Training:** Using the constraints while training gives a strong prior to the model and can result in significantly improved predictions by making the trained model *constraint-aware*. When the constraints are soft, the model is allowed to violate the constraints but pays a penalty for doing so. Accordingly, we can modify the prediction loss by adding to it an auxiliary loss proportional to the constraint violation penalty [Kervadec et al., 2019, Oktay et al., 2018, Rocktäschel et al., 2015, Xu et al., 2018]. Here the auxiliary loss term essentially acts as a '*Regularizer*'. When the constraints are hard, one can pose training as a constrained optimization problem. Ravi et al. [2018], Roy et al. [2018]) modify the weight update equation in the stochastic gradient descent methods so that the weights always lie in the constrained manifold. In their work, constraints are inspired by learnability and are expressed directly over the model parameters (section 1.1.2).

4. **Constrained Inference:** Traditionally, the most common method of enforcing constraints in machine learning has been through '*constrained inference*' on top of a learned model, *e.g.*, CCM [Roth and Yih, 2004] use Integer Linear Programming (ILP) to perform inference with global hard constraints [Roth and Yih, 2005, Chang et al., 2013, Ning et al., 2018]. Rush and Collins [2012] provide a detailed description of using dual decomposition to solve locally decomposable constraints.

1.1.6.2 Methods for Learning Unknown Constraints

We can divide the methods for learning unknown constraints into two main categories: 1) Neural solvers, and 2) Symbolic–solver based methods.

1. **Neural Solvers:** These methods represent the constraints implicitly and train a neural model to map an input puzzle directly to the corresponding constraint–satisfying solution, *e.g.* RRN [Palm et al., 2018], NLM [Dong et al., 2019].

2. **Symbolic–Solver based Methods:** These methods learn constraints explicitly, *e.g.* linear inequality constraints for a combinatorial problem [Paulus et al., 2021, Niepert et al., 2021, Wilder et al., 2019]. Once the constraints have been learnt, a symbolic solver, such as Gurobi [Gurobi Optimization, LLC, 2022] for Integer Linear Programs, can solve any input problem.

1.2 Contributions

This work focuses on 'Deep Learning with Constraints'. We use each of the dimensions defined above to position our work:

Knowledge and representation of the constraints: We first use the two dimensions of knowledge and representation to place our work in a 2×2 grid. Using the axis of 'Knowledge', we first create two broad buckets of "Known Constraints" and "Unknown Constraints". We then further divide unknown constraints based on the constraint representation, resulting in three categories that form the three main parts of this thesis: *known* constraints with *explicit* representation (part II); *unknown* constraints with *implicit* representation (part III); and *unknown* constraints with *explicit* representation (part IV) (see fig. 1.1).

Variables in the constraint expression: In all of our works, the fixed constraints, whether known or unknown, can be expressed using the output variables and are inspired by the domain knowledge or the underlying data generation process.

Soft vs hard constraints: In all of our works except for the last one, we focus on hard constraints, whether known or unknown. In the last work, the horn–clause constraints learnt by the system form a plausible explanation for the prediction of a model, but the constraint may not always hold in the data, making them soft.

Methods for enforcing constraints: When the constraints are known, we focus on learning with known constraints instead of constrained inference (part II). When the constraints are unknown, we focus on both neural solvers (part III) and symbolic–solver based methods (part IV).

In the remainder of this section, we focus on each of the three broad categories defined above (fig. 1.1) and describe our contributions in each category.

1.2.1 Known Constraints

As discussed earlier, constraints act as a good way of injecting known domain knowledge about the task. In the vanilla setting, deep neural networks are trained end to end

Constraints ⇨ / Representation ⇩	Knowledge: Known	Knowledge: Unknown
Explicit	• Primal Dual Formulation for Learning with Constraints. (NeurIPS 2019)	• OxKBC: Outcome Explanation for KBC. (AKBC 2020) • Scalable Learning of Cost and Constraints in Neural-ILP architectures. (NeurIPS 2022)
Implicit		• Training Neural Reasoners under Solution Multiplicity. (ICLR 2021) • Neural Reasoners for Output-space Invariance. (ICLR 2022)

Figure 1.1: A schematic diagram placing our different works in a 2×2 grid with one dimension representing if the constraints are *Known* vs *Unknown* and the other dimension differentiating based on how the constraints are represented: *Explicit* or *Implicit*.

from data, without incorporating any additional domain knowledge (other than neural architecture and loss function). However, for many problems of interest (e.g., structured prediction or multi-task learning), there is a set of natural constraints which need to be satisfied over the output variables. Incorporating these constraints into a model can presumably regularize the output space resulting in improved predictions.

In section 1.1.6.1 we discussed various ways of enforcing known constraints. One of them involves using constrained inference after the model has been trained. In our work, we argue that this might be sub-optimal since the original training of the network was done oblivious to the constraints. On the other hand, enforcing the constraints explicitly during training may give a strong prior to the model. It not only reduces constraint violations but can also result in significantly improved predictions by making the training *constraint-aware*.

In part II, we present a principled solution to the problem of learning a deep network with a given set of hard constraints on output labels. We formulate this as a constraint optimization problem wherein we maximize the original learning objective (e.g., cross entropy) subject to the constraints being satisfied for each example in the training data. Our contribution can be summarized as follows: (1) We present a Lagrangian based formulation for learning with constraints in a deep network. Our constraints make use of soft rules to deal with logical operators. (2) We employ a min-max based optimization to solve our constrained formulation. To the best our knowledge, we are the first to use a

min-max based formulation for learning with constraints specified over output variables in deep networks. (3) We present experimental results on three different tasks demonstrating the effectiveness of our approach, while achieving state-of-the-art results in two of the domains and also significantly reducing the number of constraint violations in each case. Our code is available at: https://github.com/dair-iitd/dl-with-constraints.

1.2.2 Unknown Constraints

We discussed in section 1.1.4 that it is not always feasible for domain experts to express their intuition in the form of a succinct Boolean expression or as linear inequalities, as done in (Binary) Integer Linear Programs (ILPs). We also observed that in addition to using constraints (either during learning or inference) to improve the performance of a model, there are other scenarios where the fixed unknown constraints form the core of a downstream reasoning task, such as solving combinatorial puzzles like sudoku [Palm et al., 2018]. Recently there has been a surge of papers that focus on designing end-to-end differentiable models for solving reasoning tasks that are inherently constraint satisfaction problems, but with unknown constraints. There are primarily two ways in which unknown constraints can be learnt from the data. First approach involves implicit representation of constraints and we discuss it in section 1.2.3. The second approach involves explicit representation of constraints and is discussed in section 1.2.4.

Many recent works [Palm et al., 2018, Dong et al., 2019] have focused on training purely neural models for reasoning tasks such as sudoku or blocks world. These neural models essentially represent a parameterized function that maps a given input puzzle to its solution, thereby *implicitly* representing the underlying constraints of the combinatorial problem in its weights. We discuss them in section 1.2.3.

Implicit representation of constraints results in black-box models that are hard to interpret. On the other hand, learning constraints in an explicit symbolic representation, using only pairs of input / output data, may not be amenable to end-to-end differentiable learning. For example, if the reasoner in fig. 1.2 is a symbolic ILP solver with unknown linear constraints, then it is not clear how to back-propagate through it. We discuss this in section 1.2.4.

Finally, we take a completely different view of constraints as explanations for the predictions of an underlying model. For example, consider the task of Knowledge Base Completion (KBC), where the model infers new facts based on existing facts in an incomplete Knowledge Base. We train a model that learns to output explicit constraints as post-facto explanations for the predictions of a neural KBC model. Here we are dealing with explicit representation of constraints as horn–clauses, hence we discuss this

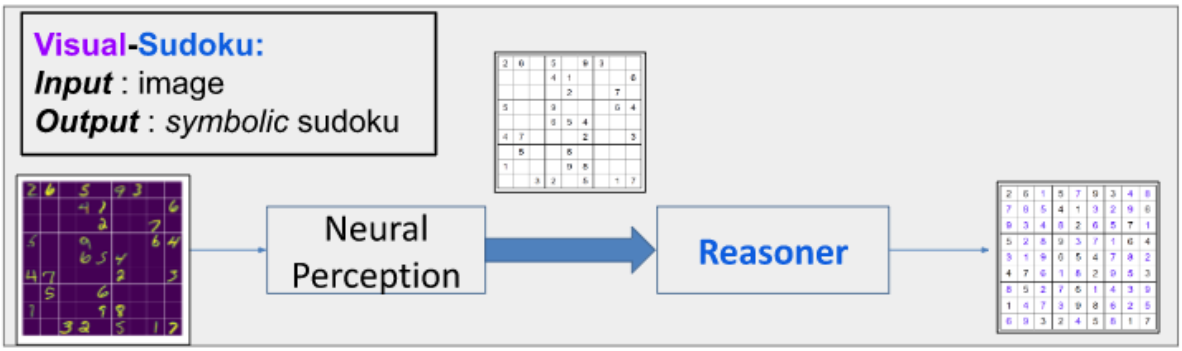


Figure 1.2: A neuro-symbolic pipeline where the input is an image of a sudoku puzzle and the output is the corresponding solved sudoku. The 'perception model' decodes the image in each cell into one of the nine classes and the 'reasoner' discovers the unknown constraints while training and fills each cell with a digit between 1 and 9.

in section 1.2.4.

1.2.3 Implicit Representation of Unknown Constraints

Recent research has explored neural models for solving symbolic reasoning tasks, requiring higher level inferences, such as neural theorem proving [Rocktäschel et al., 2015, Evans and Grefenstette, 2018, Minervini et al., 2020], and playing blocks world [Dong et al., 2019]. The advantage of neural models for these tasks is that it will create a unified, end-to-end trainable representation for integrated AI systems that combine perceptual and high level reasoning. Here we focus on one such high level reasoning task – solving combinatorial problems in structured output spaces, *e.g.*, solving a Sudoku or N-Queens puzzle. These can be thought of as Constraint Satisfaction problems (CSPs) where the underlying constraints are not explicitly available, and need to be learned from training data. We focus on learning such constraints by a non-autoregressive neural model where variables in the structured output space are decoded simultaneously (and therefore independently). Notably, most of the current state-of-the-art neural models for solving combinatorial problems, *e.g.*, NLM [Dong et al., 2019], SATNet [Wang et al., 2019] and RRN [Palm et al., 2018], work with non autoregressive architectures because of their high efficiency of training and inference, since they do not have to decode the solution sequentially.

There are two key aspects of combinatorial problems have been ignored while training neural models for them: 'Solution Multiplicity' and 'Output-space Invariance'.

1.2.3.1 Learning under Solution Multiplicity

One key characteristics of combinatorial problems is *solution multiplicity* – there could be many correct solutions for any given input, even though we may be interested in finding any one of these solutions. For example, the simple sudoku puzzle in fig. 1.3 has two possible solutions and obtaining any one of them suffices for solving Sudoku.

Unfortunately, existing literature has completely ignored solution multiplicity, resulting in sub-optimally trained networks. Our preliminary analysis of a state-of-the-art neural Sudoku solver [Palm et al., 2018][1], which trains and tests on instances with single solutions, showed that it achieves a high accuracy of 96% on instances with single solution, but the accuracy drops to less than 25%, when tested on inputs that have multiple solutions.

Intuitively, the challenge comes from the fact that (a) there could be a very large number of possible solutions for a given input, and (b) the solutions may be highly varied. For example, a 16-givens Sudoku puzzle could have as many as 10,000 solutions, with maximum hamming distance between any two solutions being 61. Hence, we argue that an explicit modeling effort is required to represent this solution multiplicity.

In our work, we formally define the novel problem of learning under solution multiplicity. We call it *One-of-Many Learning* (1oML). We show that a naïve strategy that picks all, or randomly picks any of the correct solutions, is sub-optimal. Next, we introduce a multiplicity aware loss (CC-Loss) and demonstrate its limitations for non-autoregressive models on structured output spaces. In response, we propose a formulation that picks an appropriate solution of a given train query for computing the loss. The selection of the appropriate solution for loss computation can be done in greedy manner, resulting in MinLoss formulation. We show that this is also sub-optimal, and propose an exploration based solution (SelectR) that uses an external selection module to select the appropriate solution for training. Our code for 1oML is available at: https://github.com/dair-iitd/1oML.

1.2.3.2 Neural Models for Output-space Invariance

Another limitation of many reasoners such as RRN [Palm et al., 2018], SATNet [Wang et al., 2019] is that they assume a fixed value-set, *i.e.*, the set from which variables are assigned values is assumed to be constant during training and testing. As a result, they fail to generalize across the size of the value-set, *i.e.*, they can not solve 16×16 sudokus if trained on 9×9 sudoku puzzles.

While an NLM architecture [Dong et al., 2019], consisting of a stack of layers called Logic Machines, aims to learn the first order lifted rules of the underlying problem, and is thus amenable to output space invariance. However, NLMs fail to scale, *i.e.*, they have

[1] Available at `https://data.dgl.ai/models/rrn-sudoku.pkl`

poor performance when the underlying CSP in the training data has many variables, *e.g.*, standard 9×9 sudoku puzzles that have 729 Boolean variables.

On the other hand, both SATNet and RRN scale better than NLM, but they train and test on problems of the same size. They do not learn lifted rules and hence do not generalize across size of the value-set. This is a significant limitation, since it may not always be possible to generate sufficient training data for similar large problems in which variables take values from a bigger value-set.

In this work, we study '*output-space invariance*' in combinatorial puzzles from the same domain: a scenario in which different inputs to the model may come from different but related CSPs that are groundings of a common underlying Lifted CSP [Joslin and Roy, 1997], a (finite) first-order representation that can be grounded to CSPs of varying variable and value-set sizes. We define our problem as: given solved instances and corresponding constraint graph of an unknown ground CSP with a value-set of size k, can we learn neural models that generalize to instances of the same lifted CSP, but with a different value-set of size k' (typically $k' > k$)?

We build our solution using RRNs as the base architecture. They run recurrent message passing on the constraint graph. The nodes are then decoded to obtain a solution. We present two ways to enhance RRNs for output-space invariance.

Binarized Model: Our first model converts a multi-class classification problem into a binary classification problem by converting a multi-valued variable into multiple Boolean variables, one for each value in the value-set. All GNN weights are tied, and because all the variables in the binarized model are Boolean, embeddings for binary values '0' and '1', trained at training time, are directly applicable at test time.

Multi-valued Model: Our second model directly operates on the given multi-valued variables and the corresponding constraint graph, but introduces a *value node* for every value in the value-set. The challenge in this model is initializing value nodes at test time when $k' > k$. We circumvent this problem by training upfront k' or more value embeddings by randomly sub-selecting a k sized subset during each learning iteration. This random sub-selection exploits the symmetry of value-set elements across instances. During test time, k' of the learned embeddings are used.

Our code for output-space invariance is available at: https://github.com/dair-iitd/output-space-invariance.

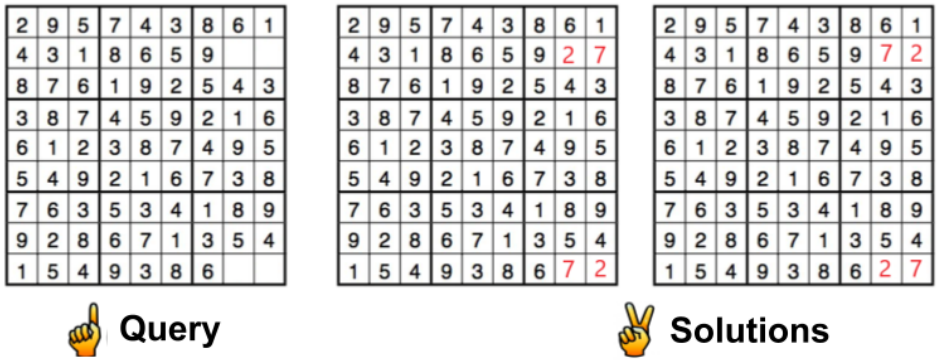


Figure 1.3: Solution Multiplicity: A simple sudoku input with only 4 empty cells and its corresponding solutions.

1.2.4 Explicit Representation of Unknown Constraints

1.2.4.1 Explicit constraints of a combinatorial problem

Implicit representation of constraints results in black-box reasoners which are hard to interpret. Another alternative is to learn the constraints in an explicit form, *e.g.*, as linear inequalities of an Integer Linear Program (ILP). In addition to making the constraints interpretable, representing them explicitly allows us to use symbolic reasoners that take the explicit constraints as input and solve the underlying combinatorial problem. For example, consider the 'Neural ILP' architecture in fig. 1.4 where the reasoning is represented as an Integer Linear Program (ILP). In contrast to the neural reasoners, a forward pass through a symbolic reasoner will solve an optimization problem represented by its input $\mathbf{c}(\theta)$ and parameters $\mathbf{A}$ and $\mathbf{b}$. Once the correct constraints are learnt, we also expect such symbolic reasoners to achieve much better generalization as compared to their neural counterparts.

However, to learn the constraints and the cost, one needs to backpropagate through the symbolic optimization layer and it is not trivial to do so. The derivatives w.r.t. $\mathbf{A}, \mathbf{b}$ and $\mathbf{c}$ are either zero almost everywhere (small changes in the parameters do not change the optima), or undefined (when the optima changes from one corner to another). Recently, few works [Paulus et al., 2021, Niepert et al., 2021] have shown how to compute useful gradients w.r.t. $\mathbf{A}, \mathbf{b}$ and $\mathbf{c}$. However, it requires solving the optimization problem in each learning iteration, making the learning process extremely slow and hence we create a framework for scalable learning of constraints that does not solve the optimization problem at all while training. We discuss it next.

Scalable Learning of Cost and Constraints in a Neural-ILP Architecture: In this work we propose a solver-free framework for learning the linear cost and constraints of a combinatorial problem posed as an ILP. Intuitively, we look at the linear constraints as binary classifiers, separating the given correct solutions from the rest of the space. To

train these classifiers, the given correct solutions act as positive samples for all of them, and we propose different ways of randomly selecting the negative samples. To learn the cost, we convert it into a linear constraint involving the cost vector and corresponding target solution such that the constrained optimization problem becomes a constraint satisfaction problem. This additional linear constraint is learnt in a similar manner as other constraints.

Our code for scalable learning of cost and constraints in a Neural-ILP architecture is available at: https://github.com/dair-iitd/ilploss.

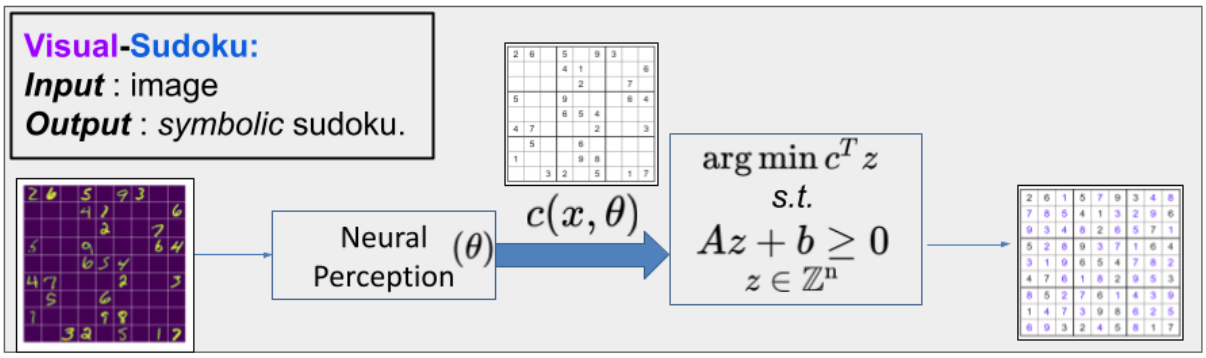


Figure 1.4: Neural-ILP Architecture: The reasoning component is represented as an optimization problem, which can be solved either by a black-box solver such as Gurobi [Gurobi Optimization, LLC, 2022], or by following steps of a specific algorithm such as cutting-planes [Gomory, 2010]

1.2.4.2 Explicit constraints as explanations

In this section of the thesis, we present a novel perspective on constraints, namely 'constraints as explanations'. We argue that constraints not only serve to inject domain knowledge or solve symbolic tasks but can also be used to explain the outcomes of neural models. For instance, consider a prediction such as "*Sachin* is *a citizen of India*". It can be explained by an existing fact "*Sachin* is *born in India*", along with a first–order constraint: "If someone is born in a country, they are a citizen of that country", represented as: s *born in* $o \implies s$ *citizen of* $o, \forall s, o$.

With a surge in the deployment of AI models in real-world applications, there is an increasing need for producing explanations for the predictions of black-box AI models that are used for different tasks within the application. One such task is Knowledge Base Completion (KBC) in which a model infers new facts based on existing facts in an incomplete Knowledge Base (KB), also known as a Knowledge Graph (KG). These facts can then be used to answer queries over the KB/KG.

Inherently explainable models for KBC, like symbolic Horn-clause style inference [Lao and Cohen, 2010, Schoenmackers et al., 2010] use paths in the KG as features, but these are no longer competitive with modern neural methods. Recent research has explored path based approaches for KBC as well. They use random walk with restarts [Lao and Cohen, 2010] and more recently, reinforcement learning [Das et al., 2018, Xiong et al., 2017] to construct multi-hop paths in the knowledge graph and use those to make predictions. The paths may offer natural interpretability [Nickel et al., 2016], but the performance of these models is also significantly lower than recent tensor factorization based neural models like DistMult [Yang et al., 2015] and ComplEx [Trouillon et al., 2016], especially on large KBs like FB15K [Bordes et al., 2013], and YAGO [Suchanek et al., 2007].

An alternative to creating explainable models is to generate explanations for the predictions of a high-quality black-box model [Wachter et al., 2017], *e.g.* TF methods, like DistMult [Yang et al., 2015] and ComplEx [Trouillon et al., 2016]. We take this approach and propose an outcome explanation engine for factorization based KBC models.

OxKBC: Outcome Explanation for Factorization Based Knowledge Base Completion: In this work, we propose OxKBC – an **O**utcome e**X**planation engine for **KBC**, which provides a post-hoc explanation for every new fact inferred by a (uninterpretable) TF model. It first augments the underlying Knowledge Graph by introducing weighted edges between entities based on their similarity given by the underlying model. It then defines a notion of human-understandable explanation paths along with a language to generate them. Depending on the edges, the paths are aggregated into second–order templates for further selection. The explanation engine is divided into two components: a global trained model that, for a given prediction, selects the most plausible second-order explanation template (which quantifies over both relations and entities), and an explanation generator that outputs the best instantiation of the template for this prediction. Experiments over Mechanical Turk demonstrate that users overwhelmingly find our explanations more trustworthy compared to rule mining. We make our code available at: *https://github.com/dair-iitd/OxKBC*

1.3 Thesis Outline

The remainder of this thesis is organized into four main parts. Part II, part III, and part IV categorize the work based on the two dimensions of representation (explicit or implicit) and knowledge (known or unknown) of the underlying constraints, and part V summarizes the work and present few directions for future work. Part II deals with the case of '*explicit* and *known* constraints. In part III we discuss about the neural reasoners

that represent *unknown* constraints *implicitly* in their weights, and Part IV focuses on *explicit* representation of *unknown* constraints.

In part II (chapter 3) we present a primal-dual formulation for deep learning with constraints and demonstrate how to use domain knowledge in addition to training data to train a neural model.

In part III (chapter 4 and 5) we focus on neural reasoners for automatic learning of unknown constraints and observe in chapter 4 that handling of solution multiplicity is important for training neural reasoners. Chapter 5 presents two novel architectures for neural reasoners that exhibit output space invariance and are able to solve, say 16×16 sudoku puzzles after training on 9×9 sudokus.

In part IV (chapter 6) we present a scalable framework for learning linear constraints and cost of an ILP in a Neural-ILP architecture. Finally, in chapter 7, we present a framework that generates constraints (over the existing facts, relations and entities) as explanations for the new facts inferred by a Tensor Factorization based model for KBC.

Chapter 2

Related Work and Background

This chapter broadly discusses the related work and background relevant to our work. The works that are closely related to our contributions are discussed in the respective chapters. In the case of learning with known constraints, we focus on related works that employ different methods for enforcing constraints. For the case of unknown constraints, we first describe in detail the two neural solvers (RRN and NLM) that are used as the underlying neural models in our work on solution multiplicity. In addition, RRN also forms the basis of our novel architecture for value–set invariance. We then describe CombOptNet [Paulus et al., 2021] whose issues with scalability motivated our work that proposes a scalable method for training Neural–ILP architectures. As we did for categorizing our contributions in section 1.2, we use the same dimensions of 'known vs unknown' and 'explicit vs implicit' representation of constraints to classify the related work and background.

2.1 Explicitly Known Constraints

We categorize previous work for enforcing known constraints into four buckets based on whether constraints are 'hard' or 'soft' and whether we enforce them during 'inference' or 'training' (see fig. 2.1). Constraints are hard when we are not allowed to violate them. Doing so results in an incorrect answer, *e.g.*, if any of the underlying constraints of the sudoku puzzle is violated, the corresponding solution won't be correct. On the other hand, constraints are soft when we are allowed to violate them, but we pay some penalty for their violation, *e.g.*, sparsity constraints enforced via L1 regularization.

2.1.1 Constrained Inference

Constrained inference involves using an external module whose input is the model's prediction and it results in an output that satisfies the constraints. It doesn't rely on

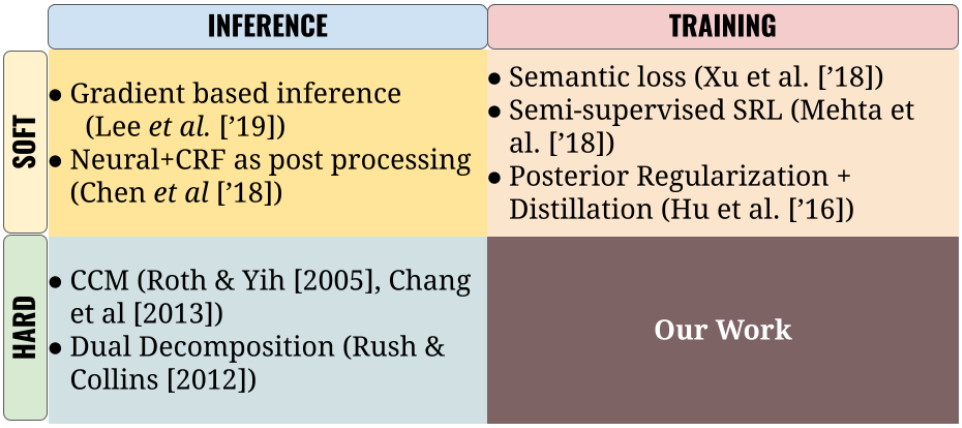


Figure 2.1: Known constraints: categorization of related works on the basis of how they formulate the constraints (hard vs soft) and when do they enforce the constraints (training or inference).

the model to generate a constraint–satisfying output. Different methods can be used for constrained inference, depending on whether the constraints are hard or soft.

2.1.1.1 Hard Constraints

Integer Linear Program (ILP) is one of the most common methods for satisfying hard constraints. For example, Constrained Conditional Models (CCM) [Roth and Yih, 2004] use Integer Linear Programming (ILP) to perform inference with global hard constraints [Roth and Yih, 2005, Chang et al., 2013]. Rush and Collins [2012] provide a detailed description of using dual decomposition to solve locally decomposable constraints. Recent work has explored incorporating these ideas with modern neural networks.

Neural-ILP inference in structured prediction: Many works propose using a constrained ILP inference on the output of neural models, especially when the output is structured, *i.e.*, output is a (multi-dimensional) vector and the structure may impose constraints on the output vector, *e.g.*, hierarchy on the labels in fine grained entity typing [Murty et al., 2018]. In the case of a Boolean vector space, a neural network outputs a score for each label being true and running constrained inference at the time of prediction involves maximizing the total assignment score subject to the known constraints.

Ning et al. [2019] use ILP inference to enforce anti-symmetric (if A before B then B after A) and transitive closure (if A before B and B before C then A before C) constraints in the temporal-reasoning task where objective is to find temporal relationship between event pairs in a document. Similarly, in multi-task setup, Han et al. [2019] and Ning et al. [2018] use ILP to enforce consistency between outputs in the joint task for event extraction and temporal reasoning, and temporal and causal reasoning, respectively.

In a multiple-choice QA (Machine comprehension) task, Pujari and Goldwasser [2019] first use a Natural Language Inference (NLI) system [Parikh et al., 2016] to establish a relationship between different answer choices (*e.g.*, choice A implies choice B). At the time of inference, they constraint the output to respect the discovered relationships by using a Deep relational learning (DRaiL) system [Zhang et al., 2016] for constrained ILP inference.

In the joint task of entity typing and relation extraction task, we expect the types of head and tail entities for a given relation to be of a specific type, *e.g.*, *'WorksAt'* should have *'PERSON'* as a head entity and *'ORG'* as a tail entity. [Pan and Srikumar, 2018] use these hard constraints during inference. However, instead of casting it as an ILP, they resort to graph search algorithms [Russell and Norvig, 2020, Xu et al., 2009] mimicking ILP inference and propose a few heuristics to speed up the search.

2.1.1.2 Soft Constraints

When constraints are soft, a constrained inference may not always result in an assignment that satisfies all the constraints. Here we focus on soft constraints inspired from domain knowledge.

Neural-CRF Models: In many scenarios, the structure in the output space imposes certain soft constraints. For example, in image segmentation, labels of neighboring pixels are more likely to belong to the same segment. One popular technique to impose such an inductive bias over the transition between labels of neighboring pixels is by adding a CRF layer [Koller and Friedman, 2009] at the end of a deep network [Lample et al., 2016]. In a CRF inference, a label vector with the maximum total of assignment score (unary potentials) and pairwise transition score (pairwise-potential) is selected as an output (assuming only unary and pairwise potentials in the CRF model). For stereo-vision and semantic image segmentation, Knobelreiter et al. [2017] and Chen et al. [2018] use a neural model to obtain independent assignment scores for each pixel (unary potentials) and run a CRF inference with manually defined pairwise potentials (transition score) to enforce smoothness priors. Huang et al. [2015] propose a linear chain CRF on top of an LSTM in a sequence tagging task.

Gradient Based Inference: Recently, [Lee et al., 2019] proposed gradient based inference method for Semantic Role Labelling [Carreras and Màrquez, 2005]. At the time of prediction, they change the originally learnt network parameters for each prediction separately until the constraints are satisfied. Intuitively, they write a program which verifies if output satisfies certain linguistic constraints or not. If not, they take a gradient step to minimize the probability of the current incorrect prediction that violates the

constraints.

One common issue with constrained inference is that it slows the prediction process, which might not be desirable when deploying these models in real-world scenarios. Also, the model training could be sub-optimal as it is done without any knowledge of the inference process whereas at the time of prediction it is used along with inference. On the other hand, using the constraints at the time of training may provide an additional signal and result in a model which is 'constraint-aware'.

2.1.2 Constrained Training

In section 1.1.6.1 (item 3), we discussed different ways of incorporating known constraints while training. Here we focus on optimization-based methods that change the loss function or the training process to inject constraints while training the model.

2.1.2.1 Hard Constraints

Recently some attempts have been made to train neural networks with constraints by directly solving the constrained optimization problem [Ravi et al., 2018, Roy et al., 2018, Chen and Deng, 2013]. Ravi et al. [2018] propose a conditional gradient (CG) based method for constrained optimization. To incorporate geometric constraints on the weights, Roy et al. [2018] propose a Projected Gradient Descent (PGD) based modification of SGD with momentum [Sutskever et al., 2013] and RMSProp [Hinton et al., 2012] such that the weights always remain within the constrained manifold after each update. [Chen and Deng, 2013] employ a primal–dual formulation for optimizing with constraints in deep models. Their constraints are specified over the weights in a recurrent neural network and are only concerned with imparting stability to the overall learning algorithm. Note that all three methods enforce the constraints directly on the model parameters and are inspired by learnability instead of task-specific domain knowledge (see item 1 in section 1.1.2). In one of our works, we attempt to solve the constrained optimization directly, but for the constraints inspired by domain knowledge, and propose a primal-dual formulation to solve the constrained optimization problem.

2.1.2.2 Soft Constraints

Most of the works to enforce constraints while learning deep models focus on a soft formulation of constraints where a penalty proportional to constraint violation is added to the standard prediction loss such as cross entropy between the network prediction and ground truth labels [Kervadec et al., 2019, Oktay et al., 2018, Rocktäschel et al., 2015,

Xu et al., 2018].

For the medical 3D image segmentation task, [Kervadec et al., 2019] enforce size constraints on the expected number of pixels for a given label. It first assumes a lower and upper bound, (a, b), on the number of pixels for each label class and then adds a penalty equivalent to squared deviation from the bounds. If E_l represents the expected number of pixels for a given label l, then constraint loss is proportional to: $\mathbb{1}\{E_l < a\}(E_l - a)^2 + \mathbb{1}\{E_l > b\}(E_l - b)^2$. In an alternate formulation of similar constraints on size and shape, Oktay et al. [2018] learn an auto-encoder to encode the segmentation mask (target variable) in a lower 64-dimensional space and argue that the lower dimensional representation of the mask would capture the size and shape attributes. While training the segmentation network, they encode the predicted and ground truth masks and enforce the two encodings to be close to each other.

For the case of discrete labels, neural networks usually output probabilities over them. As a result, to use the constraints expressed on the discrete labels, one needs to convert them into an arithmetic expression involving probabilities over the labels. Fuzzy T-norm [Klement et al., 2004, Gupta and Qi, 1991] is often a method of choice for this conversion, resulting in a constraint function over the probabilities, which are a function of learnable network weights. Li et al. [2019] enforce consistency between the output labels in Natural Language Inference (NLI) task, which involves determining if a given hypothesis entails, contradicts, or is unrelated to a given premise. Given three sentences, S_1, S_2, and S_3, if S_1 entails S_2 and S_2 entails S_3, then a consistency constraint would enforce that S_1 should entail S_3. They do so by first converting the constraint over labels to constraint over probabilities using soft-logic and then adding, to the cross-entropy loss, an auxiliary loss term proportional to the constraint violation penalty. Similarly, for the joint task of temporal relation extraction and event hierarchy construction, [Wang et al., 2020] enforce anti-symmetry and transitivity constraints.

Instead of adding a constraint violation penalty based on soft–logic, posterior regularization [Ganchev et al., 2010] adapts the learned distribution in a post–facto manner to satisfy structural constraints over latent variables in expectation. To enforce constraints while training, [Pathak et al., 2015, Hu et al., 2016a,b, 2018] exploit it via distillation [Hinton et al., 2015]. It results in an alternating optimization technique where in the first step, a network's output is projected onto the space of probability distributions that satisfy the constraints in expectation. Such a rule regularized network is called a 'teacher network'. In the second step, information from the teacher network is distilled into a 'student network' by enhancing the likelihood based objective with a loss term proportional to the KL divergence between the output of student and teacher networks.

As an alternative to soft-logic for converting a Boolean expression into an expression

over probabilities, [Xu et al., 2018] propose 'semantic loss function' to express the logical constraints in the form of output probabilities. They enumerate the entire output state space and enforce that the probability of states that do not satisfy constraints should be zero. It is expressed in a single arithmetic statement by equating the sum of probabilities over all the states satisfying constraints to one. Specifically, let α be the boolean statement corresponding to the constraint over n output variables $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$. Let $\mathbf{y} = \{y_1, \ldots, y_n\}$ be an assignment of the n variables. If the assignment $\mathbf{y}$ satisfies the constraint, we represent it as: $\mathbf{y} \vDash \alpha$. Let $\mathbf{p} = p_1, \ldots p_n$ be the output probabilities given by the neural network for the variables $Y_1, \ldots, Y_n$, i.e. $p_i = Probability(y_i = 1)$. Then semantic loss is proportional to:

$$\sum_{\mathbf{y} \vDash \alpha} \prod_{\{i:y_i=1\}} p_i \prod_{\{i:y_i=0\}} (1 - p_i) \tag{2.1}$$

The sum iterates over all the possible states satisfying constraints, and the product is nothing but the probability of the state $\mathbf{y}$ as computed by the neural network. It is worth noting that this formulation implicitly assumes that the state probability factorizes over the n variables. To compute the sum of probabilities efficiently, they need to (pre-)compile a circuit for every constraint. However, computing these circuits can be NP–hard in many cases leading to intractability.

2.2 Unknown Constraints

Recently there has been a lot of focus on developing deep learning models for symbolic reasoning tasks. One such task involves solving combinatorial problems, which can be viewed as instances of a constraint satisfaction problem, albeit with unknown constraints. The task of the neural model is then to discover the unknown constraints using the training data consisting of many solved instances.

Based on the representation of unknown constraints (see fig. 1.1), we divide the related work into two buckets. In the first bucket, we focus on approaches that create a purely neural model that maps an input puzzle directly to its solution (section 2.2.1). In the second bucket, we delve deeper into approaches that explicitly invoke a symbolic reasoner, such as an Integer Linear Program (ILP) solver, a SAT solver, or a specific algorithm such as SDP relaxation for MAXSAT (section 2.2.2). The underlying task then transforms to learning the constraints explicitly in the language of the underlying solver, *e.g.*, linear inequalities for an ILP solver or Boolean expressions for a SAT / MAXSAT solver.

2.2.1 Neural Solvers with Implicit Constraint Representation

Here we describe two neural solvers, Neural Logic Machines (NLM) [Dong et al., 2019], and Recurrent Relational Networks (RRN) [Palm et al., 2018]), in detail. Both of them represent the constraints of an underlying combinatorial problem in their weights implicitly. In our work on solution multiplicity, we use both of them as the underlying neural model. While developing neural models for output space invariance, we build upon RRN and use NLM as a strong baseline.

2.2.1.1 Neural Logic Machines (NLM)

NLMs aim to learn predicates that can be represented as Horn clauses [Horn, 1951] over input predicates in First Order Logic (FOL). The input predicates are called *premises*, and the output predicates are called the *conclusions*. An input $\mathbf{x}$ to an NLM model is a list of predicates grounded over a universe of objects. The corresponding output $\mathbf{y}$ is another list of predicates grounded over the same set of objects. For example, consider a relational reasoning task over family trees, where the input consists of unary predicates $IsBoy(o)$ and $IsGirl(o)$, and a binary predicate $IsParentOf(o_1, o_2)$ and the objective is to learn binary predicates $IsDaughterOf(o_1, o_2)$ and $IsSonOf(o_1, o_2)$. Here o, o_1, o_2 represent the objects in the universe, *i.e.*, members of a family. Each instance $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ in the training data would be created from a family tree $T^{(i)}$ over n_i members, with $\mathbf{x}^{(i)}$ consisting of the two unary predicates ($IsBoy(o)$ and $IsGirl(o), o \in \{1, \ldots, n_i\}$) grounded over the n_i family members (objects), and a binary predicate $IsParentOf(o_1, o_2), o_1, o_2 \in \{1, \ldots, n_i\}$ grounded over the n_i^2 pairs of objects or family members. The output would consist of two binary predicates $IsDaughterOf(o_1, o_2)$ and $IsSonOf(o_1, o_2)$ grounded over the same n_i^2 pairs. Notice that each input to an NLM model can be grounded over a different universe (different families in the above example) of a possibly different size, thereby making it amenable to output-space invariance.

An NLM with 'depth' D consists of a stack of D layers, and each layer takes a list of predicates as input and outputs another list of predicates, which feeds into the next layer. Hence, an intermediate layer may learn auxiliary predicates, which may assist in learning the output predicates. The maximum arity B of a predicate in any layer is called the 'breadth' of the NLM architecture.

To represent a $k-$ary predicate over a universe of n objects, NLM uses a $k-$dimensional tensor of size n^k. [1] Further, all predicates of the same arity k are grouped together into a

[1] The exact shape of the tensor is $[n^{\underline{k}}] \triangleq n \times n-1 \times \cdots \times n-k+2 \times n-k+1$ because NLM does not allow an argument to repeat in a predicate, *i.e.*, $p(o, o)$ is not allowed for a binary predicate p. This is because such a predicate can be represented by another unary predicate $p'(o) \triangleq p(o, o)$. We ignore this implementation detail for the sake of simplicity.

single $(k+1)-$dimensional tensor by stacking all the $k-$dimensional predicate tensors. Thus, the output of the j^{th} layer L_j (or input to the $(j+1)^{th}$ layer) is a list l_j of $(B+1)$ tensors: one for each arity upto maximum arity B, and one for nullary predicates. The k^{th} element of the list l_j, *i.e.*, $l_j[k], k \in \{0, \ldots, B\}$, is a $(k+1)-$dimensional tensor of size $n_k^{(j)} n^k$, representing a total of $n_k^{(j)}$ predicates, each of size n^k and arity k. Note that NLM takes a probabilistic view of the predicates, *i.e.*, the tensors contain values between $[0, 1]$, representing the probabilities of the predicate evaluating to true for the corresponding arguments. [2]

Computation in each layer: Two types of computations occur in each layer: inter–arity, followed by intra–arity.

1. **Inter–arity computations:** Here we creates intermediate $k-$arity predicates from each of the input $(k-1)-$arity and $(k+1)-$arity predicates using the **expansion** and **reduction** rules, respectively, as defined next. The intuition is to allow $(k-1)$ and $(k+1)-$arity predicates to participate in the discovery of new $k-$arity predicates that happens during intra–arity computations. To do so, we expand the dimension of $(k-1)-$arity predicates and reduce the dimension of $(k+1)-$arity predicates. During expansion of a $(k-1)-$arity predicate, its corresponding $(k-1)-$dimensional tensor is repeated n times to represent a new $k-$arity predicate:

$$\forall o_k \quad \tilde{p}(o_1, \cdots, o_{k-1}, o_k) \leftarrow \quad p(o_1, \cdots, o_{k-1}).$$

   In reduction, a $(k+1)-$dimensional tensor corresponding to a $(k+1)-$arity predicate is reduced to two $k-$dimensional tensors by taking a min and a max along the last dimension, resulting in two new $k-$arity predicates defined as:

$$p'(o_1, \cdots, o_k) \leftarrow (\forall o_{k+1} \quad p(o_1, \cdots, o_{k+1})) \quad \text{and}$$
$$p''(o_1, \cdots, o_k) \leftarrow (\exists o_{k+1} \quad p(o_1, \cdots, o_{k+1})) \quad \text{respectively.}$$

   As done in T-norms, min and max operators here are used as a continuous proxy for AND and OR Boolean operators, represented by $\forall$ and $\exists$ quantifiers, respectively. After the inter–arity computations in j^{th} layer, we now have a total of $n_k'^{(j)} = n_{k-1}^{(j-1)} + 2n_{k+1}^{(j-1)} + n_k^{(j-1)}$ $k-$arity predicates represented as a $(k+1)$ dimensional tensor. Let us denote this tensor by $l'_j[k]$. Next, in the intra–arity computations, we discuss how we use $l'_j[k]$ to learn new $k-$arity predicates that form the output, $l_j[k]$, of the j^{th} layer.

2. **Intra–arity computations:** Here we use all the $k-$arity predicates, obtained after inter–arity computations in a layer, to express the layer's output predicates with the same arity. As a first step, we further expand the set of $k-$arity predicates by

[2] In practice, the predicates are true for very few sets of arguments, resulting in sparse tensors.

considering all the $k!$ possible permutations of its k arguments. Specifically, for a given permutation σ of $\{1, \ldots, k\}$, a new predicate is defined as:

$$p'''(o_1, \cdots, o_k) \leftarrow p(o_{\sigma(1)}, \cdots, o_{\sigma(k)})$$

We get a total of $k! n_k'^{(j)}$ predicates with arity k after applying permutation operation to all $n_k'^{(j)}$ predicates obtained after inter–arity computations. Let us denote this set of predicates by P_k^l.

Finally, we use all the $k-$arity predicates obtained after the permutation step to define new learnable predicates. For this, we use the power of an MLP to represent a complex Boolean expression involving logical operators (AND, OR, NOT) and many input Boolean variables. Specifically, an output $k-$arity predicate, $p'(o_1, \cdots, o_k)$, is evaluated by an MLP operating on $\{p(o_1, \cdots, o_k) | p \in P_k^l\}$. The number of input neurons in the MLP is the total number of k–arity predicates obtained after the permutation step, *i.e.*, $|P_k^l| = k! n_k'^{(j)}$.

We make two crucial observations here: (1) The MLP in the intra–arity computations applies separately on all possible arguments, $(o_1, \cdots, o_k), o_1, \cdots, o_k \in \{1, \ldots, n\}$, of the predicate. As a result, NLMs can handle predicates defined on objects from different universes of potentially different sizes. This makes them amenable to output–space invariance. (2) In each layer, a new $k-$arity predicate is influenced by only $k-1, k$, and $k+1-$arity predicates. Hence, for a $k'-$arity predicate to influence the discovery (or learning) of a new $k-$arity predicate, we must have at least $|k-k'|$ layers in the network.

2.2.1.2 Recurrent Relational Networks (RRNs)

RRNs are message passing based graph neural networks (GNNs) [Zhou et al., 2020] with message passing weights shared across different layers. The weight sharing across layers allow RRNs to have sufficiently large depth without blowing up the number of network parameters. Apparently, large depth is crucial for reasoning tasks, such as solving sudoku, which may require multiple message passing steps. At the end of message passing, the objective is to classify each graph node (node classification), or the entire graph (graph classification), into one of the desired classes based on the final state of each node. A recurrent message passing network primarily involves three steps: 1) graph creation and initialization, 2) recurrent message passing and state update for many steps, say T, and 3) value decoding. We describe each of them briefly, using the standard 9×9 sudoku as our running example, modeled as a node classification task.

Graph creation and initialization: The nodes of the message passing graph, $G = (N, E)$, represent the input variables and the edges represent existence of a relationship

between nodes. N and E denote the set of all nodes and edges, respectively. Nodes, $n_j \in N$, are initialized based on their initial state, $u_0(n_j) \in \mathbb{R}^d$, represented as a dense vector in the d dimensional real space $\mathbb{R}^d$. For example, in the graph corresponding to a sudoku puzzle, a node represents a sudoku cell, connected with all the nodes representing other cells in the same row, column and grid. Each of the 10 digits in $\{0, \ldots, 9\}$, including digit 0 for empty cells, is represented as a learnable d dimensional embedding. The initial state of a node is determined by the embedding of the digit in the corresponding cell in the input sudoku puzzle. In the case of visual sudoku, this initial state is the output of a CNN which encodes image of each cell into a d dimensional vector.

Recurrent message passing and state update: At each message passing step, a node receives messages from its neighbours and updates its own state. Specifically, if $e_{i,j}$ represents an edge from node n_i to n_j, then node n_j first receives a message, $m_t[e_{i,j}]$ on each incoming edge. These messages are aggregated into a single message $m_t[n_j]$. This message, along with the current state $h_t(n_j)$, and the initial state, $u_0(n_j)$, determines the next state, $h_{t+1}(n_j)$:

$$\begin{aligned} m_t[e_{i,j}] =& f(h_t(n_i), h_t(n_j)), \quad \forall e_{i,j} \in E \\ m_t[n_j] =& \sum_{e_{i,j} \in E} m_t[e_{i,j}] \quad \forall n_j \in N \\ h_{t+1}(n_j) =& g\left(h_t(n_j), m_t[n_j], u_0(n_j)\right), \forall n_j \in N \end{aligned}$$

Palm et al. [2018] use an MLP as the message passing function f and an LSTM [Hochreiter and Schmidhuber, 1997] as the state update function g. Notably, passing $u_0(n_j)$ as part of input at every step t ensures that the model doesn't forget its initial state.

Value decoding: To make the prediction, the final state, $h_T(n_j)$, for each node, n_j, is passed through a decoder function, r, to compute $o_{T,j} = r(h_T(n_j))$. It represents a probability distribution over the label space $\mathcal{V}$, *i.e.*, $o_{T,j}(v) = Pr_T(n_j = v), \forall v \in \mathcal{V}, \forall n_j \in N$. For example, probabilities over digit in $\mathcal{V} = \{1, \ldots, 9\}$ for each sudoku cell. The digit with maximum probability is then selected as the prediction for that cell. Palm et al. [2018] model r as an MLP with a softmax non–linearity at the final layer.

Loss computation during training: To speed up the training process, instead of computing loss using only the final predictions $o_{T,j}$, Palm et al. [2018] propose to decode the hidden state at each message passing step t and compute the cross entropy loss *w.r.t.* $o_{t,j}, \forall t \in \{1, \ldots, T\}$:

$$L = -\frac{1}{T} \sum_{t=1}^{t=T} \sum_{n_j \in N} \sum_{v \in \mathcal{V}} \mathbb{1}\{n_j = v\} \log o_{t,j}(v) = -\frac{1}{T} \sum_{t=1}^{t=T} \sum_{n_j \in N} \sum_{v \in \mathcal{V}} \mathbb{1}\{n_j = v\} \log Pr_t(n_j = v)$$

2.2.2 Symbolic Solvers with Explicit Constraint Representation

Paulus et al. [2021] recently proposed a method for backpropagating through an ILP layer in a neural-ILP architecture. In chapter 6, we propose a scalable alternative and CombOptNet acts as our baseline. Hence we describe it in detail in this section.

2.2.2.1 CombOptNet: Learning Integer Programming Constraints

Both NLM and RRN train a neural model that maps an input puzzle directly to its solution. Another alternative is to learn the underlying constraints and the objective function explicitly and then invoke a symbolic solver that takes them as input and finds the solution. For example, many combinatorial problems, such as sudoku, can be expressed as an Integer Linear Program (ILP). So one may learn a neural function that maps an input puzzle to the linear cost vector, $\mathbf{c}$, and linear constraints, represented by matrix $\mathbf{A}$ and vector $\mathbf{b}$, of the underlying ILP. $\mathbf{c}, \mathbf{A}$ and $\mathbf{b}$ are then consumed by an ILP solver, such as Gurobi [Gurobi Optimization, LLC, 2022], to find the solution, $\hat{\mathbf{y}}(\mathbf{c}, \mathbf{A}, \mathbf{b})$, by solving the underlying ILP:

$$\hat{\mathbf{y}}(\mathbf{c}, \mathbf{A}, \mathbf{b}) = \arg\min_{\mathbf{z} \in \mathcal{Y}} \mathbf{c}^T \mathbf{z} \quad \text{subject to} \quad \mathbf{A}\mathbf{z} + \mathbf{b} \geq \mathbf{0}, \mathbf{z} \in \mathbb{Z}^n, \mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{c}, \mathbf{b} \in \mathbb{R}^n$$

Note that we represent the solution $\hat{\mathbf{y}}$ as a function mapping $\mathbf{c}, \mathbf{A}$ and $\mathbf{b}$ to the solution, but computing this function requires solving an optimization problem. In the above equation, $\mathcal{Y}$ is a known bounded subset of $\mathbb{Z}^n$.

When training such a neuro-symbolic model in an end-to-end fashion, the main issue is computing the gradients of loss *w.r.t.* $\mathbf{c}, \mathbf{A}$ and $\mathbf{b}$. At most of the points, the optimal solution of the ILP does not change with small changes in $\mathbf{c}, \mathbf{A}$ and $\mathbf{b}$, and hence gradient is zero at all those points. On the other hand, when the optima changes, it jumps discretely to another corner point of the feasible region, making the gradient un–defined.

Paulus et al. [2021] propose a useful approximation for the gradients *w.r.t.* the ILP parameters $\mathbf{c}, \mathbf{A}$ and $\mathbf{b}$. They observe that the derivative of the loss *w.r.t.* the current solution $\hat{\mathbf{y}}$ gives us a desired descent direction in which the optima should move after updating the ILP parameters. Let $\hat{\mathbf{y}} - \mathrm{d}\hat{\mathbf{y}}$ denote the desired optima after gradient updates. Their main idea is to change the ILP parameters such that $\hat{\mathbf{y}}$ is no longer optimal and $\hat{\mathbf{y}}' = \hat{\mathbf{y}} - \mathrm{d}\hat{\mathbf{y}}$ becomes the new optima, *i.e.*, it satisfies all the constraints $(\mathbf{a}_i^T \hat{\mathbf{y}}' + \mathbf{b}_i \geq 0)$, and has the minimum objective $(\mathbf{c}^T \hat{\mathbf{y}}')$. Here $\mathbf{a}_i$ is the i^{th} row of the constraint matrix $\mathbf{A}$. However, the desired optima $\hat{\mathbf{y}}' = \hat{\mathbf{y}} - \mathrm{d}\hat{\mathbf{y}}$ may not be integeral. Hence they propose to decompose $\mathrm{d}\hat{\mathbf{y}}$ as: $\mathrm{d}\hat{\mathbf{y}} = \sum_{k=1}^{n} \lambda_k \Delta_k$, where $\Delta_k \in \{-1, 0, 1\}^n$ and $\lambda_k \in \mathbb{R}, \lambda_k \geq 0$. Intuitively, it is being expressed as a linear combination of its neighbours. Now the problem reduces to computing derivatives using $\hat{\mathbf{y}}'_k = \hat{\mathbf{y}} - \Delta_k$ as the desired optima of the ILP, and these

derivatives are then combined linearly using $\lambda_k, k \in \{1, \ldots, n\}$.

Gradient updates of the constraint parameters A and b: Let $d(\mathbf{z}; [\mathbf{a}_i|\mathbf{b}_i]) = \frac{\mathbf{a}_i^T \mathbf{z} + \mathbf{b}_i}{|\mathbf{a}_i|}$ represent the signed Euclidean distance of a point $\mathbf{z} \in \mathbb{R}^n$ from the hyperplane corresponding to the i^{th} constraint, represented as, $[\mathbf{a}_i|\mathbf{b}_i]$. Paulus et al. [2021] use this signed Euclidean distance of $\hat{\mathbf{y}}'_k$ and $\hat{\mathbf{y}}$ from each of the m constraints to determine the degree, $P_{\Delta_k}(\mathbf{A}, \mathbf{b})$, to which $\hat{\mathbf{y}}'_k$ and $\hat{\mathbf{y}}$ are infeasible and feasible, respectively. They call it a piecewise affine local *mismatch* function. The gradient updates *w.r.t.* $\mathbf{b}$ and $\mathbf{A}$ are defined as the gradient of P_{Δ_k} *w.r.t.* $\mathbf{b}$ and $\mathbf{A}$. The objective is to make $\hat{\mathbf{y}}'_k$ feasible if it is infeasible, and make $\hat{\mathbf{y}}$ infeasible if $\hat{\mathbf{y}}'_k$ is already feasible. Let S_k represent the set of constraints which are violated for $\hat{\mathbf{y}}'_k$, *i.e.*, $S_k = \{j | d(\hat{\mathbf{y}}'_k; [\mathbf{a}_i|\mathbf{b}_i]) < 0\}$. Our objective is to increase all these signed distances, or decrease its negative ($\sum_{j \in S_k} -d(\hat{\mathbf{y}}'_k; [\mathbf{a}_j|\mathbf{b}_j])$), so that $\hat{\mathbf{y}}'_k$ becomes feasible. To make $\hat{\mathbf{y}}$ infeasible, we try to reduce the distance of the closest hyperplane ($\min_j d(\hat{\mathbf{y}}; [\mathbf{a}_j|\mathbf{b}_j])$) as the corresponding constraint is most likely to be violated. If $\hat{\mathbf{y}}'_k == \hat{\mathbf{y}}$, or $\hat{\mathbf{y}}'_k \notin \mathcal{Y}$, then we do not want any gradients to flow. The following equation captures these intuitions:

$$P_{\Delta_k}(\mathbf{A}, \mathbf{b}) = \begin{cases} \sum\limits_{j \in S_k} -d(\hat{\mathbf{y}}'_k; [\mathbf{a}_j|\mathbf{b}_j]) + \mathbb{1}\{\hat{\mathbf{y}}'_k \neq \hat{\mathbf{y}}\} \min\limits_{j \in \{1,\ldots,n\}} d(\hat{\mathbf{y}}; [\mathbf{a}_j|\mathbf{b}_j]) & \text{if } \hat{\mathbf{y}}'_k \in \mathcal{Y} \\ 0 & \text{if } \hat{\mathbf{y}}'_k \notin \mathcal{Y} \end{cases}$$

Once $P_{\Delta_k}(\mathbf{A}, \mathbf{b}), \forall k \in \{1, \ldots, n\}$ are defined, the gradients of the loss L *w.r.t.* $\mathbf{A}$ and $\mathbf{b}$ are nothing but their linear combination using $\lambda_k, k \in \{1, \ldots, n\}$:

$$\frac{\partial L}{\partial \mathbf{A}} = \sum_{k=1}^{n} \lambda_k \frac{\partial P_{\Delta_k}}{\partial \mathbf{A}}(\mathbf{A}, \mathbf{b}) \quad \text{and} \quad \frac{\partial L}{\partial \mathbf{b}} = \sum_{k=1}^{n} \lambda_k \frac{\partial P_{\Delta_k}}{\partial \mathbf{b}}(\mathbf{A}, \mathbf{b})$$

Gradient updates of the cost parameters c: Here the intuition is to make $\hat{\mathbf{y}}'_k = \hat{\mathbf{y}} - \Delta_k$ optimial by minimizing its objective value without impacting the objective value of $\hat{\mathbf{y}}$. In addition, they do so only when $\hat{\mathbf{y}}'_k$ is feasible, otherwise, set gradients *w.r.t.* $\mathbf{c}$ as 0:

$$P_{\Delta_k}(\mathbf{c}) = \begin{cases} -\mathbf{c}^T \Delta_k & \text{if } \hat{\mathbf{y}} \neq \hat{\mathbf{y}}'_k \\ 0 & \text{if } \hat{\mathbf{y}}'_k \notin \mathcal{Y} \text{ or } \hat{\mathbf{y}}'_k \text{ is infeasible} \end{cases}$$

Once $P_{\Delta_k}(\mathbf{c})$ is defined, the gradients *w.r.t.* $\mathbf{c}$ are computed similar to gradients *w.r.t.* $\mathbf{A}$ or $\mathbf{b}$:

$$\frac{\partial L}{\partial \mathbf{c}} = \sum_{k=1}^{n} \lambda_k \frac{\partial P_{\Delta_k}}{\partial \mathbf{c}}(\mathbf{c})$$

Part II

Explicitly Known Constraints

Chapter 3

Learning with Known Constraints

In Chapter 1 we observed that output(s) of a neural model are often symbolic and in many cases, there may exist some natural constraints involving those output(s). We further observed that these constraints could be either *known* or *unknown*. In this chapter, we will focus on the case of known constraints that are *explicitly* provided to us in the form of a Boolean or an arithmetic expression over the output variables.

Consider the the joint task of NER and POS labeling. A constraint on the output of a joint model for NER and POS tagging might specify that the NER label 'organization' is consistent only with the POS labels 'noun' and 'preposition'. These explicit constraints can be a great way of injecting prior knowledge into a deep learning model, thereby improving overall performance. In this chapter, we present a constrained optimization formulation for training a deep network with a given set of hard constraints on output labels. Our novel approach first converts the label constraints into soft logic constraints over probability distributions outputted by the network. It then converts the constrained optimization problem into an alternating min-max optimization with Lagrangian variables defined for each constraint. Since the constraints are independent of the target labels, our framework easily generalizes to semi-supervised setting. We experiment on the tasks of Semantic Role Labeling (SRL), Named Entity Recognition (NER) tagging, and fine-grained entity typing and show that our constraints not only significantly reduce the number of constraint violations, but can also result in state-of-the-art performance.

3.1 Introduction

Deep learning models are typically trained end–to–end from data, without additional knowledge about the task (other than neural architecture and loss function). However, for many problems of interest (e.g., structured prediction or multi-task learning), there is a set

of natural constraints which need to be satisfied over the output variables. For example, for the task of NER and POS labeling, the constraint might specify that a word which is given the NER label 'institution' must have the POS label 'noun' or a 'preposition'. Or in 3D human pose estimation from a single view, one may impose symmetry constraints, like equal length of two arms, equal distance of shoulders from the spine etc. ([Márquez-Neila et al., 2017]). These constraints can be seen as additional background knowledge made available by the domain experts. Incorporating these constraints into a model can presumably regularize the output space resulting in improved predictions.

One line of work trains the neural models without this knowledge, but imposes constraints at inference time [Lee et al., 2019]. We argue that this is bound to be sub-optimal since the original training of the network was done in oblivion to the constraints. Though the deep network, in principle, could learn these directly from the data, but, in practice, this is true only when the available training data is large. Our experiments reveal that such unconstrained models result in a large number of constraint violations, especially when trained in low data settings. On the other hand, modeling the constraints explicitly during training gives a strong prior to the model – it not only reduces constraint violations but can also result in significantly improved predictions by making the training *constraint-aware*.

In this chapter, we present a principled solution to the problem of learning a deep network with a given set of hard constraints on output labels. We first formulate this as a constraint optimization problem wherein we maximize the original learning objective (*e.g.*, cross entropy) subject to the constraints being satisfied for each example in the training data. When the network makes predictions in form of probabilities over the output variables, we can rewrite the constraints on output labels as constraints on probabilities output by the network using soft logic ([Bröcheler et al., 2010], [Novák, 1987]). This rewrite can be seen as imposing constraints over the set of allowed distributions over the output space, i.e., allowing only those distributions that satisfy the constraints. We then convert this problem into a Lagrangian formulation, with one Lagrange variable per constraint. We solve it using alternating min-max based optimization. Though the resulting problem can be highly non-convex non-concave, convergence guarantees to a local min-max point (in the limit) follow from the theory of min-max optimization ([Jin et al., 2019]). Since our constraints are specified over the predicted variables (no ground truth labels are involved), our formulation easily extends to semi-supervised setting where the unlabeled data only contributes to constraint terms in the formulation.

We note that there have been a few recent attempts at adding constraints during training time, but with significant differences from our proposed approach. These include the work by Xu et al. [2018], Mehta et al. [2018] and Diligenti et al. [2017b]. While these

existing approaches model the constraints as soft and incorporate a constraint violation penalty directly in the loss term, our constraints are modeled as hard, and we resort to a full Langrangian based optimization. The existing work can require an exponential sum to be computed ([Xu et al., 2018]), require an additional constraint violation penalty as input with no explicit convergence guarantees ([Mehta et al., 2018]), or require a specific functional form for the constraints ([Diligenti et al., 2017b]). In contrast, our formulation is tractable because we do not compute an exponential sum, requires no additional constraint violation term, converges to the stationary point of the objective function, and does not assume a specific functional form for the constraints. We detail these differences in the related work.

We experiment on three different NLP tasks: (a) semantic role labeling, (b) NER tagging, and (c) fine grained entity typing over hierarchical label space. Our experiments clearly demonstrate that, in low data setting, our constraint based learning not only reduces the number of violations, but also result in significantly improved prediction accuracy compared to unconstrained baselines as well as vanilla post processing of constraints at inference time. Semi-supervised learning results in further improvements using our formulation. Furthermore, in some cases, our approach altogether eliminates the need for post-processing of constraints, since they have already been learned by the neural model. For two of the tasks, we obtain state-of-the-art results for small data sizes. On NER, our constrained learning completely eliminates the need for further post-processing with constraints, saving on precious inference time.

Our contribution in this chapter can be summarized as follows: (1) We present a principled approach for incorporating domain knowledge in the form of hard constraints. We present a Lagrangian based formulation for learning with constraints in a deep network. Our constraints make use of soft logic to deal with logical operators. (2) We employ a min-max based optimization to solve our constrained formulation. To the best of our knowledge, we are the first to use a min-max based formulation for learning with constraints specified over output variables in deep networks. Convergence to local min-max points ([Jin et al., 2019]) in the limit follows from the theory of min-max optimization. (3) We present experimental results on three different tasks demonstrating the effectiveness of our approach, while achieving state-of-the-art results in two of the domains and also significantly reducing the number of constraint violations in each case.

Outline: In this chapter, we first begin with a motivation for enforcing constraints on the outputs of machine learning models and argue about advantages of *learning with constraints* over *constrained inference* in section 3.1. In section 3.2 we draw a comparison between our method and other recent approaches for constrained learning of deep models. Section 3.3 presents our Lagrangian based primal-dual formulation along with a way of converting

constraints over discrete labels to constraints over probabilities of the discrete labels as outputted by the model. To solve the constrained optimization problem, section 3.4 converts the primal-dual formulation into a concrete algorithm with alternating min-max steps: minimizing *w.r.t.* weights of the network and maximizing *w.r.t.* the Lagrange variables. We present the experimental results in section 3.5 where we experiment over three different NLP tasks. Finally, we present a short summary and a conclusion in section 3.6.

3.2 Related Work

In section 2.1 we already discussed different ways of enforcing known constraints on the output of a neural model. The focus of our work is learning with constraints and accordingly, here we compare our work with latest approaches for enforcing constraints while learning [Hu et al., 2016a, Xu et al., 2018, Mehta et al., 2018, Diligenti et al., 2017b].

Hu et al. [2016a] perform posterior regularization over the weights being learned at each step, so that the resultant distribution satisfies a given set of logical rules (section 2.1.2.2). Their work is different from ours in two main aspects. (1) The imposition of constraints in their algorithm is only indirect by mimicking the rule regularized network. In contrast, we optimize for satisfying the constraints directly. (2) They model constraints as soft whereas we deal with hard constraints. Further, they could achieve limited success in their experiments.

Xu et al. [2018] incorporate constraints by forcing the probability of states violating the constraints to zero (section 2.1.2.2). They model this as a soft constraint by incorporating a constraint violation penalty in the loss function. In contrast, we model constraints as hard through the use of Lagrangian. Our formulation disallows the non-satisfying states directly through the use of constraints, and unlike them, does not require an exponential sum to be computed.

[Mehta et al., 2018] present an approach for learning with constraints and demonstrate the effectiveness of their method for the specific task of Semantic Role Labeling (SRL). They model the constraints by adding another term to the loss which penalizes the currently predicted state for violating the constraint. They require an additional constraint violation penalty from the designer. Their approach can be seen as a local search in the weight space, so that the resultant weights result in a satisfying assignment. They do not have a global metric which optimizes the weights for satisfying the constraints (e.g, forcing the non-satisfying states to zero probability), and it not clear if their algorithm will converge in the limit. In contrast, we can provide convergence guarantees to min-max points of the

objective and we experiment on a variety of NLP tasks.

[Diligenti et al., 2017b] propose an approach for learning with constraints, where the constraints are specified as logical formulas. Though their approach seems similar to ours at the outset, there are some important differences. Unlike us, they do not work with full Lagrangian formulation. Their approach is simply modifying the loss, and can not handle hard constraints. Further, they require the constraint function to be of specific form (i.e., functionals in the range $[0, 1]$), and present experiments only on a single task. Our work makes no such assumption about the form of constraints and we experiment on a variety of tasks.

3.3 Constrained Learning of Neural Models

In this section, we first set up our problem statement, followed by a presentation of our primal-dual formulation for learning with constraints. The special case of discrete output labels is discussed in section 3.3.2, where we use soft-logic to convert a constraint, expressed as a Boolean expression on discrete outputs, to an arithmetic expression involving probabilities over the same discrete variables.

Problem Statement: Consider learning a neural network over a set of training examples given as $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=i}^{m}$. Each $\mathbf{x}^{(i)} \in \mathbb{R}^n$ represents an n-dimensional feature vector in a real valued space. Each $\mathbf{y}^{(i)} \in \mathcal{V}^r$ represents an r dimensional target (label) vector, where each element of the vector takes values from a discrete (or continuous) valued space denoted by $\mathcal{V}$. Note that r may be input-dependent, *e.g.*, in sequence labeling tasks. Given the parameters $\mathbf{w}$ of the network, let $l_{\mathbf{w}}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$ denote the loss obtained by predicting $\hat{\mathbf{y}}^{(i)}$ when the target is $\mathbf{y}^{(i)}$. For instance, this could be the cross entropy loss when the labels are discrete. The goal of learning is to find a set of parameters $\mathbf{w}^*$ of the network such that the average loss $L(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} l_{\mathbf{w}}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$ between the network outputs $\hat{\mathbf{y}}^{(i)}$ and the target values $\mathbf{y}^{(i)}$ is minimized, *i.e.*, $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\, L(\mathbf{w})$.

In this work, we are interested in a scenario where we are additionally provided with a set of (hard) constraints which hold over the output label space. We assume that these constraints are provided to us by the domain experts and are available in the form of background knowledge. Our goal is to incorporate this background knowledge to learn a more robust and generalizable model. Our formulation is based on constructing a Lagrangian, which tries to minimize the original objective subject to the given constraints. We solve our problem using an alternating optimization over a max-min formulation.

3.3.1 A Lagrangian-based Formulation

Let us assume we are given a set of K constraints as $\{C_1(\hat{\mathbf{y}}), C_2(\hat{\mathbf{y}}), \cdots, C_K(\hat{\mathbf{y}})\}$. We will use index k to vary over the constraint set. Each constraint is a function of the predicted values $\hat{\mathbf{y}}$ on a given example $\mathbf{x}$. Since each of the network outputs in turn is directly a function of weights $\mathbf{w}$ of the network (for a given $\mathbf{x}$), for ease of notation, we will simply write the constraint set to be $\{C_1(\mathbf{w}), C_2(\mathbf{w}), \cdots, C_K(\mathbf{w})\}$. Note that the dependence on input vector $\mathbf{x}$ is implicit in this notation. Further, without loss of generality, we will assume that each of our constraints $C_k(\mathbf{w})$ is expressed as an inequality constraint over an appropriately defined function $f_k(\mathbf{w})$, i.e., $C_k(\mathbf{w}) : \{f_k(\mathbf{w}) \leq 0\}$. This can also model constraints of the form $\overline{f}_k(\mathbf{w}) = 0$ by replacing them with two inequality constraints of the form $\overline{f}_k(\mathbf{w}) \leq 0$ and $-\overline{f}_k(\mathbf{w}) \leq 0$.

When dealing with constraints over the set of m training examples, we will incorporate the dependence on the i^{th} example by including the index in the super script, i.e., we will denote the k^{th} constraint over the i^{th} example as $C_k^i(\mathbf{w})$. We are now ready to define our constrained formulation.

$$\text{argmin}_{\mathbf{w}}\, L(\mathbf{w}) \;\; \text{subject to} \quad f_k^i(\mathbf{w}) \leq 0; \;\; \forall 1 \leq i \leq m; \;\; \forall 1 \leq k \leq K. \tag{3.1}$$

One problem with the above formulation is that it has $O(mK)$ number of constraints. In particular, the number of constraints grows linearly with number of examples, which may become unwieldy. We use the following trick to reduce the number of constraints. Since we are only interested in eliminating the states that do not satisfy the constraints, we can in fact ignore the value that the function $f_k^i(\mathbf{w})$ takes when the corresponding constraint is satisfied. Accordingly, we define the Hinge function $H : \mathbb{R} \to \mathbb{R}$ as: $H(c) = c \;\; for \;\; c \geq 0, \;\; \text{and} \;\; 0 \;\; for \;\; c < 0$.

We equivalently replace each constraint of the form $f_k^i(\mathbf{w}) \leq 0$ by $H(f_k^i(\mathbf{w})) = 0$ without changing the original formulation. Intuitively, $H(f_k^i(\mathbf{w}))$ can be thought of as describing the loss incurred when the corresponding constraint is not satisfied. This loss is zero when the constraint is satisfied. This transformation will be useful in the next step when we combine together instances of a single type of constraint applied to different examples in the training set. In the new formulation, we get our primal objective as:

$$\text{argmin}_{\mathbf{w}}\, L(\mathbf{w}) \;\; \text{subject to} \quad H(f_k^i(\mathbf{w})) = 0; \;\; \forall 1 \leq i \leq m; \;\; \forall 1 \leq k \leq K. \tag{3.2}$$

Clearly, $H(f_k^i(\mathbf{w})) \geq 0, \forall i, k$ by definition. Therefore, a necessary and sufficient condition to enforce $\forall i : H(f_k^i(\mathbf{w})) = 0$ is $\sum_i H(f_k^i(\mathbf{w})) = 0$. This is true for all k. Defining $h_k(\mathbf{w}) = \sum_i H(f_k^i(\mathbf{w}))$, we can therefore write our primal objective in Section 3.3.1 as:

$$\text{argmin}_{\mathbf{w}}\, L(\mathbf{w}) \;\; \text{subject to} \quad h_k(\mathbf{w}) = 0; \;\; \forall 1 \leq k \leq K. \tag{3.3}$$

A standard way of solving the optimization problem described in Equation 3.3 is to find a

stationary point of the corresponding Lagrangian, $\mathcal{L}$

$$\mathcal{L}(\mathbf{w}; \Lambda) = L(\mathbf{w}) + \sum_{k=1}^{K} \lambda_k h_k(\mathbf{w}) \tag{3.4}$$

Here, $\Lambda = \{\lambda_k\}_{k=1}^{K}$ denotes the K sized vector of Lagrange multipliers. Note that since $h_k(\mathbf{w})$ is always non-negative, the constraint $h_k(\mathbf{w}) = 0$ is equivalent to $h_k(\mathbf{w}) \leq 0$. Hence the Lagrange multipliers are always non-negative. Our optimization problem in the primal can be written as:

$$\min_{\mathbf{w}} \max_{\Lambda} \mathcal{L}(\mathbf{w}, \Lambda) \tag{3.5}$$

Instead of solving the primal in (3.5), we often solve its corresponding dual:

$$\max_{\Lambda} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \Lambda) \tag{3.6}$$

We make two comments on our formulation. First, the use of the hinge function achieves two objectives: (a) no penalty is paid when constraints are satisfied, and (b) the number of dual variables is reduced from $O(mK)$ to $O(K)$, making the formulation scalable. Second, our formulation can handle arbitrary constraints as long as they are differentiable. Note that even simple form of constraints (such as linear) over the output variables typically represent highly non-linear functions of the networks weights.

3.3.2 Constraint Language for Discrete Output Spaces

A common learning scenario for many problems is when each element of the target $\mathbf{y}$ belongs to a discrete space. In such cases, each $\mathbf{y}$ is given as a vector $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_r)$, where each $\mathbf{y}_j \in \mathcal{V}$ where $\mathcal{V}$ represents a set of discrete values. The network output is then represented as an r dimensional vector, where each element of the vector represents a probability distribution over $\mathcal{V}$, i.e. $P_{\mathbf{w}}(\mathbf{y}_j|\mathbf{x})\ \forall j, 1 \leq j \leq r$. One of the common loss functions for discrete spaces is the cross entropy based loss though other loss functions can also be used. At prediction time, given a new test example $\mathbf{x}$, we output the vector of values which have the highest probability for each element $\mathbf{y}_j$ in the output space, i.e., $\text{argmax}_{\mathbf{y}_j \in \mathcal{V}} P_{\mathbf{w}}(\mathbf{y}_j|\mathbf{x}), \forall j, 1 \leq j \leq r$. In this section, we lay out the details of a language which can handle logical constraints specified over discrete output spaces as described above. Our formulation is based on soft logic used earlier in the literature [Bröcheler et al., 2010], and represents constraints in the form of inequalities: $f_k(\mathbf{w}) \leq 0$ where $\mathbf{w}$ are network weights.

Our constraints are defined as logical expressions over values v that each $\mathbf{y}_j$ can take. A constraint C can take the following form: (a) $C : \mathbb{1}\{\mathbf{y}_j = v\}$ (b) $C = \neg C_1$ (c) $C = C_1 \vee C_2$ (d) $C = C_1 \wedge C_2$. Here, C_1, C_2 denote constraints constructed recursively using above rules. The first expression (a) can be thought of as an atomic constraint, and rest are constructed

Constraint: C	$g(\mathbf{w})$**: Choice 1**	$g(\mathbf{w})$**: Choice 2**
$\mathbf{y}_j = v$	$P_{\mathbf{w}}(\mathbf{y}_j = v)$	
$\neg C_1$	$1 - g_1(\mathbf{w})$	
$C_1 \vee C_2$	$\min(g_1(\mathbf{w}) + g_2(\mathbf{w}), 1)$	$\max(g_1(\mathbf{w}), g_2(\mathbf{w}))$
$C_1 \wedge C_2$	$\max(g_1(\mathbf{w}) + g_2(\mathbf{w}) - 1, 0)$	$\min(g_1(\mathbf{w}), g_2(\mathbf{w}))$

Table 3.1: $g(\mathbf{w})$, $g_1(\mathbf{w})$ and $g_2(\mathbf{w})$ are soft value functions for C, C_1 and C_2, respectively.
.

by applying logical operators over existing constraint(s). Note that $C_1 \implies C_2$ can be written as $\neg C_1 \vee C_2$. Given a logical constraint C over the values output by a network with parameters $\mathbf{w}$, we construct a function $g(\mathbf{w}) \in [0, 1]$ which denotes the soft value of the corresponding logical expression. Table 3.1 describes conversions from a logical expression to a corresponding (soft) value. Finally, given a constraint C and the associated function $g(\mathbf{w})$, the corresponding constraint can be written as: $g(\mathbf{w}) = 1$. Since $g(\mathbf{w}) \in [0, 1]$, it is equivalent to: $g(\mathbf{w}) \geq 1$ or $f(\mathbf{w}) = 1 - g(\mathbf{w}) \leq 0$.

We note that since all our constraints are over variables with probability distributions defined over them, introducing soft logic does not make the constraints any softer, it only gives a way to combine underlying probability values. When making a prediction using the model's output probabilities over values in $\mathcal{V}$, we convert continuous variables (probabilities) into discrete predictions. On the other hand, the process described here can be thought of as reverse of it: the constraint C expressed as a Boolean expression involving the discrete values is being converted into a continuous function of probabilities over the discrete values.

3.4 Training

Supervised: We solve the dual optimization problem described in Equation (3.6) by alternating gradient descent (ascent) steps over $\mathbf{w}$ and Λ, respectively. The gradients of the $\mathcal{L}$ with respect to $\mathbf{w}$ and λ_k are given as:

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \Lambda) = \nabla_{\mathbf{w}} L(\mathbf{w}) + \sum_{k=1}^{K} \lambda_k \nabla_{\mathbf{w}} h_k(\mathbf{w}); \quad \frac{\partial \mathcal{L}(\mathbf{w}; \Lambda)}{\partial \lambda_k} = h_k(\mathbf{w}), \forall k. \tag{3.7}$$

Non-differentiability due to the Hinge function in h_k can be handled by using sub-gradients. Correspondingly, the parameter update equations can be written as:

$$\mathbf{w}^{(t_1+1)} \leftarrow \mathbf{w}^{(t_1)} - \alpha_{\mathbf{w}} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \Lambda); \quad \Lambda^{(t+1)} \leftarrow \Lambda^{(t)} + \alpha_{\Lambda} \nabla_{\Lambda} \mathcal{L}(\mathbf{w}; \Lambda) \tag{3.8}$$

Algorithm 1 presents the pseudocode for our learning algorithm. Initially, $\mathbf{w}$ are updated for a *warmup* number of iterations with each $\lambda_k = 0$ (i.e., no constraints). Then, we

perform the following in succession: for every one update of Λ parameters, we update the $\mathbf{w}$ parameters for l steps, where l grows based on an arithmetic progression in increments of d. Intuitively, this ensures that ratio of the effective learning rates for $\mathbf{w}$ updates and Λ updates goes to infinity with increasing number of Λ updates as $l \to \infty$. For convergence, we resort to the theory of min-max optimization presented by [Jin et al., 2019]. Their key result states that for a min-max optimization problem, alternating gradient ascent (descent) over max (min) variables converges to the local min-max point (analogous of local minima in the single variable case) if the ratio of learning rates of inner and outer variables goes to ∞ in the limit. A significant advantage of our formulation is that in practice the inner loop can often involve application of algorithms such as AdaDelta or RMSProp, which perform gradient descent, but we may not have direct control over the learning rate for $\mathbf{w}$ parameters. But our step based update still ensures that effective ratio of learning rates goes to infinity. We state this formally in our next theorem.

Theorem 1. *Algorithm 1 converges to a Local minmax point of* $\mathcal{L}(\mathbf{w}; \Lambda)$ *for any* $d \geq 1$.

Proof. Without loss of generality, we can assume $warmup = 0$. Then, for a given t (number of Λ updates), let t_1 denote the number of corresponding $\mathbf{w}$ updates. Then, $t_1 = 1 + d + \cdots + t * d$, i.e., $t_1 = O(t^2 d)$. Therefore, the ratio of effective learning rates for $\mathbf{w}$ and Λ updates $= \frac{\alpha_{\mathbf{w}}}{\alpha_{\Lambda_0}}(1 + \beta t)O(td)$. This term goes to ∞ with increasing t. Hence, by Theorem 28 in [Jin et al., 2019], Algorithm 1 converges to the local Minmax point of $\mathcal{L}(\mathbf{w}; \Lambda)$. □

Semi-supervised: Our framework can be easily extended to the case of semi-supervised learning. Since we do not have the target value $\mathbf{y}$ for unlabeled examples, we can't compute the loss (cross-entropy term) in expression for $\mathcal{L}(\mathbf{w}; \Lambda)$ in Equation (3.4) and hence, contribution of unlabeled examples to this term is ignored. On the other hand, the second term in the expression for $\mathcal{L}(\mathbf{w}; \Lambda)$ (corresponding to constraints) does not depend on the target values $\mathbf{y}$. Therefore, for unlabeled examples, we can take this contribution into account by computing this term just like in the case of labeled examples. As demonstrated by our experiments, this simple idea of using unlabeled data only for enforcing the constraints can act as a strong regularizer and result in significantly improved models, especially when there is small amount of labeled data available for training. This is also observed in earlier work ([Xu et al., 2018, Mehta et al., 2018]).

Algorithm 1 Training of a Deep Net with Constraints. Hyperparameters: *warmup*, d, β, α^0_Λ, $\alpha_{\mathbf{w}}$

Initialize: $\mathbf{w}$ randomly; $\lambda_k = 0,\ \forall k = 1 \dots K$
for *warmup iterations* **do**
  Update w: Take an SGD step wrt $\mathbf{w}$ on $\mathcal{L}(\mathbf{w}; \Lambda)$ on a mini-batch
end
Initialize: $l = 1$; $t = 1$; $t_1 = 1$; $\alpha_\Lambda = \alpha^0_\Lambda$
while *not converged* **do**
  Update Λ: Take an SGA step wrt Λ on $\mathcal{L}(\mathbf{w}; \Lambda)$ on a mini-batch
  Increment $t = t + 1$ //counting Λ updates
  for l *steps* **do**
    Update w: Take an SGD step wrt $\mathbf{w}$ on $\mathcal{L}(\mathbf{w}; \Lambda)$ on a mini-batch
    Increment $t_1 = t_1 + 1$ //counting $\mathbf{w}$ updates
  end
  Update $l = l + d$
  Set learning rates: $\alpha_\Lambda = \alpha^0_\Lambda \frac{1}{1+\beta t}$
end

3.5 Experiments

The goal of our experiments is to answer three questions. (1) Does constrained training help in learning more accurate models, especially in the low data setting? (2) Does constrained training result in models with better constraint satisfaction at prediction time? (3) What is the impact of semi-supervision? Please refer to section A.1 for details on hardware and software environment. We perform experiments on three different NLP benchmarks, which we describe next.

3.5.1 Semantic Role Labeling (SRL)

Given a sentence with a predicate (verb), the goal of SRL is to extract and label the arguments for it to determine who did what to whom, when and where, etc. In SRL literature, there is a long history of using linguistic and structural constraints in inference (e.g., [Punyakanok et al., 2008]). We assess the value of constraints in learning more robust neural models.

Dataset & Baseline Model: We use English Ontonotes 5.0 dataset[1] using the CONLL 2011/12 shared task format ([Pradhan et al., 2012]) as the training data. The labeling task is modeled as sequence labeling using the BIOUL encoding. The baseline model (B) uses a deep Bidirectional LSTM, initialized with ElMo+Glove embeddings.[2]

[1] *http://cemantix.org/data/ontonotes.html*
[2] implemented in *https://allennlp.org/models#semantic-role-labeling*

Constraints: We impose two types of constraints. (1) Syntactic Constraints: let $SY = \{(a, b)|a < b\}$ be the set of syntactic spans of a sentence in its syntactic parse tree. Let $y_j^{B_l}$ and $y_j^{L_l}$ be the indicator variables corresponding to beginning and end (last) tag of argument label l at j^{th} word. Then syntactic constraints can be written as $y_a^{B_l} \implies \bigvee_{j \in \{b:(a,b) \in SY\}} y_j^{L_l}$ and $y_b^{L_l} \implies \bigvee_{j \in \{a:(a,b) \in SY\}} y_j^{B_l}$, $\forall\ a, b, l$. These constraints are similar to those used by [Mehta et al., 2018], albeit in a different formulation. (2) Transition Constraints: BIOUL encoding naturally defines valid transitions for a sequence, e.g., L_l must be preceded by B_l or I_l. For a given tag t, let V_t be the set of valid tags for the next word. Then, transition constraints enforce that: $\forall\ j, t : y_j^t \implies \bigvee_{u \in V_t} y_{j+1}^u$.

Methodology: We compare against two different models, the baseline (B), and the baseline augmented with Viterbi decoding (B+CI). This constrained decoding enforces transition constraints at inference time. We name the constrained learning versions of these algorithms by CL and CL+CI, respectively. Note that for test instances, the syntactic spans SY are not available. We use the standard train/dev/test split and use the official Perl script to compute span based F1-scores. We train with 1%, 5% and 10% of training data selected randomly. Refer to section A.1.1 for details on hyperparameters.

Results: Table 3.2 presents our results. We observe significant F1 gains of constrained learning (B+CL) over the baseline B, supporting the hypothesis that constraints can help in learning more robust models. We find that constrained learning with constrained decoding consistently performs the best, even though marginal improvements over B+CI are smaller.[3] We also note that the benefit of constrained learning decreases as training data increases, suggesting that this approach is most useful in low data settings. In addition to F1-scores, we also report total number of constraint violations and find that constrained learning consistently makes significantly fewer violations. At the same time, we note that there are still substantial violations remaining. This is not entirely suprising, since learning span constraints without known spans is akin to learning a significant aspect of the syntactic parsing task, making the learning task much harder.

[3]Our F1-scores are not directly comparable with those reported in [Mehta et al., 2018], since their exact training splits (or code) are unavailable. Overall, our gains due to constrained learning are similar to theirs.

Table 3.2: Effect of constrained learning on SRL, with and without constrained inference (CI).

	F1 Score			Total Constraint Violations		
Scenario	**1% Data**	**5% Data**	**10% Data**	**1% Data**	**5% Data**	**10% Data**
B	62.99	72.64	76.04	14,857	9,708	7,704
CL	66.21	74.27	77.19	9,406	7,461	5,836
B+CI	67.90	75.96	78.63	5,737	4,247	3,654
CL + CI	68.71	76.51	78.72	5,039	3,963	3,476

3.5.2 Named Entity Recognition (NER)

The task corresponds to assigning a tag for each word from a given set of NER tags, *e.g.*, 'location', 'person' etc. In addition, we also assume that the (training) dataset is labeled with Part of Speech (POS) tags for each word. This information is readily available for many datasets. We treat POS tagging as an auxiliary task in the standard multi-task learning (MTL) framework.

Dataset & Baseline Model: We use the publicly available GMB[4] dataset ([Bos et al., 2017]) in our experiments. It contains about 62 thousand sentences, 24 different NER tags and 43 different POS tags. We randomly split it into 60/20/20 train/dev/test sets respectively. After removing the hierarchy among NER tags (*e.g.*, mapping 'person-title' and 'person-family-name' to a single 'person'), we are left with 9 high-level NER tags. We use the BIO encoding in our modeling. Our baseline model (B) is a BiLSTM that is setup in an MTL framework for predicting both NER and POS. For both the tasks, we use a single BiLSTM layer whose parameters are shared between the two tasks.

Constraints: We encode our prior linguistic knowledge about the relationships between NER and POS as constraints – for any NER tag t_e, we have an allowed set of POS tags $T_p(t_e)$. If a word takes an NER tag t_e, then its POS tag must come from the set $T_p(t_e)$, i.e., $y_j^{NER} = t_e \Rightarrow y_j^{POS} \in T_p(t_e)$. Here, y_j^{NER}, y_j^{POS} are the output variables corresponding to NER and POS tags for the j^{th} word, respectively. Table 3.3 enumerates the exact constraints used in this experiment.

Table 3.3: Constraints used in NER task

B-org $\Longrightarrow$ {NNP}	I-per $\Longrightarrow$ {NNP}
B-tim $\Longrightarrow$ {NNP, CD, JJ}	I-org $\Longrightarrow$ {NNP}
B-geo $\Longrightarrow$ {NNP}	I-geo $\Longrightarrow$ {NNP, NNPS}
B-gpe $\Longrightarrow$ {JJ, NNS, NNP}	I-tim $\Longrightarrow$ {CD, NNP, NN, IN}
B-per $\Longrightarrow$ {NNP}	I-eve $\Longrightarrow$ {NNP}
B-eve $\Longrightarrow$ {NNP}	I-art $\Longrightarrow$ {NNP}
B-art $\Longrightarrow$ {NNP, NNPS, JJ, NNS}	I-gpe $\Longrightarrow$ {NNP}
B-nat $\Longrightarrow$ {NNP}	I-nat $\Longrightarrow$ {NNP}

Methodology: We compare the following models. (1) B: Baseline, (2) CI: Constrained Inference, (3) CL: Constrained Learning, and (4) SCL: Semi-supervised Constrained Learning. B is the base model, CI does regular training with constrained inference using

[4] *https://gmb.let.rug.nl/data.php*

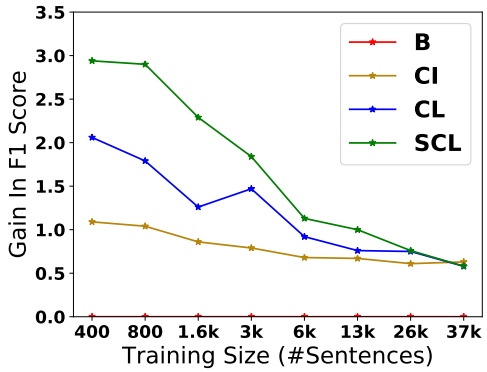
Table 3.4: F–score for different models (mean ± stdev), along with average number of constraint violations

	F1-Score(Mean ± Stdev)				Mean #Violations			
Train Size	**B**	**CL**	**SL**	**CI**	**B**	**CL**	**SL**	**CI**
400	51.6 ± 0.99	53.7 ± 1.16	54.6 ± 0.83	52.7 ± 0.79	4,482	383	7	401
800	57.3 ± 1.45	59.1 ± 1.34	60.2 ± 0.74	58.3 ± 1.25	4,208	201	8	610
1,600	62.3 ± 1.05	63.6 ± 0.51	64.6 ± 0.71	63.2 ± 0.84	3,902	222	4	880
3,200	66.2 ± 0.59	67.7 ± 0.38	68.1 ± 0.5	67 ± 0.55	3,715	141	8	1,147
6,400	69.8 ± 0.54	70.8 ± 0.34	71 ± 0.43	70.5 ± 0.53	3,456	514	64	1,418
12,800	72.1 ± 0.28	72.9 ± 0.36	73.1 ± 0.38	72.8 ± 0.27	3,540	115	147	1,626
25,600	74.3 ± 0.24	75.1 ± 0.17	75.1 ± 0.25	74.9 ± 0.2	3,376	347	315	1,697
37,206	75.3 ± 0.24	75.8 ± 0.21	75.8 ± 0.21	75.9 ± 0.25	3,455	333	333	1,823

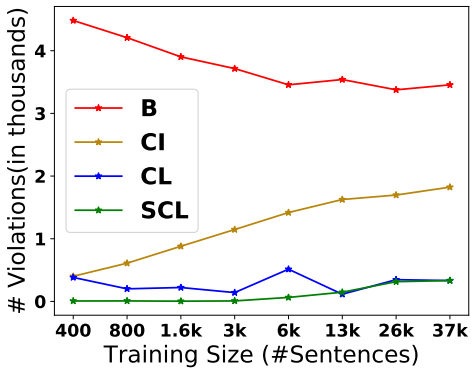
dual decomposition ([Rush and Collins, 2012]), CL is our model doing constrained training (supervised) and SCL does constrained training using semi-supervised data. In order to test the performance of our model in low data setting, we randomly select data subsets of sizes {400, 800, 1600, 6400, 12800, 25600, 37206} and use them for training each model. In each case, the data not used for training is used as unlabeled data for SCL (after removing the labels). The reported results are averaged over 10 different randomly selected samples for each training size. Refer to section A.1.1 for details on hyperparameters.

Results: Table 3.4 enumerates the mean F1-Score over 10 random shuffles of data, along with its stdev, for different models with varying training size. We also tabulate the number of violations in each scenario. Figure 3.1a compares the performance of the four models. We plot the baseline model at zero, and plot the performance of all other models relative to the baseline (see supplement for absolute numbers and standard deviations). There is a good gain in F1-score when learning with constraints, with most gain obtained for smaller training sizes. Semi-supervision results in significant additional gains. Figure 3.1b plots the number of constraint violations with varying training size. For CL and SCL, this number is close to 0 all through. Counter intuitively, the violations increase monotonically for CI. This is because with less training data, learning is very shallow, resulting in 'Other' prediction most of the time, and the constraints are trivially satisfied. As learned model becomes more complex, CI finds it harder to satisfy the constraints without hurting the performance. We do early stopping of dual decomposition based on dev set performance. This results in decent F1 but high constraint violations. If run till convergence, CI results in all constraints being satisfied but with performance lower than the baseline. CL does not suffer from this phenomenon due to constraint aware learning.

We also experiment with using CI (constrained inference) on top of CL and SCL; this



(a) Avg. Gain in F1 Score Over Baseline.



(b) Avg. number of Constrained Violations

Figure 3.1: NER: Comparison of different training techniques. B: Baseline; CL: Constrained Learning; SCL: Semi-supervised Constrained Learning; CI: Constrained Inference

results in no additional gains, since most constraints are already being satisfied. This highlights that constrained learning may sometimes obviate the need for constrained inference, This can lead to a huge reduction in precious test times. For instance CI has test times 3-15 times that of CL, depending on testing batch size.

3.5.3 Fine Grained Entity Typing

This is a multi-label classification problem. Given a set M of textual mentions of an entity e, we are interested in finding all the types that the mentions in M belong to ([Yao et al., 2013, Verga et al., 2017, Murty et al., 2018]). Note that the labels here are entity types.

Dataset & Baseline Model: We work with Typenet[5] ([Murty et al., 2017]), a publicly available dataset of hierarchical entity types for extremely fine-grained entity typing. It has been curated by mapping Freebase ([Bollacker et al., 2008]) types into Wordnet ([Miller, 1995]) hierarchy. The dataset contains over 1,900 types, placed in a hierarchy of average depth of 7.8. It also provides a corpus of textual mentions extracted from Wikipedia articles. It contains 344,246 entities mapped to 1,081 types arranged in the type hierarchy. For baseline (B), we use the state of the art model proposed for this task by [Murty et al., 2018].[6] Each mention m is represented by an encoding computed using a CNN over the sentence, and each type is represented using an embedding vector. The two are combined to get a similarity score. Scores coming from different mentions in a set are pooled to get a final score for each entity. To exploit the hierarchical structure, an additional loss term (H) encourages entities close in the hierarchy to get similar embedding

[5] *https://github.com/iesl/TypeNet*
[6] https://github.com/MurtyShikhar/Hierarchical-Typing

vectors. This can be thought of as imposing a soft constraint on the entity types. We compare with both these versions in our experiments.

Constraints: We enforce two types of constraints in our model. (a) Type Inclusion: given two types t_i and t_j such that t_i is ancestor of t_j in the type hierarchy, for any entity, if t_i is selected as a possible entity type, then t_j should also be selected. *I.e.*, $y_{t_i} \Rightarrow y_{t_j}$ where y_{t_i} and y_{t_j} are indicators variables for the corresponding types being selected. This results in 1,891 constraints. (b) Type Exclusion: pairs of types t_i and t_j (e.g., 'library' and 'camera') that should not co-occur for any entity. *I.e.*, $y_{t_i} \Rightarrow \neg y_{t_j}$. This results in a total of about 555,000 constraints.

Methodology: We compare four different models: (a) B: Baseline, (b) B+H: Baseline with hierarchically constrained embeddings (c) CL: constrained learning (d) SCL: constrained learning with semi-supervision. Our constrained learning models are learned on top of vanilla baseline (and do not make use of hierarchical embeddings). We use the original splits of 90%, 5% and 5% for training, validation and testing, respectively ([Murty et al., 2018]). We compare the performance of the four models for training at (1) 5% of the data (2) 10% of the data, and (3) full training set. The smaller training subsets are chosen randomly. As earlier, any unused data in the training fold is used for semi-supervision (after removing labels). Refer to section A.1.1 for details on hyperparameters.

Results: Table 3.5 presents our comparison results. We note that our baseline results are significantly higher than those reported in [Murty et al., 2018]. We believe this is because they did not train the model until convergence; running till convergence results in significantly higher numbers. After additional training, the relative advantage of B+H model over B as reported in their paper is lost. Our constrained model (CL) can give up to 11 pt increase in the performance both at 5% of the data as well as 10% of the data. With semi-supervision, this gain hovers in the range of 12-14 pts. There is three orders of magnitude drop in the number of constraint violations when using constrained learning. Interestingly, CL performance with 100% data is slightly worse than semi-supervision with 10% data. We hypothesize that the reason is noise in training data in terms of either missing or incorrect labels. We note that there are $634,544$ type inclusion constraint violations in the training data containing $294,781$ entities. As a result, more noisy data is likely hurting the performance.

We also compare our constrained learning against [Diligenti et al., 2017a]'s approach of using soft constraints, where the violation penalty is multiplied with a constant λ and added to the original loss. When using the best values of the λ parameter, we find that both methods perform similarly. However, Diligenti's performance requires extensive search over the λ parameters, and varies significantly based on the value of λ. On the other hand, our formulation can implicitly discover optimal λ_k values (we have one for

every constraint) by way of our Langrangian formulation. This obviates the need for an explicit search for λ which can be expensive. In fact, setting constant λ to be the average of λ_k's from our formulation gives its best score.

Table 3.5: TypeNet: MAP Scores (in %) and # of constraint violations for different training sizes.

	MAP Scores			Constraint Violations		
Scenario	**5% Data**	**10% Data**	**100% Data**	**5% Data**	**10% Data**	**100% Data**
B	68.62	69.21	70.47	22,715	21,451	22,359
B+H	68.71	69.31	71.77	22,928	21,157	24,650
CL	80.13	81.36	82.80	25	45	12
SCL	82.22	83.81		41	26	

3.6 Conclusion

In this chapter, we have proposed a primal-dual based approach for solving the problem of learning with hard constraints in deep learning models. While earlier work has modeled the constraints as soft, incorporating penalty in the loss term, we instead directly optimize the hard constraints using a Langrangian based formulation. We show that our algorithm converges to local min-max points of the objective. For the case of discrete output spaces, we also present a constraint language using soft logic. Experiments on three different NLP tasks show the effectiveness of our approach compared to non-constrained baselines, as well as constrained inference, achieving the state-of-the-art results in two of the domains. In one of the domains, our approach completely eliminates the need for expensive constrained inference. All our code is publicly available at: *https://github.com/dair-iitd/dl-with-constraints*.

We note that it is not always possible for humans to encode their domain knowledge in the form of explicit constraints. To overcome this, in the following chapters, we will shift our attention towards learning constraints automatically. In particular, we focus on symbolic problems where the constraints form the underlying core for solving them.

Part III

Unknown Constraints with Implicit Representation

Chapter 4

1oML: One of Many Learning

In the last chapter, we exploited domain knowledge expressed as Boolean expressions over output labels to regularize and train better models. However, it may not be always feasible for domain experts to express their knowledge succinctly as a Boolean expression. Thus, we may have to discover the constraints automatically and that is the focus of rest of the work. To better understand the task of learning constraints, we focus on problems where constraints form the core of a reasoning model, *e.g.*, reasoners for combinatorial problems. This is in contrast to chapter 3 where we used constraints over outputs to aid the learning of a neural model.

Many works have proposed neural architectures (reasoners) for solving combinatorial problems in structured output spaces. In many such problems, there may exist multiple solutions for a given input, *e.g.*, a partially filled Sudoku puzzle may have many completions satisfying all constraints. Further, we are often interested in finding *any one* of the possible solutions, without any preference between them. Existing approaches completely ignore this solution multiplicity. In this chapter, we argue that being oblivious to the presence of multiple solutions can severely hamper their training ability. Our contribution is two fold. First, we formally define the task of learning one-of-many solutions for combinatorial problems in structured output spaces, which is applicable for solving several problems of interest such as N-Queens, and Sudoku. Second, we present a generic learning framework that adapts an existing prediction network for a combinatorial problem to handle solution multiplicity. Our framework uses a selection module, whose goal is to dynamically determine, for every input, the solution that is most effective for training the network parameters in any given learning iteration. We propose an RL based approach to jointly train the selection module with the prediction network. Experiments on three different domains, and using two different prediction networks, demonstrate that our framework significantly improves the accuracy in our setting, obtaining up to 21 pt gain over the

baselines.

4.1 Introduction

Recent research has explored application of neural networks for solving symbolic reasoning tasks, requiring higher level inferences, such as solving SAT [Selsam et al., 2019, Amizadeh et al., 2019a,b], neural theorem proving [Rocktäschel et al., 2015, Minervini et al., 2020], differentiable ILP (∂ILP) [Evans and Grefenstette, 2018], playing blocks world [Dong et al., 2019], solving sudoku [Wang et al., 2019]. The capability of neural models to perform symbolic reasoning is often seen as a step towards the framework for unified AI, *i.e.*, building end-to-end trainable system for tasks, which need to combine low level perception with high level cognitive reasoning [Kahneman, 2011]. Our paper focuses on one such high level reasoning task – solving combinatorial problems in structured output spaces, *e.g.*, solving a Sudoku or N-Queens puzzle. These can be thought of as Constraint Satisfaction problems (CSPs) where the underlying constraints are not explicitly available, and need to be learned from training data. We focus on learning such constraints by a non-autoregressive neural model where variables in the structured output space are decoded simultaneously (and therefore independently, given the input). Notably, most of the current state-of-the-art neural models for solving combinatorial problems, *e.g.*, SATNet [Wang et al., 2019], RRN [Palm et al., 2018], NLM [Dong et al., 2019], work with non autoregressive architectures because of their high efficiency of training and inference, since they do not have to decode the solution sequentially.

One of the key characteristics of such problems is *solution multiplicity* – there could be many correct solutions for any given input, even though we may be interested in finding any one of these solutions. For example, in a game of Sudoku with only 16 digits filled, there are always multiple correct solutions [McGuire et al., 2012], and obtaining any one of them suffices for solving Sudoku. Unfortunately, existing literature has completely ignored solution multiplicity, resulting in sub-optimally trained networks. Our preliminary analysis of a state-of-the-art neural Sudoku solver [Palm et al., 2018][1], which trains and tests on instances with single solutions, showed that it achieves a high accuracy of 96% on instances with single solution, but the accuracy drops to less than 25%, when tested on inputs that have multiple solutions. Intuitively, the challenge comes from the fact that (a) there could be a very large number of possible solutions for a given input, and (b) the solutions may be highly varied. For example, a 16-givens Sudoku puzzle could have as many as 10,000 solutions, with maximum hamming distance between any two solutions

[1] Available at https://data.dgl.ai/models/rrn-sudoku.pkl

being 61. Hence, we argue that an explicit modeling effort is required to represent this solution multiplicity.

As the first contribution of our work, we formally define the novel problem of *One-of-Many Learning* (1oML). It is given training data of the form $\{(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}})\}$, where $\mathbf{Y}_{\mathbf{x}^{(i)}}$ denotes a subset of all correct outputs $\mathcal{Y}_{\mathbf{x}^{(i)}}$ associated with input $\mathbf{x}^{(i)}$. The goal of 1oML is to learn a function f such that, for any input $\mathbf{x}$, $f(\mathbf{x}) = \mathbf{y}$ for some $\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}$. We show that a naïve strategy that uses separate loss terms for each $(\mathbf{x}^{(i)}, \mathbf{y}_j^{(i)})$ pair where $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ can result in a bad likelihood objective. Next, we introduce a multiplicity aware loss (CC-Loss) and demonstrate its limitations for non-autoregressive models on structured output spaces. In response, we present our first-cut approach, MinLoss, which picks up the single $\mathbf{y}_j^{(i)}$ closest to the prediction $\hat{\mathbf{y}}^{(i)}$ based on the current parameters of prediction network (base architecture for function f), and uses it to compute and back-propagate the loss for that training sample $\mathbf{x}^{(i)}$. Though significantly better than naïve training, through a simple example, we demonstrate that MinLoss can be sub-optimal in certain scenarios, due to its inability to pick a $\mathbf{y}_j^{(i)}$ based on global characteristics of solution space.

To alleviate the issues with MinLoss, we present two exploration based techniques, I-ExplR and SelectR, that select a $\mathbf{y}_j^{(i)}$ in a non-greedy fashion, unlike MinLoss. Both techniques are generic in the sense that they can work with any prediction network for the given problem. I-ExplR relies on the prediction network itself for selecting $\mathbf{y}_j^{(i)}$, whereas SelectR is an RL based learning framework which uses a selection module to decide which $\mathbf{y}_j^{(i)}$ should be picked for a given input $\mathbf{x}^{(i)}$, for back-propagating the loss in the next iteration. The SelectR's selection module is trained jointly along with the prediction network using reinforcement learning, thus allowing us to trade-off exploration and exploitation in selecting the optimum $\mathbf{y}_j^{(i)}$ by learning a probability distribution over the space of possible $\mathbf{y}_j^{(i)}$'s for any given input $\mathbf{x}^{(i)}$.

We experiment on three CSPs: N-Queens, Futoshiki, and Sudoku. Our prediction networks for the first two problems are constructed using Neural Logic Machines [Dong et al., 2019], and for Sudoku, we use a state-of-the-art neural solver based on Recurrent Relational Networks [Palm et al., 2018]. In all three problems, our experiments demonstrate that SelectR vastly outperforms naïve baselines by up to 21 pts, underscoring the value of explicitly modeling solution multiplicity. SelectR also consistently improves on other multiplicity-aware methods, viz. CC-Loss, MinLoss, and I-ExplR.

This work primarily focuses on defining an appropriate loss function and training techniques for non-autoregressive models. However, to address thesis reviewers' concerns and for completeness, we also experiment with an autoregressive encoder-decoder model and demonstrate the importance of multiplicity-aware loss functions for their training. We first observe that autoregressive models have better representation power than their

non-autoregressive counterparts, giving them the ability to represent a probability simplex over multiple solutions for a given input. However, it comes at a much higher training and inference cost when compared with non-autoregressive models. Owing to their better representation power, autoregressive models perform better than their non-autoregressive counterparts when trained with the naïve strategy under solution multiplicity. However, all our techniques to handle multiplicity result in significant improvement, even for autoregressive models. Finally, we show that a non-autoregressive model trained using SelectR, our best performing exploration-based technique, surpasses the performance of the auto-regressive model, and that too with much lower training and inference cost.

4.2 Background and Related Work

Related ML Models: There are a few learning scenarios within weak supervision which may appear similar to the setting of 1oML, but are actually different from it. We first discuss them briefly. '*Partial Label Learning*' (PLL) [Jin and Ghahramani, 2002, Cour et al., 2011, Xu et al., 2019, Feng and An, 2019, Cabannes et al., 2020] involves learning from the training data where, for each input, a noisy set of candidate labels is given amongst which only one label is correct. This is different from 1oML in which there is no training noise and all the solutions in the solution set $\mathbf{Y_x}$ for a given $\mathbf{x}$ are correct. Though some of the recent approaches to tackle ambiguity in PLL [Cabannes et al., 2020] may be similar to our methods, i.e., MinLoss , by the way of deciding which solution in the target set should be picked next for training, the motivations are quite different. Similarly, in the older work by [Jin and Ghahramani, 2002], the *EM model*, where the loss for each candidate is weighted by the probability assigned to that candidate by the model itself, can be seen as a naïve exploration based approach, applied to a very different setting. In PLL, the objective is to select the correct label out of many incorrect ones to reduce training noise, whereas in 1oML, selecting only one label for training provably improves the learnability and there is no question of reducing noise as all the labels are correct. Further, most of the previous work on PLL considers classification over a discrete output space with, say, L labels, where as in 1oML, we work with structured output spaces, e.g., an r dimensional vector space where each dimension represents a discrete space of L labels. This exponentially increases the size of the output space, making it intractable to enumerate all possible solutions as is typically done in existing approaches for PLL [Jin and Ghahramani, 2002].

Within weak supervision, the work on '*Multi Instance Learning*' (MIL) approach for Relation Extraction (RE) employs a selection module to pick a set of sentences to be used

for training a relation classifier, given a set of noisy relation labels [Feng et al., 2018, Qin et al., 2018]. This is different from us where multiplicity is associated with any given input, not with a class (relation).

Other than weak supervision, 1oML should also not be confused with the problems in the space of multi-label learning [Tsoumakas and Katakis, 2007]. In multi-label learning, given a solution set $\mathbf{Y_x}$ for each input $\mathbf{x}$, the goal is to correctly predict each possible solution in the set $\mathbf{Y_x}$ for $\mathbf{x}$. Typically, a classifier is learned for each of the possible labels separately. On the other hand, in 1oML, the objective is to learn any one of the correct solutions for a given input, and a single classifier is learned. The characteristics of the two problems are quite different, and hence, also the solution approaches. As we show later, the two settings lead to requirements for different kinds of generalization losses.

Solution Multiplicity in Other Settings: There is some prior work related to our problem of solution multiplicity, albeit in different settings. An example is the task of video-prediction, where there can be multiple next frames ($\mathbf{y}_j^{(i)}$) for a given partial video $\mathbf{x}^{(i)}$ [Henaff et al., 2017, Denton and Fergus, 2018]. The multiplicity of solutions here arises from the underlying uncertainty rather than as a inherent characteristic of the domain itself. Current approaches model the final prediction as a combination of the deterministic part oblivious to uncertainty, and a non-determinstic part caused by uncertainty. There is no such separation in our case since each solution is inherently different from others.

Another line of work, which comes close to ours is the task of Neural Program Synthesis [Devlin et al., 2017, Bunel et al., 2018]. Given a set of Input-Output (IO) pairs, the goal is to generate a valid program conforming to the IO specifications. For a given IO pair, there could be multiple valid programs, and often, training data may only have one (or a few) of them. Bunel et al. [2018] propose a solution where they define an alternate RL based loss using the correctness of the generated program on a subset of held out IO pairs as reward. In our setting, in the absence of the constraints (or rules) of the CSP, there is no such additional signal available for training outside the subset of targets $\mathbf{Y_x}$ for an input $\mathbf{x}$.

It would also be worthwhile to mention other tasks such as Neural Machine translation [Bahdanau et al., 2015, Sutskever et al., 2014], Summarization [Nallapati et al., 2017, Paulus et al., 2018], Image Captioning [Vinyals et al., 2017, You et al., 2016] *etc.*, where one would expect to have multiple valid solutions for any given input. *E.g.*, for a given sentence in language A, there could be multiple valid translations in language B. To the best of our knowledge, existing literature ignores solution multiplicity in such problems, and simply trains on all possible given labels for any given input.

Models for Symbolic Reasoning: Our work follows the line of recent research, which proposes neural architectures for implicit symbolic and relational reasoning problems

[Santoro et al., 2018, Palm et al., 2018, Wang et al., 2019, Dong et al., 2019]. We experiment with two architectures as base prediction networks: Neural Logic Machines (NLMs) [Dong et al., 2019], and Recurrent Relational Networks (RRNs) [Palm et al., 2018]. NLMs allow learning of first-order logic rules expressed as Horn Clauses over a set of predicates, making them amenable to transfer over different domain sizes. The rules are instantiated over a given set of objects, where the groundings are represented as tensors in the neural space over which logical rules operate. RRNs use a graph neural network to learn relationships between symbols represented as nodes in the graph, and have been shown to be good at problems that require multiple steps of symbolic reasoning.

4.3 Theory and Algorithm

4.3.1 Problem Definition

Notation: Each possible solution (target) for an input (query) $\mathbf{x}$ is denoted by an r-dimensional vector $\mathbf{y} \in \mathcal{V}^r$, where each element of $\mathbf{y}$ takes values from a discrete space denoted by $\mathcal{V}$. Let $\mathcal{Y} = \mathcal{V}^r$, and let $\mathcal{Y}_\mathbf{x}$ denote the set of all solutions associated with input $\mathbf{x}$. We will use the term *solution multiplicity* to refer to the fact that there could be multiple possible solutions $\mathbf{y}$ for a given input $\mathbf{x}$. In our setting, the solutions in $\mathcal{Y}_\mathbf{x}$ span a structured combinatorial subspace of $\mathcal{V}^r$, and can be thought of as representing solutions to an underlying Constraint Satisfaction Problem (CSP). For example in N-Queens, $\mathbf{x}$ would denote a partially filled board, and $\mathbf{y}$ denote a solution for the input board.

Given a set of inputs $\mathbf{x}^{(i)}$ along with a subset of associated solutions $\mathbf{Y}_{\mathbf{x}^{(i)}} \subseteq \mathcal{Y}_{\mathbf{x}^{(i)}}$, *i.e.*, given a set of $(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}})$ pairs, we are interested in learning a mapping from $\mathbf{x}$ to any one $\mathbf{y}$ among many possible solutions for $\mathbf{x}$. Formally, we define the *One-of-Many-Learning (1oML)* problem as follows.

Definition 1. *Given training data $\mathcal{D}$ of the form, $\{(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}})\}_{i=1}^m$, where $\mathbf{Y}_{\mathbf{x}^{(i)}}$ denotes a subset of solutions associated with input $\mathbf{x}^{(i)}$, and m is the size of training dataset,* One-of-Many-Learning (1oML) *is defined as the problem of learning a function f such that, for any input $\mathbf{x}$, $f(\mathbf{x}) = \mathbf{y}$ for some $\mathbf{y} \in \mathcal{Y}_\mathbf{x}$, where $\mathcal{Y}_\mathbf{x}$ is the set of all solutions associated with $\mathbf{x}$.*

We use parameterized neural networks to represent our mapping function. We use M_Θ to denote a non-autoregressive network M with associated set of parameters Θ. We use $\hat{\mathbf{y}}^{(i)}$ ($\hat{\mathbf{y}}$) to denote the network output corresponding to input $\mathbf{x}^{(i)}$ ($\mathbf{x}$), *i.e.*, $\hat{\mathbf{y}}^{(i)}$ ($\hat{\mathbf{y}}$) is the arg max of the learnt conditional distribution over the output space $\mathcal{Y}$ given the input $\mathbf{x}^{(i)}$

($\mathbf{x}$). We are interested in finding a Θ^* that solves the 1oML problem as defined above. Next, we consider various formulations for the same.

4.3.2 Objective Function

Naïve Objective: In the absence of solution multiplicity, *i.e.* when target set $\mathbf{Y}_{\mathbf{x}^{(i)}} = \{\mathbf{y}^{(i)}\}$, $\forall i$, the standard method to train such models is to minimize the total loss, $L(\Theta) = \sum_{i=1}^{m} l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$, where $l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$ is the loss between the prediction $\hat{\mathbf{y}}^{(i)}$ and the unique target $\mathbf{y}^{(i)}$ for the input $\mathbf{x}^{(i)}$. We find the optimal Θ^* as $\text{argmin}_\Theta\, L(\Theta)$. A Naïve extension of this for 1oML would be to sum the loss over all targets in $\mathbf{Y}_\mathbf{x}$, *i.e.*, minimize the following loss function:

$$L(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)}) \tag{4.1}$$

We observe that loss function in eq. (4.1) would unnecessarily penalize the model when dealing with solution multiplicity. Even when it is correctly predicting one of the targets for an input $\mathbf{x}^{(i)}$, the loss with respect to the other targets in $\mathbf{Y}_{\mathbf{x}^{(i)}}$ could be rather high, hence misguiding the training process. Example 1 below demonstrates such a case. For illustration, we will use the cross-entropy loss, *i.e.*, $l_\Theta(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_k \sum_l \mathbb{1}\{\mathbf{y}[k] = v_l\} \log(P(\hat{\mathbf{y}}[k] = v_l))$, where $v_l \in \mathcal{V}$ varies over the elements of $\mathcal{V}$, and k indices over r dimensions in the solution space. $\mathbf{y}[k]$ denotes the k^{th} element of $\mathbf{y}$.

Example 1. *Consider a learning problem over a discrete (Boolean) input space $\mathcal{X} = \{0, 1\}$ and Boolean target space in two dimensions, i.e., $\mathcal{Y} = \mathcal{V}^r = \{0, 1\}^2$. Let this be a trivial learning problem where $\forall \mathbf{x}$, the solution set is $\mathbf{Y}_\mathbf{x} = \{(0, 1), (1, 0)\}$. Then, given a set of examples $\{\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}}\}$, the Naïve objective (with l_Θ as cross entropy) will be minimized, when $P(\hat{\mathbf{y}}^{(i)}[k] = 0) = P(\hat{\mathbf{y}}^{(i)}[k] = 1) = 0.5$, for $k \in \{1, 2\}$, $\forall i$, which can not recover either of the desired solutions: $(0, 1)$ or $(1, 0)$.*

The problem arises from the fact that when dealing with 1oML, the training loss defined in eq. (4.1) is no longer a consistent predictor of the generalization error as formalized below.

Lemma 2. *The training loss $L(\Theta)$ as defined in eq. (4.1) is an inconsistent estimator of generalization error for 1oML, when l_Θ is a zero-one loss, i.e., $l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)}) = \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\}$.*

Proof. Let $\mathcal{D}$ represent the distribution using which samples $(\mathbf{x}, \mathcal{Y}_\mathbf{x})$ are generated. In our setting, generalization error $\varepsilon(M_\Theta)$ for a prediction network M_Θ can be written as: $\varepsilon(M_\Theta) = \mathbb{E}_{(\mathbf{x}, \mathcal{Y}_\mathbf{x}) \sim \mathcal{D}}(\mathbb{1}\{\hat{y} \notin \mathcal{Y}_\mathbf{x}\})$, where $\hat{\mathbf{y}} = M_\Theta(\mathbf{x})$, *i.e.* the prediction of the network on unseen example sampled from the underlying data distribution. Assume a scenario when

$\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}$, $\forall i$, *i.e.*, for each input $\mathbf{x}^{(i)}$ all the corresponding solutions are present in the training data. Then, an unbiased estimator $\hat{\varepsilon}_{\mathcal{D}}(\mathbf{M}_\Theta)$ of the generalization error, computed using the training data is written as: $\hat{\varepsilon}_{\mathcal{D}}(M_\Theta) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \notin \mathbf{Y}_{\mathbf{x}^{(i)}}\}$. Clearly, the estimator obtained using $L(\Theta)$ (Naïve Objective), when the loss function $l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)})$ is replaced by a zero-one loss $\mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\}$, is not a consistent estimator for the generalization error. This can be easily seen by considering a case when $\hat{\mathbf{y}}^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ and $|\mathbf{Y}_{\mathbf{x}^{(i)}}| > 1$. □

For the task of PLL, Jin and Ghahramani [2002] propose a modification of the cross entropy loss to tackle multiplicity of labels in the training data. Instead of adding the log probabilities, it maximizes the log of total probability over the given target set. Inspired by Feng et al. [2020], we call it CC-Loss:

$$L_{cc}(\Theta) = -\frac{1}{m}\sum_{i=1}^{m} \log\left(\sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} Pr\left(\mathbf{y}_j^{(i)} | \mathbf{x}^{(i)}; \Theta\right)\right) \tag{4.2}$$

However, for non-autoregressive models, CC-Loss also suffers from the same issues illustrated in example 1 for naïve objective. Moreover, in the case of structured prediction, optimizing L_{cc} requires careful implementation due to its numerical instability. We illustrate this with an example:

Example 2. *Consider solving a* 9×9 *sudoku puzzle,* $\mathbf{x}^{(i)}$*. The probabilty of a particular target board,* $\mathbf{y}_j^{(i)}$*, is a product of* $r = 9^2 = 81$ *individual probabilities over the discrete space* $\mathcal{V} = \{1 \cdots 9\}$ *of size* 9*, i.e.,*

$$Pr(\mathbf{y}_j^{(i)} | \mathbf{x}^{(i)}; \Theta) = \prod_{k=1}^{r} Pr(\mathbf{y}_j^{(i)}[k] | \mathbf{x}^{(i)}; \Theta)$$

In the beginning of the training process, the network outputs nearly uniform probability over $\mathcal{V}$ *for each of the* r *dimensions, making* $Pr(\mathbf{y}_j^{(i)} | \mathbf{x}^{(i)}; \Theta)$ *very small (*$= 9^{-81} \sim 5.09\text{e}{-78}$*). The derivative of* log *of such a small quantity becomes numerically unstable.*

This issue is circumvented in the case of naïve loss by directly working with log *probabilities and log-sum-exp trick* [2]*. However, in the case of CC-Loss, we need to sum the probabilities over the target set* $\mathbf{Y}_{\mathbf{x}^{(i)}}$ *before taking* log*, and computing* $Pr(\mathbf{y}_j^{(i)} | \mathbf{x}^{(i)}; \Theta)$ *makes it numerically unstable. Motivated by log-sum-exp trick, we use the following modifications which involves computing only* log *probabilities. For simplicity of notation, we will use* $\Pr(\mathbf{y}_j^{(i)})$ *to denote* $\Pr(\mathbf{y}_j^{(i)} | \mathbf{x}^{(i)}; \Theta)$ *and* $L_{cc}^{(i)}$ *to denote the CC Loss for the* i^{th} *training sample.*

$$L_{cc}^{(i)} = -\log\left(\sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} Pr\left(\mathbf{y}_j^{(i)}\right)\right)$$

[2] https://blog.feedly.com/tricks-of-the-trade-logsumexp/

Multiply and divide by $maxp^{(i)} = \max_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} Pr(\mathbf{y}_j^{(i)})$*:*

$$L_{cc}^{(i)} = -\log\left(maxp^{(i)} \sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \frac{Pr(\mathbf{y}_j^{(i)})}{maxp^{(i)}}\right)$$

Using the identity $\alpha = \exp(\log(\alpha))$*, we get:*

$$L_{cc}^{(i)} = -\log(maxp^{(i)}) - \log\left(\sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \exp\left(\log\left(\frac{Pr(\mathbf{y}_j^{(i)})}{maxp^{(i)}}\right)\right)\right)$$

$$= -\log(maxp^{(i)}) - \log\left(\sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \exp\left(\log\left(Pr(\mathbf{y}_j^{(i)})\right) - \log\left(maxp^{(i)}\right)\right)\right)$$

In the above equations, we first separate out the max probability target (similar to log-sum-exp trick), and then exploit the observation that the ratio of (small) probabilities is more numerically stable than the individual (small) probabilities. Further, we compute this ratio using the difference of individual log probabilities.

New Objective: We now motivate a better objective function based on an unbiased estimator. In general, we would like $\mathbf{M}_\Theta$ to learn a conditional probability distribution $Pr(\mathbf{y}|\mathbf{x}^{(i)};\Theta)$ over the output space $\mathcal{Y}$ such that the entire probability mass is concentrated on the desired solution set $\mathbf{Y}_{\mathbf{x}^{(i)}}$, *i.e.*, $\sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} Pr(\mathbf{y}_j^{(i)}|\mathbf{x}^{(i)};\Theta) = 1, \forall i$. If such a conditional distribution is learnt, then we can easily sample a $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ from it. CC-Loss is indeed trying to achieve this. However, owing to the conditional independence of different dimensions in a non-autoregressive model, it is impossible to represent all possible joint distributions over the possible solutions in $\mathbf{Y}_{\mathbf{x}^{(i)}}$. For example, as seen in example 1, a non-autoregressive model fails to represent a uniform distribution over $\{(0,1),(1,0)\}$. While doing so, it ends up assigning a probability of 0.5 to both the values 0 and 1, independently for the two dimensions, resulting in a uniform distribution over the entire output space $\mathcal{Y} = \{(0,0),(0,1),(1,0),(1,1)\}$[3].

Since we can't represent all possible distributions concentrated on the desired solution set $\mathbf{Y}_{\mathbf{x}^{(i)}}$, we design a loss function that forces the model to learn a distribution in which the probability mass is concentrated on *any one* of the targets $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$. We call such distributions as one-hot. To do this, we introduce $|\mathbf{Y}_{\mathbf{x}^{(i)}}|$ number of new learnable Boolean parameters, $\mathbf{w}^{(i)}$, for each query $\mathbf{x}^{(i)}$ in the training data, and correspondingly define the following loss function:

$$L_{\mathbf{w}}(\Theta, \mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \mathbf{w}_j^{(i)} l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)}) \tag{4.3}$$

[3]We empirically study the impact of solution-multiplicity on auto-regressive models in section 4.5

Here, $\mathbf{w}_j^{(i)} \in \{0,1\}$ and $\sum_j \mathbf{w}_j^{(i)} = 1, \forall i$, where j indices over solutions $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$. The last constraint over Boolean variables $\mathbf{w}_j^{(i)}$ enforces that exactly one of the weights in $\mathbf{w}^{(i)}$ is 1 and all others are zero.

Lemma 3. *Under the assumption* $\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}, \forall i$, *the loss* $L'(\Theta) = \min_{\mathbf{w}} L_{\mathbf{w}}(\Theta, \mathbf{w})$, *defined as the minimum value of* $L_{\mathbf{w}}(\Theta, \mathbf{w})$ *(defined in eq. (4.3)) with respect to* $\mathbf{w}$, *is a consistent estimator of generalization error for 1oML, when* l_Θ *is a zero-one loss, i.e.,* $l_\Theta(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)}) = \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\}$.

Proof. Let $\mathcal{D}$ represent the distribution using which samples $(\mathbf{x}, \mathcal{Y}_\mathbf{x})$ are generated. In our setting, generalization error $\varepsilon(M_\Theta)$ for a prediction network M_Θ is:

$$\varepsilon(M_\Theta) = \mathbb{E}_{(\mathbf{x}, \mathcal{Y}_\mathbf{x}) \sim \mathcal{D}}(\mathbb{1}\{\hat{y} \notin \mathcal{Y}_\mathbf{x}\})$$

where $\hat{\mathbf{y}} = M_\Theta(\mathbf{x})$, *i.e.* the prediction of the network on unseen example sampled from the underlying data distribution. Assume a scenario when $\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}$, $\forall i$, *i.e.*, for each input $\mathbf{x}^{(i)}$ all the corresponding solutions are present in the training data. Then, an unbiased estimator $\hat{\varepsilon}_\mathcal{D}(\mathbf{M}_\Theta)$ of the generalization error, computed using the training data is written as:

$$\hat{\varepsilon}_\mathcal{D}(M_\Theta) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \notin \mathbf{Y}_{\mathbf{x}^{(i)}}\}$$

Now, consider the objective function $L'(\Theta)$ as:

$$L'(\Theta) = \min_{\mathbf{w}} L_{\mathbf{w}}(\Theta, \mathbf{w}) = \min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} \sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \mathbf{w}_j^{(i)} \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \min_{\mathbf{w}^{(i)}} \sum_{\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} \mathbf{w}_j^{(i)} \mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\}$$

$$\text{s.t. } \mathbf{w}_j^{(i)} \in \{0,1\}\ \forall i, \forall j \text{ and } \sum_{j=1}^{|\mathbf{Y}_{\mathbf{x}^{(i)}}|} \mathbf{w}_j^{(i)} = 1, \forall i = 1 \ldots m$$

For any $\mathbf{x}^{(i)}$, if the prediction $\hat{\mathbf{y}}^{(i)}$ is correct, *i.e.*, $\exists \mathbf{y}_{j*}^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ *s.t.* $\hat{\mathbf{y}}^{(i)} = \mathbf{y}_{j*}^{(i)}$, then $\mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_{j*}^{(i)}\} = 0$ and $\mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\} = 1, \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}, \mathbf{y}_j^{(i)} \neq \mathbf{y}_{j*}^{(i)}$. Now minimizing over $\mathbf{w}^{(i)}$ ensures $\mathbf{w}_{j*}^{(i)} = 1$ and $\mathbf{w}_j^{(i)} = 0\ \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}, \mathbf{y}_j^{(i)} \neq \mathbf{y}_{j*}^{(i)}$. Thus, the contribution to the overall loss from this example $\mathbf{x}^{(i)}$ is zero.

On the other hand if the prediction is incorrect then $\mathbb{1}\{\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}_j^{(i)}\} = 1,\ \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$, thus making the loss from this example to be 1 irrespective of the choice of $\mathbf{w}^{(i)}$. As a result, $L'(\Theta)$ is exactly equal to $\hat{\varepsilon}_\mathcal{D}(M_\Theta)$ and hence it is a consistent estimator for generalization error. □

Next, we define our new objective as:

$$\min_{\Theta,\mathbf{w}} L_{\mathbf{w}}(\Theta,\mathbf{w}) \;\text{ s.t. } \mathbf{w}_j^{(i)} \in \{0,1\}\ \forall i,\forall j \text{ and } \sum_{j=1}^{|\mathbf{Y}_{\mathbf{x}^{(i)}}|} \mathbf{w}_j^{(i)} = 1, \forall i = 1\ldots m \tag{4.4}$$

4.3.3 Greedy Formulation: MinLoss

In this section, we present one possible way to optimize our desired objective $\min_{\Theta,\mathbf{w}} L_{\mathbf{w}}(\Theta,\mathbf{w})$. It alternates between optimizing over the Θ parameters, and optimizing over $\mathbf{w}$ parameters. While Θ parameters are optimized using SGD, the weights $\mathbf{w}$ are selected greedily for a given $\Theta = \Theta^{(t)}$ at each iteration, *i.e.*, it assigns a non-zero weight to the solution corresponding to the minimum loss amongst all the possible $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ for each $i = 1\ldots m$:

$$\mathbf{w}_j^{(i)^{(t)}} = \mathbb{1}\left\{\mathbf{y}_j^{(i)} = \operatorname*{argmin}_{\mathbf{y}\in\mathbf{Y}_{\mathbf{x}^{(i)}}} l_{\Theta^{(t)}}\left(\hat{\mathbf{y}}^{(i)^{(t)}}, \mathbf{y}\right)\right\}, \forall i = 1\ldots m \tag{4.5}$$

This can be done by computing the loss with respect to each target, and picking the one which has the minimum loss. We refer to this approach as MinLoss. Intuitively, for a given set of $\Theta^{(t)}$ parameters, MinLoss greedily picks the weight vector $\mathbf{w}^{(i)^{(t)}}$, and uses them to get the next set of $\Theta^{(t+1)}$ parameters using SGD update.

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha_\Theta \nabla_\Theta L_{\mathbf{w}}\left(\Theta,\mathbf{w}\right)|_{\Theta=\Theta^{(t)},\mathbf{w}=\mathbf{w}^{(t)}} \tag{4.6}$$

One significant challenge with MinLoss is the fact that it chooses the current set of $\mathbf{w}$ parameters independently for each example based on current Θ values. While this way of picking the $\mathbf{w}$ parameters is optimal if Θ has reached the optima, *i.e.* $\Theta = \Theta^*$, it can lead to sub-optimal choices when both Θ and $\mathbf{w}$ are being simultaneously trained. Following example illustrates this.

Example 3. *Consider a simple task with a one-dimensional continuous input space $\mathcal{X} \subset \mathbb{R}$, and target space $\mathcal{Y} = \{0,1\}$. Consider learning with 10 examples, given as ($\mathbf{x} = 1, \mathbf{Y}_\mathbf{x} = \{1\}$) (5 examples), ($\mathbf{x} = -1, \mathbf{Y}_\mathbf{x} = \{0,1\}$) (4 examples), ($\mathbf{x} = -2, \mathbf{Y}_\mathbf{x} = \{1\}$) (1 example). The optimal decision hypothesis is given as: $\mathbf{y} = \mathbb{1}\{\mathbf{x} > \alpha\}$, for $\alpha \leq -2$, or $\mathbf{y} = \mathbb{1}\{\mathbf{x} < \beta\}$, for $\beta \geq 1$. Assume learning this with logistic regression using MinLoss as the training algorithm optimizing the objective in eq. (4.4). If we initialize the parameters of logistic such that the starting hypothesis is given by $\mathbf{y} = \mathbb{1}\{\mathbf{x} > 0\}$ (logistic parameters: $\theta_1 = 0.1$, $\theta_0 = 0$), MinLoss will greedily pick the target $\mathbf{y} = 0$ for samples with $\mathbf{x} = -1$, repeatedly. This will result in the learning algorithm converging to the decision hypothesis $\mathbf{y} = \mathbb{1}\{\mathbf{x} > -0.55\}$, which is sub-optimal since the input with $\mathbf{x} = -2$ is incorrectly classified (fig. 4.1).*

Proof. For logistic regression, when input $\mathbf{x}$ is one dimensional, probability of the prediction

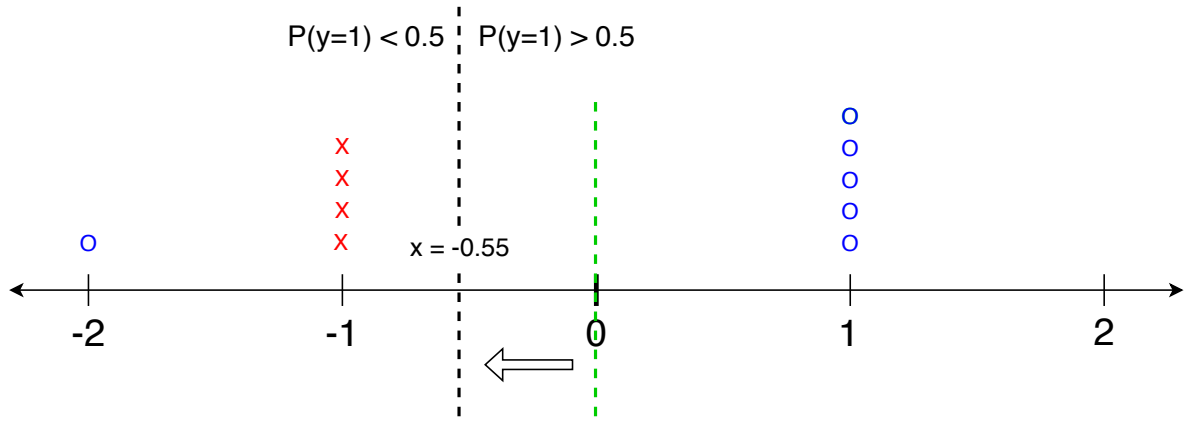


Figure 4.1: Decision Boundary learnt by logistic regression guided by MinLoss. Green vertical line at $\mathbf{x} = 0$ is the initial decision boundary and black vertical line at $\mathbf{x} = -0.55$ is the decision boundary at convergence.

being 1 for any given point $\mathbf{x} = [x]$ is given as:

$$P(\mathbf{y} = 1) = \sigma(\theta_1 x + \theta_0) \quad \text{where} \quad \sigma(z) = \frac{1}{1 + e^{-z}}, z \in \mathbb{R}$$

The decision boundary is the hyperplane on which the probability of the two classes, 0 and 1, is same, *i.e.* the hyperplane corresponding to $P(\mathbf{y} = 0) = P(\mathbf{y} = 1) = 0.5$ or $\theta_1 x + \theta_0 = 0$.

Initially, $\theta_1 = 0.1$ and $\theta_0 = 0$ implies that decision boundary lies at $\mathbf{x} = 0$ (shown in green). All the points on the left of decision boundary are predicted to have 0 label while all the points on the right have 1 label. For all the dual label points ($\mathbf{x} = 1$), $P(\mathbf{y} = 1) < 0.5$, thus MinLoss greedily picks the label 0 for all these points. This choice by MinLoss doesn't change unless the decision boundary goes beyond -1.

However, we observe that with gradient descent using a sufficiently small learning rate, logistic regression converges at $\mathbf{x} = -0.55$ with MinLoss never flipping its choice. Clearly, this decision boundary is sub-optimal since we can define a linear decision boundary ($\mathbf{y} = \mathbb{1}\{\mathbf{x} > \alpha\}$, for $\alpha \leq -2$, or $\mathbf{y} = \mathbb{1}\{\mathbf{x} < \beta\}$, for $\beta \geq 1$) that classifies all the points with label 1 and achieves 100% accuracy. □

MinLoss is not able to achieve the optimum since it greedily picks the target for each query $\mathbf{x}^{(i)}$ based on current set of parameters and gets stuck in local mimima. We address this in the next section.

4.3.4 Reinforcement Learning Formulation: SelectR

In this section, we will design a training algorithm that fixes some of the issues observed with MinLoss. Considering the Example 3 above, the main problem with MinLoss is its

inability to consider alternate targets which may not be greedily optimal at the current set of parameters. A better strategy will try to explore alternative solutions as a way of reaching better optima, *e.g.*, in example 3 we could pick, for the input $\mathbf{x} = -1$, the target $\mathbf{y} = 1$ with some non-zero probability, to come out of the local optima. In the above case, this also happens to be the globally optimal strategy. This is the key motivation for our RL-based strategy proposed below.

A natural questions arises: how should we assign the probability of picking a particular target? A naïve approach would use the probability assigned by the underlying M_Θ network as a way of deciding the amount of exploration on each target $\mathbf{y}$. We call it I-ExplR. We argue below why this may not always be an optimal choice.

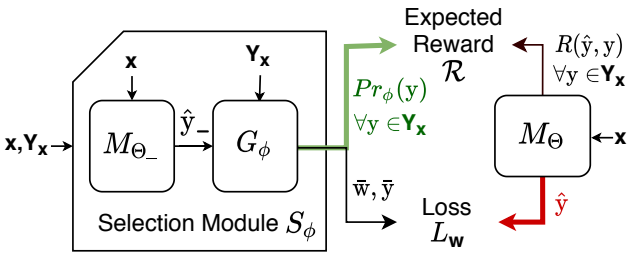


Figure 4.2: Flow-diagram for our RL Framework

We note that the amount of exploration required may depend in complex ways on the global solution landscape, as well as the current set of parameters. Therefore, we propose a strategy, which makes use of a separate *selection module* (a neural network), which takes as input, the current example $(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}})$, and outputs the probability of picking each target for training Θ in the next iteration. Our strategy is RL-based since, we can think of choosing each target (for a given input) as an action that our selection module needs to take. Our selection module is trained using a reward that captures the quality of selecting the corresponding target for training the prediction network. We next describe its details.

Selection Module (S_ϕ): This is an RL agent or a policy network where the action is to select a target, $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$, for each $\mathbf{x}^{(i)}$. Given a training sample, $(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}})$, it first internally predicts $\hat{\mathbf{y}}^{(i)}_ = \mathbf{M}_{\Theta_}(\mathbf{x}^{(i)})$, using a past copy of the parameters $\Theta_$. This prediction is then fed as an input along with the target set, $\mathbf{Y}_{\mathbf{x}^{(i)}}$, to a latent model, G_ϕ, which outputs a probability distribution $Pr_\phi(\mathbf{y}_j^{(i)}), \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$, s.t. $\sum_{\mathbf{y}_j^{(i)}} Pr_\phi(\mathbf{y}_j^{(i)}) = 1$. S_ϕ then picks a target $\bar{\mathbf{y}}^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ based on the distribution $Pr_\phi(\mathbf{y}_j^{(i)})$ and returns a $\mathbf{w}^{(i)}$ such that $\forall i$, $\mathbf{w}_j^{(i)} = 1$ if $\mathbf{y}_j^{(i)} = \bar{\mathbf{y}}^{(i)}$, and $\mathbf{w}_j^{(i)} = 0$ otherwise.

Update of ϕ Parameters: The job of the selection module is to pick one target, $\bar{\mathbf{y}}^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$, for each input $\mathbf{x}^{(i)}$, for training the prediction network $\mathbf{M}_\Theta$. If we were given an oracle to tell us which $\bar{\mathbf{y}}^{(i)}$ is most suited for training $\mathbf{M}_\Theta$, we would have trained the selection module S_ϕ to match the oracle. In the absence of such an oracle, we train S_ϕ using a reward scheme. Intuitively, $\bar{\mathbf{y}}^{(i)}$ would be a good choice for training $\mathbf{M}_\Theta$, if

it is "easier" for the model to learn to predict $\bar{\mathbf{y}}^{(i)}$. In our reward design, we measure this degree of ease using hamming distance between $\bar{\mathbf{y}}^{(i)}$ and $\mathbf{M}_\Theta$'s prediction $\hat{\mathbf{y}}^{(i)}$, *i.e.*, $R(\hat{\mathbf{y}}^{(i)}, \bar{\mathbf{y}}^{(i)}) = \sum_{k=1}^{r} \mathbb{1}\{\hat{\mathbf{y}}^{(i)}[k] = \bar{\mathbf{y}}^{(i)}[k]\}$. We note that there are other choices as well for the reward, *e.g.*, a binary reward, which gives a positive reward of 1 only if the prediction model $\mathbf{M}_\Theta$ has learnt to predict the selected target $\bar{\mathbf{y}}^{(i)}$. Our reward scheme is a granular proxy of this binary reward and makes it easier to get a partial reward even when the binary reward would be 0.

The expected reward for RL can then be written as:

$$\mathcal{R}(\phi) = \sum_{i=1}^{m} \sum_{\bar{\mathbf{y}}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}} Pr_\phi\left(\bar{\mathbf{y}}_j^{(i)}\right) R\left(\hat{\mathbf{y}}^{(i)}, \bar{\mathbf{y}}_j^{(i)}\right) \tag{4.7}$$

We make use of policy gradient to compute the derivative of the expected reward with respect to the ϕ parameters. Accordingly, update equation for ϕ can be written as:

$$\phi^{(t+1)} \leftarrow \phi^{(t)} + \alpha_\phi \nabla_\phi \mathcal{R}\left(\phi\right)|_{\phi=\phi^{(t)}} \tag{4.8}$$

Update of Θ Parameters: Next step is to use the output of the selection module, $\mathbf{w}^{(i)}$ corresponding to the sampled target $\bar{\mathbf{y}}^{(i)}$, $\forall i$, to train the M_Θ network. The update equation for updating the Θ parameters during next learning iteration can be written as:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha_\Theta \nabla_\Theta L_{\mathbf{w}}\left(\Theta, \mathbf{w}\right)|_{\Theta=\Theta^{(t)}, \mathbf{w}=\mathbf{w}^{(t)}} \tag{4.9}$$

Instead of backpropagating the loss gradient at a sampled target $\bar{\mathbf{y}}^{(i)}$, one could also backpropagate the gradient of the expected loss given the distribution $Pr_\phi(\mathbf{y}_j^{(i)})$. In our experiments, we backpropagate through the expected loss since our action space for the selection module S_ϕ is tractable. Figure 4.2 represents the overall framework. In the diagram, gradients for updating Θ flow back through the red line and gradients for updating ϕ flow back through the green line.

4.3.5 Training Algorithm

We put all the update equations together and describe the key components of our training algorithm below. Algorithm 2 presents a detailed pseudocode.

Pre-training: It is a common strategy in many RL based approaches to first pre-train the network weights using a simple strategy. Accordingly, we pre-train both the M_Θ and S_ϕ networks before going into joint training. First, we pre-train $\mathbf{M}_\Theta$. In our experiments, we observe that in some cases, pre-training $\mathbf{M}_\Theta$ using only those samples from training data $\mathcal{D}$ for which there is only a unique solution, *i.e.*, $\{(\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}}) \in \mathcal{D} \text{ s.t. } |\mathbf{Y}_{\mathbf{x}^{(i)}}| = 1\}$ gives better performance than pre-training with MinLoss. Therefore, we pre-train using both the approaches and select the better one based on their performance on a held out

Algorithm 2 Joint Training of Prediction Network $\mathbf{M}_\Theta$ & Selection Module S_ϕ

$\Theta_0 \leftarrow$ **Pre-train** Θ using eq. (4.5) and eq. (4.6)
In Selection Module (SM): $\Theta_ \leftarrow \Theta_0$
$\phi_0 \leftarrow$ **Pre-train** ϕ using rewards from $\mathbf{M}_\Theta$ in eq. (4.8)
Initialize: $t \leftarrow 0$
while *not converged* **do**
  $B \leftarrow$ Randomly fetch a mini-batch
  for $i \in B$ **do**
    Get weights: $\mathbf{w}^{(i)} \leftarrow S_\phi((\mathbf{x}^{(i)}, \mathbf{Y}_{\mathbf{x}^{(i)}}), \Theta_)$
    Get model predictions: $\hat{\mathbf{y}}^{(i)} \leftarrow \mathbf{M}_{\Theta^{(t)}}(\mathbf{x}^{(i)})$
    Get rewards: $\mathbf{r}^{(i)} \leftarrow [R(\hat{\mathbf{y}}^{(i)}, \mathbf{y}_j^{(i)}),\ \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}]$
  end
  Update ϕ: Use eq. (4.8) to get $\phi^{(t+1)}$
  Update Θ: Use eq. (4.9) to get $\Theta^{(t+1)}$
  Update $\Theta_ \leftarrow \Theta^{(t+1)}$ if $t\%copyitr = 0$ (in Selection Module)
  Increment $t \leftarrow t + 1$
end

dev set. Once the prediction network is pre-trained, a copy of it is given to the selection module to initialize $\mathbf{M}_{\Theta_}$. Keeping Θ and $\Theta_$ fixed and identical to each other, the latent model, G_ϕ, in the selection module is pre-trained using the rewards given by the pre-trained $\mathbf{M}_\Theta$ and the internal predictions given by $\mathbf{M}_{\Theta_}$.

Joint Training: After pre-training, both prediction network $\mathbf{M}_\Theta$ and selection module S_ϕ are trained jointly. In each iteration t, selection module first computes the weights, ${\mathbf{w}^{(i)}}^t$, for each sample in the mini-batch. The prediction network computes the prediction ${\hat{\mathbf{y}}^{(i)}}^t$ and rewards $R({\hat{\mathbf{y}}^{(i)}}^t, \mathbf{y}_j^{(i)}), \forall \mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$. The parameters ϕ^t and Θ^t are updated simultaneously using eq. (4.8) and eq. (4.9), respectively. The copy of the prediction network within selection module, *i.e.*, $\mathbf{M}_{\Theta_}$ in S_ϕ, is updated with the latest parameters Θ^t after every *copyitr* updates where *copyitr* is a hyper-parameter.

4.4 Experiments with Non-autoregressive Models

The main goal of our experiments is to evaluate the four multiplicity aware methods: CC-Loss, MinLoss, informed exploration (I-ExplR) and RL based exploration (SelectR), when compared to baseline approaches that completely disregard the problem of solution multiplicity. We also wish to assess the performance gap, if any, between queries with a unique solution and those with many possible solutions. To answer these questions, we conduct experiments on three different tasks (N-Queens, Futoshiki & Sudoku), trained over two different prediction networks. We repeat all the experiments thrice using different

seeds and report the mean and standard error. Further details of software environments, optimizer, hyperparameters and dataset generation are in appendix B.1.

4.4.1 Datasets and Prediction Networks

N-Queens: Given a query, *i.e.*, a chess-board of size $N \times N$ and a placement of $k < N$ non-attacking queens on it, the task of N Queens is to place the remaining $N - k$ queens, such that no two queens are attacking each other. We train a Neural Logic Machine (NLM) model [Dong et al., 2019] as the prediction network $\mathbf{M}_\Theta$ for solving queries for this task. To model N-Queens within NLM, we represent a query $\mathbf{x}$ and the target $\mathbf{y}$ as N^2 dimensional Boolean vectors with 1 at locations where a Queen is placed. We use another smaller NLM architecture as the latent model G_ϕ.

We train our model on 10–Queens puzzles and test on 11–Queens puzzles, both with 5 placed queens. This size-invariance in training and test is a key strength of NLM architecture, which we exploit in our experiments. To generate the train data, we start with all possible valid 10–Queens board configurations and randomly mask any 5 queens, and then check for all possible valid completions to generate potentially multiple solutions for an input. Test data is also generated similarly. Training and testing on different board sizes ensures that no direct information leaks from test to train. Queries with multiple solutions have 2-6 solutions, so we choose $\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}, \forall \mathbf{x}^{(i)}$. See section B.1.1.1 in appendix for more details.

Futoshiki: This is a logic puzzle in which we are given a grid of size $N \times N$, and the goal is to fill the grid with digits from $\{1 \dots N\}$ such that no digit is repeated in a row or a column. k out of N^2 positions are already filled in the input query $\mathbf{x}$ and the remaining $N^2 - k$ positions need to be filled. Further, inequality constraints are specified between some pairs of adjacent grid positions, which need to be honored in the solution. Our prediction network, and latent model use NLM, and the details are very similar to that of N–Queens.

Similar to N–Queens, we do size-invariant training – we train our models on 5×5 puzzles with 14 missing digits and test on 6×6 puzzles with 20 missing digits. Similar to N–Queens, we generate all possible valid grids and randomly mask out the requisite number of digits to generate train and test data. For both train and test queries we keep up to five inequality constraints of each type: $>$ and $<$. See section B.1.1.2 in appendix for more details.

Sudoku: We also experiment on Sudoku, which has been used as the task of choice for many recent neural reasoning works [Palm et al., 2018, Wang et al., 2019]. We use Relational Recurrent Networks (RRN) [Palm et al., 2018] as the prediction network since

it has recently shown state-of-the-art performance on the task. We use a 5 layer CNN as our latent model G_ϕ. Existing Sudoku datasets [Royle, 2014, Park, 2018], do not expose the issues with solution multiplicity. In response, we generate our own dataset by starting with a collection of Sudoku puzzles with unique solutions that have 17 digits filled. We remove one of the digits, thus generating a puzzle, which is guaranteed to have solution multiplicity. We then randomly add 1 to 18 of the digits back from the solution of the original puzzle, while ensuring that the query continues to have more than 1 solution. This generates our set of multi-solution queries with a uniform distribution of filled digits from 17 to 34. We mix an equal number of unique solution queries (with same filled distribution). Because some $\mathbf{x}^{(i)}$s may have hundreds of solutions, we randomly sample 5 of them from $\mathcal{Y}_{\mathbf{x}^{(i)}}$, i.e., $|\mathbf{Y}_{\mathbf{x}^{(i)}}| \leq 5$ in the train set. For each dataset, we generate a devset in a manner similar to the test set. See section B.1.1.3 in appendix for more details.

Table 4.1: Statistics of datasets. 'Train', 'Test' and task names are abbreviated. Devset similar to test.

	N-Qn (Tr)	**N-Qn (Tst)**	**Futo. (Tr)**	**Futo. (Tst)**	**Sud. (Tr)**	**Sud. (Tst)**
# of queries	165,744	10,000	10,000	10,000	20,000	10,000
%age of *MS* queries	7.04%	16.67%	17.05%	24.95%	50%	50%
Avg solns per *MS* query	2.1	2.2	2.2	2.4	13.8	13.7

4.4.2 Baselines and Evaluation Metric

Our comparison baselines include: (1) *Naïve*: backpropagating $L(\Theta)$ through each solution independently using Equation (4.1), (2) *Unique*: computing $L(\Theta)$ only over the subset of training examples that have a unique solution, and (3) *Random*: backpropagating $L(\Theta)$ through one arbitrarily picked solution $\mathbf{y}^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$ for every $\mathbf{x}^{(i)}$ in the train data, and keeping this choice fixed throughout the training.

We separately report performance on two mutually exclusive subsets of test data: *OS*: queries with a unique solution, and *MS*: those with multiple solutions. For all methods, we tune various hyperparameters (and do early stopping) based on the devset performance. Additional parameters for the four multiplicity aware methods include the ratio of *OS* and *MS* examples in training.[4] I-ExplR and SelectR also select the pre-training strategy as described in Section 4.3.5. For all tasks, we consider the output of a prediction network as correct only if it is a valid solution for the underlying CSP. No partial credit is given for guessing parts of the output correctly.

[4]Futoshiki and N–Queens training datasets have significant *OS*-*MS* imbalance (see Table 4.1), necessitating managing this ratio by undersampling *OS*. This is similar to standard approach in class imbalance problems.

Table 4.2: Mean test accuracy (±standard error) over three runs for multiplicity aware methods compared with baselines. *OS*: test queries with only one solution, *MS*: queries with more than one solution.

		Naïve	*Random*	*Unique*	CC-Loss	MinLoss	I-ExplR	SelectR
N-Queens	***OS***	70.59 ± 0.09	75.09 ± 0.33	72.91 ± 0.65	75.31 ± 0.45	77.29 ± 0.38	77.35 ± 1.07	**79.73 ± 0.34**
	MS	55.34 ± 2.82	66.85 ± 2.46	61.13 ± 1.13	75.76 ± 1.60	77.22 ± 1.28	79.46 ± 3.31	**79.68 ± 1.35**
	Overall	68.04 ± 0.46	73.72 ± 0.59	70.94 ± 0.71	75.39 ± 0.47	77.28 ± 0.48	77.70 ± 1.40	**79.72 ± 0.46**
Futoshiki	***OS***	65.59 ± 0.62	67.63 ± 0.96	65.49 ± 0.28	77.68 ± 0.34	76.78 ± 0.81	**78.15 ± 0.65**	78.01 ± 0.70
	MS	14.99 ± 2.17	19.13 ± 3.14	14.22 ± 2.77	69.30 ± 1.76	70.35 ± 1.16	70.88 ± 1.48	**71.57 ± 1.02**
	Overall	52.96 ± 0.96	55.53 ± 1.44	52.70 ± 0.74	75.59 ± 0.46	75.18 ± 0.64	76.33 ± 0.65	**76.40 ± 0.36**
Sudoku	***OS***	87.85 ± 0.84	89.19 ± 1.12	**87.53 ± 0.82**	88.26 ± 0.88	88.25 ± 0.35	88.73 ± 0.68	88.69 ± 0.55
	MS	09.13 ± 0.89	66.39 ± 2.82	13.65 ± 1.79	76.58 ± 1.63	76.93 ± 1.50	80.19 ± 1.51	**81.73 ± 2.00**
	Overall	48.49 ± 0.86	77.79 ± 1.96	50.59 ± 0.49	82.42 ± 0.45	82.59 ± 0.62	84.46 ± 0.69	**85.21 ± 0.76**

4.4.3 Results and Discussion

We report the accuracies across all tasks and models in Table 4.2. For each setting, we report the mean and standard error over three random runs (with different seeds). We first observe that *Naïve* and *Random* perform significantly worse than *Unique* in all the tasks, not only on *MS*, but on *OS* as well. This suggests that, 1oML models that explicitly handle solution multiplicity, even if by simply discarding multiple solutions, are much better than those that do not recognize it at all.

Predictably, all multiplicity aware methods vastly improve upon the performance of naïve baselines, with a dramatic 13-52 pt gains between *Unique* and SelectR on queries with multiple solutions.

Comparing MinLoss and SelectR, we find that our RL-based approach outperforms MinLoss consistently, with p-values (computed using McNemar's test for the best models selected based on validation set) of 1.00e−16, 0.03, and 1.69e−18 for NQueens, Futoshiki and Sudoku respectively (see section B.1.2 in appendix for seed-wise comparisons of gains across tasks). On the other hand, informed exploration technique, I-ExplR, though improves over MinLoss on two out of three tasks, it performs worse than SelectR in all the domains. This highlights the value of RL based exploration on top of the greedy target selection of MinLoss as well as over the simple exploration of I-ExplR. We note that this is due to more exploratory power of SelectR over I-ExplR, which we discuss in the end.

Impact of variation in the number of solutions: Recall that Sudoku training set has no more than 5 solutions for a query, irrespective of the actual number of solutions – *i.e.*, for many $\mathbf{x}^{(i)}$, $\mathbf{Y}_{\mathbf{x}^{(i)}} \subsetneq \mathcal{Y}_{\mathbf{x}^{(i)}}$. Despite incomplete solution set, significant improvement over baselines is obtained, indicating that our formulation handles solution multiplicity

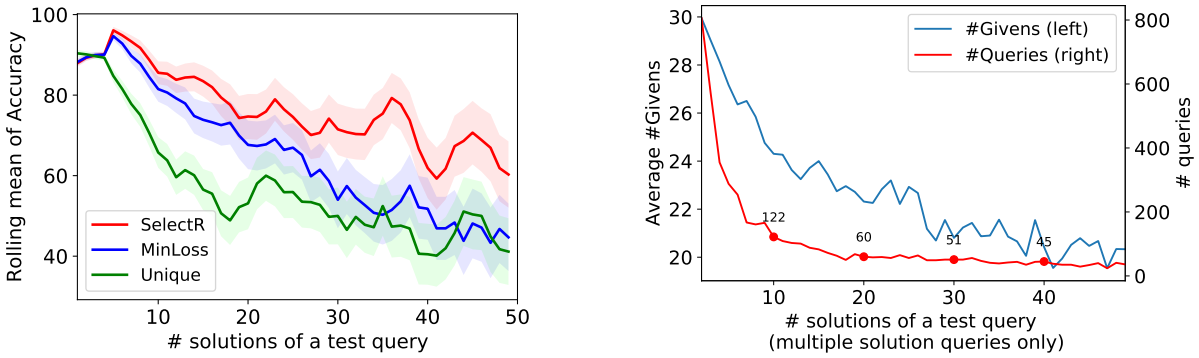


Figure 4.3: Left: Accuracy vs size of query's solution set (with 95% confidence interval). Right: #givens and #datapoints vs size of query's solution set

even with incomplete information.

Furthermore, the large variation in the size of solution set ($|\mathcal{Y}_{\mathbf{x}}|$) in Sudoku allows us to assess its effect on the overall performance. We find that all models get worse as $|\mathcal{Y}_{\mathbf{x}}|$ increases (fig. 4.3), even though SelectR remains the most robust to the variation in the size of solution set. To do this analysis, we divide the test data into different bins based on the number of possible solutions for each test input ($\mathbf{x}^{(i)}$) and compare the performance of the best model obtained in the three settings: *Unique*, MinLoss and SelectR.

By construction, the number of test points with a unique solution is equal to the total number of test points with more than one solution. Further, while creating the puzzles with more than one solution, we ensured uniform distribution of number of filled cells from 17 to 34, as is done in [Palm et al., 2018] for creating puzzles with unique solutions in their paper. Hence, the number of points across different bins (representing solution count) may not be the same. Right figure in 4.3 shows the average size of each bin and the average number of filled cells for multiple solution queries in a bin. As we move to the right in graph (i.e., increase the number of solutions for a given problem), the number of filled cells in the corresponding Sudoku puzzles decreases, resulting in harder problems. This is also demonstrated by the corresponding decrease in performance of all the models in the left figure in fig. 4.3. SelectR is most robust to this decrease in performance.

Discussion on Why SelectR is better than I-ExplR?

Here we argue why SelectR is more powerful than I-ExplR, even though the reward structure for training the RL agent is such that eventually the G_ϕ in the RL agent will learn to pick the target closest to the current prediction (to maximize reward), and hence S_ϕ will be reduced to I-ExplR. We see two reasons why SelectR is better than I-ExplR.

First, recall that the I-ExplR strategy gives the model an exploration probability based on its current prediction. But note that this is "only one" of the possible exploration

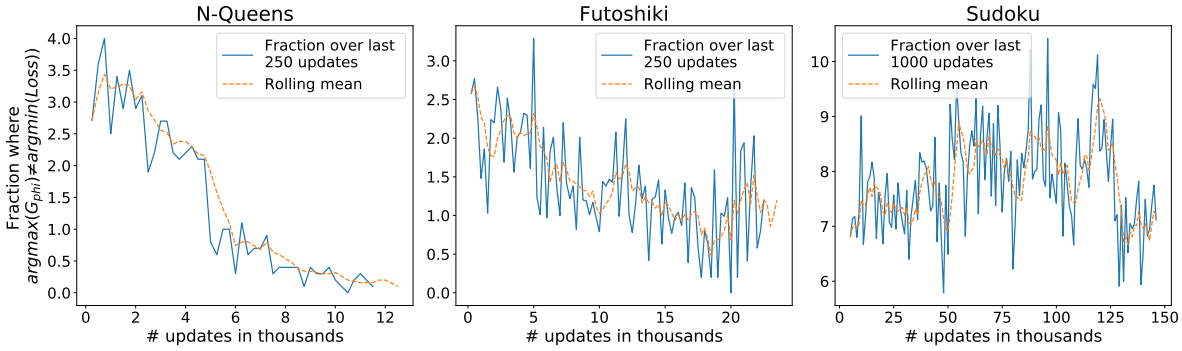


Figure 4.4: Fraction of training samples for which arg max of G_ϕ probability distribution is different from the target closest to model prediction. For I-ExplR, this fraction is 0%

strategies. For example, another strategy could be to explore based on a fixed epsilon probability. There could be several other such possible exploration strategies that could be equally justified. Instead of hard coding them, as done for I-ExplR, our G_ϕ network gives the ability to learn the best exploration strategy, which may depend in complex ways on the global reward landscape (*i.e.*, simultaneously optimizing reward over all the training examples). Hence we use a neural module for this.

Second, note that I-ExplR is parameter-free and fully dependent on $\mathbf{M}_\Theta$, thus, has limited representational power of its own to explore targets. This is not the case with G_ϕ. Its output $\bar{\mathbf{y}}$ and and the target ($\mathbf{y}_c$) closest to $\mathbf{M}_\Theta$ prediction $\hat{\mathbf{y}}$ may differ i.e. $\bar{\mathbf{y}} \neq \mathbf{y}_c$ (see next paragraph for an experiment on this). When this happens, the gradients will encourage change in Θ so that $\hat{\mathbf{y}}$ moves towards $\bar{\mathbf{y}}$, and simultaneously encourage change in ϕ so that $\bar{\mathbf{y}}$ moves towards $\mathbf{y}_c$. That is, a stable alignment between the two models could be either of the two, $\mathbf{y}_c$ or $\bar{\mathbf{y}}$. This, we believe, increases the overall exploration of the model. Which of $\mathbf{y}_c$ or $\bar{\mathbf{y}}$ get chosen depends on how strongly the global landscape (other data points) encourage one versus the other. Such flexibility is not available to I-ExplR where only Θ parameters are updated. We believe that this flexibility to explore more could enable SelectR to jump off early local optima, thus achieving better performance compared to I-ExplR.

We provide preliminary experimental evidence that supports that SelectR explores more. For every training data point q, we check if the arg max of G_ϕ probability distribution (i.e., highest probability $\bar{\mathbf{y}}$) and $\mathbf{y}_c$ differ from each other. We name such data points "exploratory". We analyze the fraction of exploratory data points as a function of training batches. See fig. 4.4. We observe that in the initial several batches, SelectR has $3-10\%$ of training data exploratory. This number is, by definition, 0% for I-ExplR since it chooses $\bar{\mathbf{y}}$ based on model probabilities. This experiment suggests that SelectR may indeed explore more early on.

4.5 Experiments with Auto-regressive Models

Recall that we motivated our multiplicity-aware loss function (Equation (4.3)) by demonstrating the incapability of non-autoregressive models to represent all possible distributions over the solution set $\mathbf{Y}_\mathbf{x}$. For example, a non-autoregressive model can't represent a uniform distribution (or any distribution) with only $\{(0,1),(1,0)\}$ as its support. While doing so, it ends up assigning a probability of 0.5 (in case of uniform distribution) to both the values 0 and 1 ($P(\mathbf{y}[k]=0) = P(\mathbf{y}[k]=1) = 0.5$), independently for the two dimensions ($k \in \{1,2\}$). This results in a uniform distribution over the entire output space $\mathcal{Y} = \{(0,0),(0,1),(1,0),(1,1)\}$.

This, however, is not the case for autoregressive models that can represent a certain class of non-trivial joint distributions. For example, $Pr(\mathbf{y}[1],\mathbf{y}[2]|\mathbf{x})$ could be modeled as $Pr(\mathbf{y}[1]|\mathbf{x})Pr(\mathbf{y}[2]|\mathbf{y}[1];\mathbf{x})$. One can easily see that to represent a distribution with only $\{(0,1),(1,0)\}$ as its support, it may assign any probability to digit 0 for the first dimension $\mathbf{y}[1]$, and assign a probability of 1 to digit 0 when $\mathbf{y}^{(i)}[1]=1$, and a probability of 1 to digit 1 when $\mathbf{y}^{(i)}[1]=0$. That is, $P(\mathbf{y}[1]=0|\mathbf{x}) = \alpha, \alpha \in [0,1]$, and $P(\mathbf{y}[2]=1|\mathbf{y}[1]=0;\mathbf{x}) = 1$ and $P(\mathbf{y}[2]=0|\mathbf{y}[1]=1;\mathbf{x}) = 1$.

The above observation warrants a question: if autoregressive models can represent multiple solutions, do we need to modify the naïve-loss when training an autoregressive model? To answer this question empirically, we train an encoder-decoder autoregressive model on the sudoku dataset using all the techniques introduced in this work and compare its performance with its non-autoregressive counterpart.

4.5.1 Prediction Network

In our earlier experiments (section 4.4), we trained an RRN [Palm et al., 2018] for sudoku, which was then the state of the art model. However, recent research [Yang et al., 2023] achieved s-o-t-a performance using a transformer-based encoder, and hence, we chose the same architecture as the backbone of our autoregressive encoder-decoder model. For a fair comparison between autoregressive and non-autoregressive models, we train two new models: 1) a transformer-based encoder (non-autoregressive) and 2) an autoregressive transformer-based encoder-decoder. In both the models, our encoder is the same as suggested in Yang et al. [2023]: a 32-layer recurrent transformer with shared weights, multi-headed attention with 4 heads, and 128 dimensional embedding space per head. As in RRN, we compute loss using the logits from all 32 layers.

For the decoder, we first arbitrarily chose row-major as the order in which we decode the sudoku puzzle. We keep its architecture identical to the encoder, except we use a causal

mask while training to ensure it remains auto-regressive. While training, we compute loss using the logits of all 32 layers of the encoder and decoder, whereas inference is done using only logits from the last layer of the decoder. Unlike a fully connected graph in Transformers, the graph created by Palm et al. [2018] for sudoku connects a cell only with other cells in the same row, column and the same 3×3 block. Accordingly, we add an appropriate attention mask to provide similar information to our transformer-based encoder and decoder.

4.5.2 Results and Discussion

Training and Inference time: Before comparing the accuracies, we highlight a huge discrepancy in the training and inference time for an encoder-decoder model compared to that of an encoder-only model.

While training an encoder-decoder model, we need to aggregate loss from 64 layers (32 from the encoder and decoder each) compared to only 32 layers in an encoder-only model. In addition, for an input $\mathbf{x}$, we need to pass each $(\mathbf{x}, \mathbf{y}^{(i)})$ separately through the model to compute its loss (unlike the encoder-only model where for all the $\mathbf{y}^{(i)}$'s, we pass $\mathbf{x}$ through the model only once and then compute the loss for each $\mathbf{y}^{(i)}$ using the logits). These factors result in at least 3x more training time for an encoder-decoder than for the encoder-only model.

During inference, a decoder generates only one digit at a time. This is implemented by calling the forward pass of the decoder 81 times for each sudoku puzzle, resulting in considerably higher inference time (more than 20x!). Recall that in SelectR, Selection Module needs to call inference in each training iteration (step 9 in Algorithm 2). With such high inference time, it is infeasible to train encoder-decoder with SelectR. Note that I-ExplR uses only the loss of each target to compute the exploration probabilities, and hence trains in a reasonable time.

Performance comparison: We compare the performance of an autoregressive encoder-decoder model with its encoder-only counterpart in Table 4.3. For easy reference, we also copied in the same table the mean accuracy obtained by non-autoregressive RRN (from table 4.2). We make the following crucial observations:

1. As expected, when trained using naïve-loss, the autoregressive encoder-decoder obtains the best performance (70.72 vs 49.98 overall). But note that this gain comes at a higher training (3x) and a much higher inference (20x) time.

2. When we account for solution multiplicity by modifying the loss functions, and hence the output distribution that the model needs to learn, the added advantage

Table 4.3: Comparing the accuracy of an autoregressive encoder-decoder with a non-autoregressive encoder. For easy reference, we copy from Table 4.2 the mean accuracy obtained by RRN. *OS*: test queries with only one solution, *MS*: queries with more than one solution.

	RRN			**Transformer Encoder**			**Transformer Encoder-Decoder**		
Loss	**MS**	**OS**	**Overall**	**MS**	**OS**	**Overall**	**MS**	**OS**	**Overall**
Naïve	9.13	87.85	48.49	19.70	80.26	49.98	45.36	96.08	**70.72**
Random	13.65	87.53	50.59	21.30	82.36	51.83	47.98	96.42	**72.20**
Unique	66.39	89.19	77.79	73.42	77.08	75.25	81.79	83.00	**82.40**
CC-Loss	76.58	88.26	82.42	77.36	86.56	81.96	84.41	91.34	**87.88**
MinLoss	76.93	88.25	82.59	84.83	91.18	**88.01**	84.65	90.54	87.60
I-ExplR	80.19	88.73	84.46	82.65	93.66	88.16	84.91	92.26	**88.59**
SelectR	81.73	88.69	85.21	86.77	94.32	**90.55**			

of better representation power vanishes. For both MinLoss (88.01 vs 87.60) and exploration–based I-ExplR(88.16 vs 88.59), the performance of encoder-only is comparable to the encoder-decoder models.

3. We observe that even for the s-o-t-a transformer-based encoder only model, SelectR achieves the best performance (gain of more than 2 pts over I-ExplR), underscoring the importance of 'learnable' exploration over 'deterministic' exploration of I-ExplR.

4. Finally, we observe that our best non-auto-regressive model trained using SelectR performs better than our best auto-regressive model trained using I-ExplR (90.55 vs 88.59), and that too with a much lower training and inference cost. This highlights the importance of enhancing the learnability (by modifying loss functions and training techniques) over improving the representation power (by using auto-regressive models) for addressing solution multiplicity.

4.6 Conclusion

In this work, we have defined 1oML: the task of learning one of many solutions for combinatorial problems in structured output spaces. We have identified solution multiplicity as an important aspect of the problem, which if not handled properly, may result in sub-optimal models. As a first cut solution, we proposed a greedy approach: MinLoss formulation. We identified certain shortcomings with the greedy approach and proposed two exploration based formulations: I-ExplR and an RL formulation, SelectR, which overcomes some of the issues in MinLoss by exploring the locally sub-optimal choices

for better global optimization. Experiments on three different tasks using two different prediction networks demonstrate the effectiveness of our approach in training robust models under solution multiplicity. On one of the tasks, we also establish the significance of our proposed techniques for training an auto-regressive model.

In our experiments, we observe that the backbone model NLM [Dong et al., 2019] is amenable to size invariance (we train on 10 Queens and test on 11 Queens; train on 5×5 futoshiki and test on 6×6 futoshiki) but it does not scale to problems of size 9×9 sudoku. On the other hand, the state-of-the-art model RRN [Palm et al., 2018] for sudoku is not size invariant even though it is scalable. In the next chapter, we formally define and tackle this issue of output space invariance and propose two architectures based on RRN. We train both our models on 9×9 sudokus and test on 16×16 sudokus. One of our models which is memory efficient can be scaled even upto puzzles of size 25×25 sudokus.

Chapter 5

Neural Models for Output-Space Invariance in Combinatorial Problems

In the experiments of the last section, we observed that one drawback of the architectures like RRN [Palm et al., 2018], which are often based on Graph Neural Networks (GNN) [Zhou et al., 2020], is that they cannot generalize across the size of the output space from which variables are assigned a value, for example, set of colors in a GCP, or board-size in sudoku. We call the output space for the variables as 'value-set'. While many works have demonstrated generalization of GNNs across graph size, there has been no study on how to design a GNN for achieving value-set invariance for problems that come from the same domain. For example, learning to solve 16×16 sudoku after being trained on only 9×9 sudokus, or coloring a 7 colorable graph after training on 4 colorable graphs. In this work, we propose novel methods to extend GNN based architectures to achieve value-set invariance. Specifically, our model builds on recently proposed RRN [Palm et al., 2018]. Our first approach exploits the graph-size invariance of GNNs by converting a multi-class node classification problem into a binary node classification problem. Our second approach works directly with multiple classes by adding multiple nodes corresponding to the values in the value-set, and then connecting variable nodes to value nodes depending on the problem initialization. Our experimental evaluation on three different combinatorial problems demonstrates that both our models perform well on our novel problem, compared to a generic neural reasoner.

5.1 Introduction

In the last chapter we experimented with two models (RRN [Palm et al., 2018] and NLM [Dong et al., 2019]) for solving combinatorial problems in structured output spaces. These methods implicitly incorporate the rules of the puzzle in their weights by training over some of its solved instances. While NLM can potentially train and test on problems of different sizes (*e.g.*, train using 10–Queens and test on 11–Queens, it fails to scale to even moderate size problems. On the other hand, RRN, along with other such models, *e.g.*, SATNet [Wang et al., 2019], can easily scale to large problems such as 9×9 sudokus, but they assume a fixed value-set, *i.e.*, the set from which variables are assigned values is assumed to be constant during training and testing. This is a significant limitation, since it may not always be possible to generate sufficient training data for similar large problems in which variables take values from a bigger value-set [Najafian et al., 2018].

It is also a desirable goal since as humans, we often find it natural to generalize to problems of unseen variable and value sizes, once we know how to solve similar problems of a different size, *e.g.*, we may solve a 12×12 sudoku after learning to solve a 9×9 sudoku. We note that graph based models have been shown to generalize well on varying graph sizes, *e.g.*, finding a satisfying solution of a CNF encoding of a CSP with 100 Boolean-variables, after training on CNF encodings of CSPs with only 40 Boolean-variables [Selsam et al., 2019]. However, the model trained using CNF encoding of Boolean-CSPs cannot be used directly for a non-Boolean CSP in which variables take value from a different (larger) value-set.

In response, we study value-set invariance in combinatorial puzzles from the same domain. To formally define a similar puzzle with variables taking values from a different value-set, we make use of *Lifted CSP* [Joslin and Roy, 1997], a (finite) first-order representation that can be ground to CSPs of varying variable and value-set sizes. We note that even though we use Lifted CSPs to define value-set invariance, its complete specification is assumed to be unknown. Specifically, we do not have access to the constraints of the CSP, and thus neural SAT solvers like NeuroSAT [Selsam et al., 2019] can not be used. While training, we only assume access to solved instances along with their constraint graph. We define our problem as: given solved instances and corresponding constraint graph of an unknown ground CSP with a value-set of size k, can we learn neural models that generalize to instances of the same lifted CSP, but with a different value-set of size k' (typically $k' > k$)? An example task includes training a model using data of 9×9 Sudoku, but testing on a 12×12 or a 16×16 Sudoku. We build our solution using RRNs as the base architecture. They run GNN on the constraint graph, and employ iterative message passing in a recurrent fashion – the nodes (variables) are then decoded to obtain

a solution. We present two ways to enhance RRNs for value-set invariance.

Binarized Model: Our first model converts a multi-class classification problem into a binary classification problem by converting a multi-valued variable into multiple Boolean variables, one for each value in the value-set. The binarized constraint graph gets defined as: if there is an edge between two variables in original constraint graph, there are k edges between Boolean nodes corresponding to the same value and the same two variables in the new graph. In addition, all k Boolean variables, corresponding to a multi-valued variable, are connected with each other. This model naturally achieves value-set invariance. At test time, a larger value-set just results in a larger graph size. All GNN weights are tied, and because all the variables in the binarized model are Boolean, embeddings for binary values '0' and '1', trained at training time, are directly applicable at test time.

Multi-valued Model: Our second model directly operates on the given multi-valued variables and the corresponding constraint graph, but introduces a *value node* for every value in the value-set. Each pre-assigned (unassigned) variable node is connected to that (respectively, every possible) value node. The challenge in this model is initializing value nodes at test time when $k' > k$. We circumvent this problem by training upfront k' or more value embeddings by randomly sub-selecting a k sized subset during each learning iteration. This random sub-selection exploits the symmetry of value-set elements across instances. During test time, k' of the learned embeddings are used.

We perform extensive experimental evaluation on puzzles generated from three different structured CSPs: Graph Coloring (GCP), Futoshiki, and Sudoku. We compare two of our models with NLM [Dong et al., 2019] baseline. – as expected, it either fails to scale or performs worse for most test sizes used in our experiments. We also compare our two models along the axes of performance and scalability and discuss their strengths and weaknesses.

5.2 Related Work and Background

This work belongs to the broad research area of neural reasoning models, in which neural models learn to solve pure reasoning tasks in a data-driven fashion. Some example tasks include theorem proving [Rocktäschel et al., 2015, Evans and Grefenstette, 2018], logical reasoning [Cingillioglu and Russo, 2019], probabilistic logic reasoning [Manhaeve et al., 2018], classical planning [Dong et al., 2019], probabilistic planning in a known MDP [Tamar et al., 2017, Bajpai et al., 2018], and our focus – combinatorial problems that are instances of an unknown constraint satisfaction problem.

There are two main research threads within neural CSPs and SAT. First thread builds

neural models for problems where the CSP constraints or SAT clauses are *explicitly* provided to the model. For example, NeuroSAT [Selsam et al., 2019] and PDP [Amizadeh et al., 2019b] assume that the CSP is expressed in a Conjunctive (or Disjunctive) Normal Form. Similarly, Circuit-SAT [Amizadeh et al., 2019a] uses the knowledge of exact constraints to convert a CSP into a Boolean Circuit. This research has similarities with logical reasoning models like DeepProbLog [Manhaeve et al., 2018], and DeepLogic [Cingillioglu and Russo, 2019], which require human designed rules for reasoning. Our work belongs to the second thread where the constraints or clauses are not provided explicitly, and only some underlying structure (e.g., Sudoku grid cell connectivity) is given along with training data. The intention is that the model not only learns to reason for the task, but also needs to learn the implicit semantics of each constraint. SATNet [Wang et al., 2019] falls in this category – it formulates a learnable low-rank Semi-definite Program (SDP) relaxation for a given MAXSAT problem trained via solved SAT problems. Similarly, Recurrent Relational Networks (RRN) [Palm et al., 2018] use recurrent message passing graph neural network to embed the variables of the unknown CSP, and the relationship between them, in a latent vector space and finally assign a value to each variable based on its embedding. Both these works assume a fixed number of variables that remains unchanged across training and test. While we build on RRNs, we substantially extend the formalism to study value-set invariance. Formally, our work can be seen as solving a (finite) first-order formulation of the CSP, called *Lifted* CSP [Joslin and Roy, 1997], which can be grounded to CSPs with varying number of variables and values. To our knowledge, there is relatively limited prior work on neural models that can generalize to variable-sized instances of an underlying first order reasoning task – one related approach builds neural models for First-order MDPs [Garg et al., 2020].

Finally, there has been a long history of work dedicated to learning rules or constraints from training data using Inductive Logic Programming [Lavrac and Raedt, 1995, Friedman et al., 1999]. Evans and Grefenstette [2018] propose differentiable neural relaxation of ILP (∂ILP). As observed in the last chapter, Neural Logic Machines (NLM) [Dong et al., 2019] learns lifted rules and is shown to be more scalable than ∂ILP. It allows learning of first-order logic rules expressed as Horn Clauses over a set of predicates. Learning of first-order rules makes NLM amenable to transfer over different CSP sizes [Nandwani et al., 2021], and are thus directly comparable to our work. As observed earlier, the main challenge of such approaches is that they fail to scale to the size of the problems considered in this work. In our experiments, we compare our methods against both deep and shallow versions of NLM.

5.2.1 Generalization of GNNs across Graph Sizes

Our work relies heavily on the assumption that GNNs generalize across size. Here we briefly discuss about some of the works that focus on this capability of GNNs.

The existing set of papers (and results) in this line of research can be broadly divided into two sub-classes. The first set of results talk about the *representation power* of GNNs to handle various graph sizes. The second set of results talk about *learnability* issues with GNNs under varying train/test distributions. We look at some of the results below and try to explain why GNNs in our case are able to generalize well, both in terms of representation power, as well as learnability.

- **Representation Power:** We hypothesize that there are two design choices that are helping us gain good representation power: 1. Ability to create a deep network without blowing up the number of parameters because of weight tying across layers, and 2. Preassigned class labels to some of the variables which act as node features and help in breaking the symmetry. We argue it on the basis of Theorem 4.2 in Yehudai et al. [2021], which proves that there exists a $(d+3)$ layered GNN that can distinguish between nodes having different local structure, which is quantified via d-patterns that can be thought of as a generalization of node degree to d-hop neighborhood. Hence, to be able to distinguish between nodes on the basis of a GNN, all we need to do is ensure that different nodes have different d-patterns. This can be achieved in 2 ways: 1). By increasing d, *e.g.*, two nodes may have the same degree and hence the same 1-pattern, but their neighbors may have different degrees, which will lead to different 2-pattern for these two nodes. 2). By assigning node features, *e.g.*, two nodes may have the same degree, but their neighbors may have different node features, leading to a different 1-pattern for them as d-pattern also takes initial node features into account. In addition, Tang et al. [2020] also argue that one way of increasing the representation power of GNNs is by increasing their depth, and it achieves the same by proposing IterGNN that applies the same GNN layer for an adaptive number of iterations depending on the input graph. This is equivalent to tying the weights of different layers as in RRNs, as well as in our models.

- **Learnability:** With respect to learnability, Yehudai et al. [2021] prove the existence of a 'bad' local minima that overfits on train data but fails on test samples that have unseen d-patterns. Our test dataset clearly has unseen d-patterns (*e.g.*, nodes in 16×16 sudoku have different degrees than nodes in 9×9 sudoku), but our models still generalize. We note that Yehudai et al. [2021] only talks about the existence of

some bad local minima, but does not rule out the possibility of the existence of other good local minima, which could generalize well, despite differences in local structure between train and test sets. This goes into the whole learnability argument, and whether we can find such not-so-bad local minimas (which presumably exist since the possibility has not been ruled out). One aspect that possibly comes to our rescue is that, unlike most GNN architectures, our design is recurrent in nature, i.e., parameters are tied across different GNN layers as inspired by Palm et al. [2018]. Parameter tying assumption, possibly helps us in learnability, since the recurrence can be seen as a form of regularization, avoiding overfitting (or getting stuck in bad local minima). Exploring this further is a direction for future work.

In addition to Yehudai et al. [2021], Bevilacqua et al. [2021] deal with varying train/test distributions by proposing a size invariant representation of graphs. Their approach focuses on graph classification tasks, and is limited to creating size invariant representations for the entire graph. The theoretical claims presented in their paper primarily focus on the limitation of standard GNN based formulations for generalizing across sizes for graph classification tasks. On the other hand, we are interested in learning representations for each node in the graph for node classification, and it is not clear how the claims, as well as techniques proposed in the paper, extend to our setting.

5.3 Preliminaries and Problem Definition

A combinatorial puzzle can be thought of as a grounded CSP and to formally define a puzzle from the same domain but a larger value-set, we resort to the notion of '*Lifted CSPs*' that represent an abstraction over multiple ground CSPs of the same type. A lifted CSP does not include a specific set of variables and values; instead, it operates in terms of variable and value *references* that can be instantiated with all ground variables and values in a ground CSP. This makes them amenable to instantiate CSPs or puzzles with varying number of variables as well as values. We define a Lifted CSP $\mathcal{L}_C$ as a three tuple $\langle \mathcal{P}, \mathcal{R}, \mathcal{C} \rangle$. $\mathcal{P}$ is a set of predicates: a predicate $p \in \mathcal{P}$ represents a Boolean function from the set of its arguments, which are variable references. Similarly, $\mathcal{R}$ is a set of relations over value space – a $r \in \mathcal{R}$ reprents a Boolean function over arguments that are value references. A predicate (or a relation) with its arguments is called an atom. $\mathcal{C}$ is a set of lifted constraints, constructed by applying logical operators to atoms – they are interpreted as universally quantified over all instantiations of variable and value references. Finally, Lifted CSP uses a special unary function `Value`, whose argument is a variable reference and evaluates to a value reference. As an example, a lifted CSP for Sudoku may have a

$\mathcal{P}$ ={`Nbr`} for whether two cells are in same row, column or box, $\mathcal{R}$ = {`Neq`}, representing two values are unequal, and a lifted constraint: $\texttt{Nbr}(c_1, c_2) \rightarrow \texttt{Neq}(\texttt{Value}(c_1), \texttt{Value}(c_2))$.

A lifted CSP $\mathcal{L}_C$ yields a ground CSP C, given a set of variables $\mathcal{O}$, and a set of values $\mathcal{V}$, and a complete instantiation of all predicates and relations over this set (*e.g.*, in Sudoku, the number of cells, possible values, and which cells are neighbors and which are not). The ground constraints are constructed by instantiating lifted constraints over all variables and values. A (satisfying) solution, $\mathbf{y}$, of a CSP refers to a complete specification of `Value`: $\mathcal{O} \rightarrow \mathcal{V}$ function, such that all the constraints are satisfied. We are often given a partial (satisfying) solution, $\mathbf{x}$ – an assignment of values to a subset of variables $\tilde{\mathcal{O}} \subseteq \mathcal{O}$ and the goal is to output $\mathbf{y}$, such that $\mathbf{y}$ agrees with $\mathbf{x}$ for the subset $\tilde{\mathcal{O}}$.

Given a ground CSP C, the *Constraint Graph*, $G_C = (N_C, E_C)$, is constructed by having each variable in the CSP represent a node in the graph and introducing an edge between two nodes n_1^C, n_2^C iff the corresponding variables appear together in some constraint. The edges in the constraint graph are typed based on the identity of the lifted constraint from which it comes. Note that there could be multiple edges between nodes n_1^C, n_2^C in G_C, if these nodes appear together in more than one constraint. We embed the knowledge about relations between values in $\mathcal{V}$ in the form of another graph, called *Relation Graph*, $G_R = (N_R, E_R)$, where there is a node for every value in the set $\mathcal{V}$, and there is a (directed) edge between nodes corresponding to v_l, v_l' depending on whether $r(v_l, v_{l'})$ is true or not, for every $r \in \mathcal{R}$. Similar to G_C, this graph can also have multi-edges between two pairs of nodes, if more than one relationship holds between the corresponding values.

Problem Definition: To achieve value-set invariance, our goal is to train a model M_Θ on training data from an unknown ground CSP C (with variables $\mathcal{O}$ and value-set $\mathcal{V}$) obtained from an unknown lifted CSP $\mathcal{L}_C$, and test it on an arbitrary ground CSP C' from the same lifted CSP (with variables $\mathcal{O}'$ and value-set $\mathcal{V}'$), where $|\mathcal{V}| \neq |\mathcal{V}'|$. Formally, we are given training data $\mathcal{D}$ as a set of tuples $\{((\mathbf{x}^{(i)}, G_{C^{(i)}}), \mathbf{y}^{(i)})\}_{i=1}^{M}$, along with a relationship graph G_R encoding relations between values in the value-set $\mathcal{V}$. Here, i^{th} instance denotes a partial and corresponding complete solution for $C^{(i)}$. We note that explicit form of the constraints in $C^{(i)}$ or $\mathcal{L}_C$ are not available, only the graphs are given to the model. Our goal is to learn model M_Θ, such that given graphs $G_{C'}$ and $G_{R'}$, and a partial solution $\mathbf{x}'$ (for CSP C') : $M_\Theta(\mathbf{x}') = \mathbf{y}'$, only if $\mathbf{y}'$ is a corresponding complete solution for $\mathbf{x}'$. Note that in one of our models, we will additionally assume that $\max |\mathcal{V}'|$, denoted as $k_{\max}$, is known to us at training time, which we argue is a benign assumption for most practical applications.

5.4 Models Description

We propose two models for value-set invariance: the *Binarized Model*, and the *Multi-valued Model*. In each case, we assume the training data is provided in the form $\mathcal{D} = (\{(\mathbf{x}^{(i)}, G_{C^{(i)}}), \mathbf{y}^{(i)}\}_{i=1}^{M}, G_R)$ as described in Section 5.3. Let $\mathcal{V}$ and $\mathcal{V}'$ denote the value-sets at train and test time, with cardinality k, k', respectively. For each model, we first present a high level intuition, followed by description of: (a) Construction of Message Passing Graph (b) Message Passing Rules (c) Loss Computation, and finally (d) Prediction on a problem with larger value-set.

5.4.1 Binary Model

Intuition behind our Binarized Model comes directly from the 'sparse encoding' of a discrete CSP into a SAT formula [de Kleer, 1989, Walsh, 2000], in which assignment of a value $v \in \mathcal{V}$ to any variable $\mathbf{x}[j] \in \mathcal{O}$ is encoded by a Boolean variable that represents $\mathbf{x}[j] == v$. Such an encoding converts a single multi-valued variable into multiple Boolean valued variables.[1] We convert a Constraint Graph with nodes representing multi-valued variables (yellow nodes, see fig. 5.1), into a Binary Graph with Boolean nodes (blue nodes). This creates a $|N_C| \times k$ grid of Boolean nodes, with a row representing a variable, a column representing a value and a grid cell (a Boolean node) representing assignment of a particular value to a particular variable. Such a graph can easily represent relationship between the values as well (horizontal red edges), thereby encapsulating the information present in the Relation Graph (fig. 5.1). We use this Binary Graph for message passing.

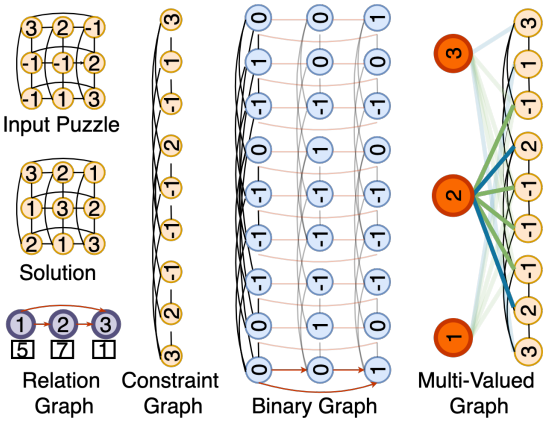


Figure 5.1: An example Futoshiki Puzzle of size 3×3 and the corresponding graphs. A value of −1 indicates an unassigned variable. Black and red edges are Constraint and Relation edges respectively. The digits 5, 7, 1 in square boxes represent a random 3-permutation of $k_{\max}$, used in multi-valued model for initialization of node embeddings.

[1] There is an alternative encoding scheme called 'compact encoding'. We discuss it in section C.1.1 in the appendix.

5.4.1.1 Construction of Message Passing Graph

We denote the *Message Passing Graph (MPG)* by $G = (N, E)$ with the set of nodes N and set of edges E, constructed as follows:

Nodes: For each node $n_j^C \in N_C$ in the Constraint Graph (fig. 5.1, yellow nodes), we construct k binary valued nodes, denoted as $n_{j,1}, n_{j,2} \cdots n_{j,k} \in N$ (blue nodes in Binary Graph).

Edges: We construct two categories of edges in G: *Constraint Edges* and *Relational Edges*. Each edge inherits an *edge type*, either from Constraint Graph, or Relation Graph. We describe them below.

1. *Constraint Edges*: The first category of edges are directly inherited from the edges of the constraint graph G_C (black vertical edges), with k copies created due to binarization. Edge type is same as in the original constraint graph and is denoted by q. Formally, for every edge, ${e^C}_{(j,j')} \in E_C$, where ${e^C}_{(j,j')}.type = q$, we introduce k edges denoted as $e^q_{(jl,j'l)}$, i.e., there is an edge between every pair of nodes, $n_{j,l}$ and $n_{j',l}$, $1 \leq l \leq k$. We denote the set of all constraint edge types as Q

2. *Relational Edges*: The second category of edges encode the information from the Relationship Graph G_R into the MPG, with $|N_C|$ copies of it created, one for each variable. For every edge ${e^R}_{(l,l')} \in E_R$ with edge type r, create an edge $e^r_{(jl,jl')}$ with type r between every pair of binary nodes $n_{j,l}$ and $n_{j,l'}$, $1 \leq j \leq |N_C|$ (*e.g.*, red edges encoding *less-than* relation between value pairs $(1, 2)$, $(2, 3)$ and $(1, 3)$). We denote the set of all relational edge types as R.

5.4.1.2 Recurrent Message Passing

Once MPG has been constructed, we follow recurrent message passing rules, with weights shared across layers, similar to RRNs [Palm et al., 2018] with some differences. For each node $n_{j,l}$ in the graph, we maintain a hidden state $h_t(n_{j,l})$, which is updated at each step t based on the messages received from its neighbors. This hidden state is used to compute the probability of a binary node taking a value of 1. Since we use sparse encoding, only the node with maximum probability amongst the k binary nodes $n_{j,l}$; $1 \leq l \leq k$, corresponding to multi-valued variable $\mathbf{x}[j]$, is assigned a value 1, at the end of message passing. Next, we discuss how the nodes are initialized before message passing starts, followed by the details of message passing and state update function.

Initialization: Irrespective of the size of value-set $\mathcal{V}$ or vertices N_C, there are 3 learnable embeddings ($u[0], u[1]$ and $u[-1]$) for initialization: two for binary values 0 and 1, and one for value -1 representing unassigned nodes. All k nodes corresponding to an unassigned

variable $\mathbf{x}[j]$ are initialized with $u[-1]$, *i.e.*, whenever $\mathbf{x}[j]$ is `NULL` (yellow nodes with -1), $u_0(n_{j,l}) = u[-1], \forall v_l \in \mathcal{V}$, where u_0 represents initial embedding function. On the other hand, if $\mathbf{x}[j]$ is preassigned a value $v_{\widehat{l}}$, then $u_0(n_{j,l}) = u[0], \forall v_l \neq v_{\widehat{l}}$, and $u_0(n_{j,\widehat{l}}) = u[1]$. For example, variable corresponding to the binary nodes in 1^{st} row has a preassigned value of '3', consequently, binary nodes in 1^{st} and 2^{nd} column of the 1^{st} row are initialized with $u[0]$, and binary node in the 3^{rd} column of 1st row, which corresponds to assignment '$\mathbf{x}[1] = 3$', is initialized with $u[1]$. Lastly, the hidden state, $h_0(n_{j,l})$, of each node, $n_{j,l}$, is initialized as a $\mathbf{0}$ vector, $\forall j, \forall v_l$.

Edge Dependent Message Passing: The nodes communicate their current hidden state via the messages sent to their neighbouring nodes across the edges. The message depends not only on the current state of the sender and receiver, but also on the *edge type* across which the message is sent. Specifically, for each edge type, z, there is a separate message passing function, f_z, with $z \in (Q \cup R)$ where Q and R are the set of all constraint edge types and relation edge types respectively. We compute the message for each edge $e^z_{(j_1 l_1, j_2 l_2)} \in E$ as:

$$m_t\left[e^z_{(j_1 l_1, j_2 l_2)}\right] = f_z\left(h_t\left(n_{j_1,l_1}\right), h_t\left(n_{j_2,l_2}\right)\right), \forall e^z_{(j_1 l_1, j_2 l_2)} \in E,\ z \in (Q \cup R)$$

Hidden State Update: For each node, the incoming messages on the edges of the same type are aggregated by taking their weighted average. The weights, a_t, are computed using Bahdanau Attention [Bahdanau et al., 2015] over constraint edges, whereas messages across relation edges are simply averaged:

$$m_{t,z}[n_{j,l_1}] = \sum_{e^z_{(jl_1, j_2 l_2)} \in E} a_t[e^z_{(jl_1, j_2 l_2)}] m_t[e^z_{(jl_1, j_2 l_2)}]\ ,\ \forall z \in (Q \cup R)$$

Finally, all messages, $m_{t,z}[n_{j,l}]\ \ \forall z \in (Q \cup R)$, are concatenated to create the input, $m_t[n_{j,l}]$ for each node, $n_{j,l}$. The hidden state at step t is updated by the following state update function to generate the state $h_{t+1}(n_{j,l})$:

$$h_{t+1}(n_{j,l}) = g\left(h_t(n_{j,l}), m_t[n_{j,l}], u_0(n_{j,l})\right), \forall n_{j,l} \in N$$

See Figure 5.2 for an illustration of edge–dependent message passing and state update at a given step t.

5.4.1.3 Loss Computation

The Binary Cross Entropy (BCE) loss for each node $n_{j,l}$ is computed w.r.t. its target, $\mathbf{y}[j, l]$, which is defined as 1 whenever $\mathbf{y}[j] = l$ and 0 otherwise. At each step $t \in \{1 \dots T\}$, we can compute the probability $Pr(n_{j,l}.v = 1; \Theta)$ of classifying a node $n_{j,l}$ as 1 by passing its hidden state through a learnable scoring function s, *i.e.*, $Pr_t(n_{j,l}.v = 1; \Theta) = \sigma(s(h_t(n_{j,l})))$, where σ is the standard Sigmoid function. Here, $n_{j,l}.v$ denotes the value that node $n_{j,l}$

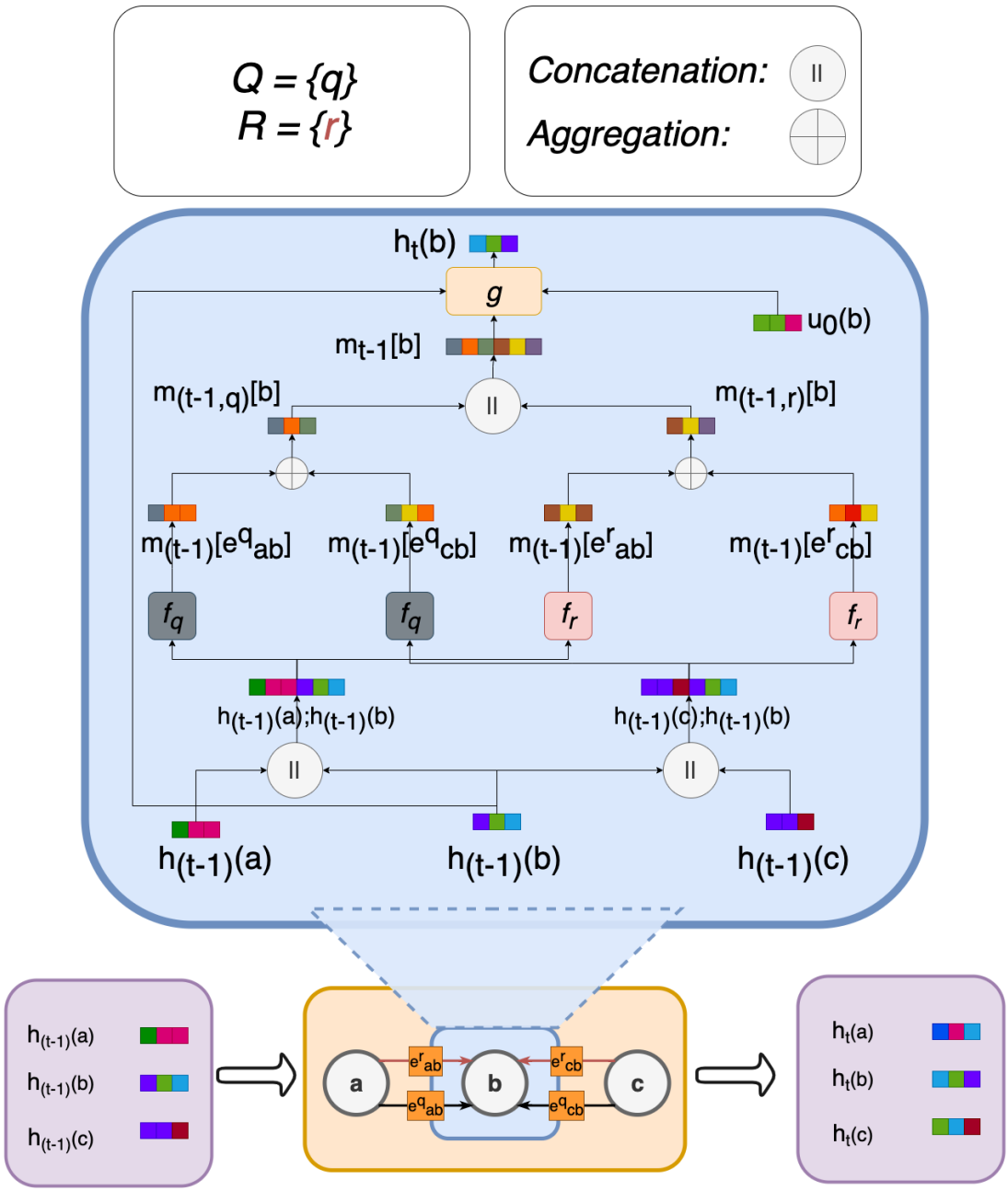


Figure 5.2: Hidden State Update at time-step t: We take a toy graph with 3 nodes, $\{a, b, c\}$, and 2 edge-types, $\{q, r\}$, to illustrate edge-dependent message passing and hidden state update of node b. First, messages along the four edges are calculated as an edge-type dependent function of the sender and receiver hidden state using f_q and f_r. Next, the incoming messages are aggregated by edge-type (*e.g.*, using attention based mechanism or simple averaging), and the outputs are concatenated to obtain the final message, $m_{t-1}[b]$. The hidden state of node b is updated by function g which takes the previous hidden state $h^{t-1}(b)$, the incoming message $m_{t-1}[b]$, and the initial embedding of the node as its input.

can take and belongs to the set $\{0, 1\}$. Loss at step t is the average BCE loss across all the nodes:

$$\frac{1}{|N|} \sum_{n_{j,l} \in N} \mathbf{y}[j, l] \log Pr_t(n_{j,l}.v = 1; \Theta) + (1 - \mathbf{y}[j, l]) \log Pr_t(n_{j,l}.v = 0; \Theta)$$

Like Palm et al. [2018], we back-propagate through the loss at every step $t \in \{1 \dots T\}$ as it helps in learning a convergent message passing algorithm. During training, the objective is to learn the 3 initial embeddings $u[-1], u[0], u[1]$, functions used in message passing and state update, and the scoring function s.

5.4.1.4 Prediction on a problem with larger size of value-set

While testing, let the constraint and relation graph be $G_{C'}$ and $G_{R'}$ with n' and k' nodes respectively. Let $\mathbf{x}'$ be a partial solution, with n' variables $\mathbf{x}'[j]$, each taking a value from value-set $\mathcal{V}'$ of size k'. As described above, we create a graph G' with $n'k'$ nodes, run message passing for T steps, and for each variable $\mathbf{x}'[j]$, compute the k' probabilities, one for each of the k' nodes $n_{j,l} \forall l \in \mathcal{V}'$ corresponding to the variable $\mathbf{x}'[j]$, which is assigned the value corresponding to maximum probability, *i.e.*, $\hat{\mathbf{y}}[j] = \arg\max_{l \in \mathcal{V}'} Pr_T(n_{j,l}.v = 1; \Theta)$.

5.4.2 Multi-valued Model

Multi-valued model differs from the binarized model by avoiding binarization of nodes, and instead explicitly adding *Value Nodes* in the message passing graph, one for each value in the value-set. The message graph consists of two components: (a) A Graph $G = (N, E)$ to represent constraints inherited from the constraint graph $G_C = (N_C, E_C)$ (b) A Graph $\tilde{G} = (\tilde{N}, \tilde{E})$ to represent relations inherited from the relationship graph $G_R = (N_R, E_R)$. We refer to G as *Constraint Message Passing Graph* (**CMPG**), and $\tilde{G}$ as *Relationship Message Passing Graph* (**RMPG**). Message passing on RMPG first generates desired number of embeddings (upto $k_{\max}$), one for each of the value nodes. This is followed by message passing on CMPG which uses the embeddings of the value nodes generated by RMPG and computes embeddings for each variable node. Finally, the variable nodes are classified based on the similarity of their embedding with the embeddings of the value nodes computed by RMPG. Learning to generate upto $k_{\max}$ embeddings from training samples with only $k (< k_{\max})$ values in the value-set is the main technical challenge that we address in this model. Next, we first describe the construction of both the message passing graphs: CMPG and RMPG, followed by a discussion on how we achieve value-set invariance in our multi-valued model. We then provide details of message passing, loss computation and prediction on problems with larger value-set.

5.4.2.1 Construction of CMPG

Nodes: For each node $n_j^C \in N_C$ in the constraint graph, we construct a k-valued node, denoted as $n_j \in N$. Total number of such nodes constructed is $|N_C|$. We refer to these as *Variable Nodes* (yellow nodes in Multi-Valued Graph in fig. 5.1). Additionally, for each value $v_l \in \mathcal{V}$ in the value-set, we create a node, denoted as $n_l^v \in N$. Total number of such nodes constructed is $|\mathcal{V}|$. We refer to these as *Value Nodes* (orange nodes).
Edges: For every edge, ${e^C}_{(j,j')} \in E_C$, where ${e^C}_{(j,j')}.type = q$, we introduce an edge denoted as $e^q_{(j,j')}$ with type q. These edges are directly inherited from the constraint graph. Edge type is same as in the original constraint graph and is denoted by q. We refer to these as *Constraint Edges* (black edges), and we denote the set of all constraint edge types as Q. Additionally, to indicate the pre-assignment of values to the variables in $\mathbf{x}$, we introduce new edges connecting value nodes to appropriate variable nodes. Whenever $\mathbf{x}[j] = v_l$, add an edge, $e^a_{(j,l)}$ between variable node n_j and value node n_l^v (blue edges). If $\mathbf{x}[j]$ is `NULL`, *i.e.*, unassigned, then add k edges, $e^{\bar{a}}_{(j,l)}, \forall v_l \in \mathcal{V}$, connecting the variable node n_j with all k value nodes n_l^v (*e.g.*, green edges connecting orange value node '2' to all '-1' variable nodes). We refer to them as *Assignment Edges*, and denote the set of their types by A.

5.4.2.2 Construction of RMPG

Nodes: For each value $v_l \in \mathcal{V}$, create a node denoted as $\tilde{n}_l^v \in \tilde{N}$ (purple nodes in Relation Graph in fig. 5.1). Total number of such nodes constructed is $|\mathcal{V}|$. We refer to these as *Value Nodes*.
Edges: For every pair of value nodes, $\tilde{n}_l^v$ and $\tilde{n}_{l'}^v$, introduce an edge $\tilde{e}^r_{(l,l')}$ with type r if $r(v_l, v_{l'})$ holds based on the relationship graph G_R, i.e., ${e^R}_{(l,l')} \in E_R$ with edge label r (red edges). The edge type is the same as relation r, and we denote the set of all edge types in RMPG as R. These edges are defined for relations that exist between values in the value-set.

5.4.2.3 Achieving Value-set Invariance

A key question arises here: why do we need to construct a separate RMPG ($\tilde{G}$)? Why not embed relevant edges in CMPG (G), as done for the binarized model? The answer lies in realizing that we represent each value in the value-set explicitly in the multi-valued model, unlike the binarized model. Hence, our model needs to learn representation for each of them in the form of value node embeddings. Further, to generalize we need to learn as many embeddings as there are values in the largest test value-set, i.e., $k_{\max} = \max |\mathcal{V}'|$. We achieve this by randomly sub-selecting a k-sized set from $\{1 \dots k_{\max}\}$ and permuting the chosen subset for each training example in a given mini-batch, and then computing

the 'relationship-aware' embeddings from this permuted subset through message passing in RMPG. The 'relationship-aware' embeddings are then used to initialize the value nodes (orange nodes) during message passing in CMPG. For instance, if the permutation obtained is $\{w_1, \cdots, w_l, \cdots, w_k\}$, where $\forall l, 1 \leq w_l \leq k_{\max}$, then embedding for the value node $\tilde{n}_l^v$ in $\tilde{G}$ is initialized by w_l^{th} learnable embedding (*e.g.*, purple nodes for values '1', '2', and '3' are initialized by the 5th, 7th, and 1st learnable embedding, respectively). After message passing on $\tilde{G}$, the 'relationship-aware' embedding of $\tilde{n}_l^v$ (purple node) is used to initialize the embedding for value node n_l^v (orange node) in G. This elegant process is able to train all the $k_{\max}$ embeddings by simply using the training data corresponding to $\mathcal{V}$, and the corresponding relationship information. Since these relationship aware embeddings need to be pre-computed before they can be passed to the downstream constraint processing, we construct two different message passing graphs, one for computing relationship-aware embeddings and one for constraint handling.

5.4.2.4 Recurrent Message Passing on RMPG:

Rules of message passing and hidden state updates at every step t are similar to RRN in Palm et al. [2018]. After updating the hidden states for total $\tilde{T}$ steps, the final embeddings, $\tilde{h}_{\tilde{T}}(\tilde{n}_l^v)\ \forall v_l \in \mathcal{V}$, are used as 'relationship-aware' embeddings for initializing the input features (embeddings) of the nodes in CMPG G. Below discuss the details of the initialization nodes and message passing update in RMPG.

Initialization: There are a total of $k_{\max}$ learnable embeddings, $\tilde{u}[l'], 1 \leq l' \leq k_{\max}$, out of which any k are randomly chosen for initializing the nodes in RMPG. *e.g.*, $\tilde{u}[5], \tilde{u}[7], \tilde{u}[1]$ are chosen to initialize the purple value nodes '1','2', and '3' in Relation Graph in fig. 5.1. Formally, for each input $\mathbf{x}$, select a k-permutation, $\mathcal{P}_{\mathbf{x}}$, of $k_{\max}$ (*e.g.*, $(5, 7, 1)$ as shown in square boxes in fig. 5.1). Initialize the embedding of $\tilde{n}_l^v$ in $\tilde{G}$ with $\tilde{u}[\mathcal{P}_{\mathbf{x}}[l]],\ \forall l \in \{1 \ldots k\}$. Initialize the hidden state, $\tilde{h}_0(\tilde{n}_l^v),\ \forall \tilde{n}_l^v \in \tilde{N}$ with a $\mathbf{0}$ vector.

Message Passing Update: At step t, update the hidden state, $\tilde{h}_t(\tilde{n}_l^v)$, of each of the value nodes in $\tilde{G}$, by the concatenation, $m_t[\tilde{n}_l^v]$, of average messages, $m_t^r[\tilde{n}_l^v]$, received across edges of type $r \in R$:

$$\tilde{h}_{t+1}(\tilde{n}_l^v) = \tilde{g}(\tilde{h}_t(\tilde{n}_l^v), m_t[\tilde{n}_l^v], \tilde{u}[\mathcal{P}_{\mathbf{x}}[l]])$$

Here $\tilde{g}$ is the hidden state update function. Like [Palm et al., 2018], it always takes the initial embedding, $\tilde{u}[\mathcal{P}_{\mathbf{x}}[l]]$, of the value node $\tilde{n}_l^v$ as one of the inputs. Notice that the message, $m_t^r[\tilde{n}_l^v]$, is the average of the messages, $f_r(\tilde{h}_t(\tilde{n}_l^v), \tilde{h}_t(\tilde{n}_{l'}^v))\ \forall\ \tilde{e}_{(l,l')}^r \in \tilde{E}$, where f_r is the message passing function for edge type $r \in R$. The hidden states are updated for $\tilde{T}$ steps and the final embeddings, $\tilde{h}_{\tilde{T}}(\tilde{n}_l^v)\ \forall v_l \in \mathcal{V}$, are used as 'relationship-aware' embeddings for initializing the input features (embeddings) of both variable nodes, n_j,

and value nodes, n_l^v in G (orange and yellow nodes respectively in Multi-Valued Graph in fig. 5.1).

5.4.2.5 Recurrent Message Passing on CMPG:

Message passing on CMPG updates the hidden state, $h_t(n_j)$, of each variable node n_j for a total of T ($t \leq T$) steps using the messages received from its neighbors. Below we briefly describe the initialization of node embeddings and the message passing update function.
Initialization: We initialize the embedding of value nodes (orange nodes), n_l^v in G, using the final 'relationship-aware' embeddings, $\tilde{h}_{\tilde{T}}(\tilde{n}_l^v)$, of $\tilde{n}_l^v$ (purple nodes) in $\tilde{G}$. The variable nodes that are preassigned a value (non-zero yellow nodes) in $\mathbf{x}$, are initialized by the embedding of the corresponding value node, *i.e.*, if $\mathbf{x}[j] = l$, then n_j is initialized with the 'relationship-aware' embedding, $\tilde{h}_{\tilde{T}}(\tilde{n}_l^v)$, of $\tilde{n}_l^v$. The embedding of nodes corresponding to the unassigned variables ('-1' yellow nodes) are initialized by the average, $(1/k)\sum_{v_l \in \mathcal{V}} \tilde{h}_{\tilde{T}}(\tilde{n}_l^v)$, of all 'relationship-aware' embeddings. Initialize hidden state $h_0(n_j)$ of each variable node n_j with a $\mathbf{0}$ vector.
Message Passing Update: At step t, similar to binarized model, each variable node receives messages from its neighbors, that are aggregated based on the edge type. For each node, the aggregated messages, $m_{t,z}[n_j]$, for different edge types, $z \in (Q \cup A)$, are stacked to create, $m_t[n_j]$, which updates the hidden state as:

$$h_{t+1}(n_j) = g\left(h_t(n_j), m_t[n_j], u_0(n_j)\right), \forall n_j \in N$$

5.4.2.6 Loss Computation

For each variable represented by node n_j, the ground truth value $\mathbf{y}[j]$ acts as the target for computing standard Cross Entropy Loss. The probabilities over $\mathcal{V}$ are computed as follows:

At step t, a scoring function, s, computes a score, $s(h_t(n_j), h_t(n_l^v))$, for assigning a value $v_l \in \mathcal{V}$ to a variable n_j based on the hidden state of corresponding value and variable nodes. For each variable node, a $Softmax$ converts these scores into probabilities over the values $v_l \in \mathcal{V}$, *i.e.*, $Pr(n_j.v = v_l) = Softmax(s(h_t(n_j), h_t(n_l^v))),\ \forall v_l \in \mathcal{V}$, where, $n_j.v \in \mathcal{V}$ denotes the value that node n_j can take. Loss at step t is nothing but the average over variable nodes:

$$L_t = -\frac{1}{|N|}\sum_{n_j \in N} \log Pr(n_j.v = \mathbf{y}[j])$$

To ensure that the multi-valued model learns different embeddings for each value in the value-set, we add an auxiliary loss term, corresponding to the total pairwise dot product (similarity) of any two embeddings, before and after message passing in $\tilde{G}$. We call it

Orthogonality Loss. Its weight, α, is a hyper-parameter.

5.4.2.7 Prediction on a problem with larger size of value-set

For a puzzle with larger value-set, $\mathcal{V}'$, a bigger RMPG is created, whose k' nodes are initialized with the (learnt) first k' embeddings. Unlike training, we always choose first k' embeddings to avoid randomness during testing. Prediction is made using the probabilities at the last step T, *i.e.*, $\hat{\mathbf{y}}[j] = \arg\max_{v_l \in \mathcal{V}'} Pr(n_j.v = v_l)$.

5.4.3 Relative Comparison

In the binarized model, the constructed graph G has $k|N_C|$ nodes and at least $k|E_C| + |N_C|k(k-1)/2$ edges due to binarization. This increases the graph size by a factor of at least k. As a result, we soon hit the memory limits of a GPU while training the binarized model with bigger problems. The model also needs significantly more inference time due to its bigger size. On the other hand, multi-valued model, while being compact in terms of its representation, needs to learn additional embeddings, for a speculative size of value-set during testing. This poses additional requirement on the model both in terms of representation, and learning, possibly affecting the quality of generalization. While this is a simple analytical understanding of the possible merits of the two models, we examine experimentally the impact of these issues on real datasets.

Table 5.1: Dataset details

Task	Train				Test			
	k	#(Vars.)	Mask (%)	#(Missing Vars.)	k'	#(Vars.)	Mask (%)	#(Missing Vars.)
Futoshiki	6	36	28-70	10-25	{6,7,8,9,10,11,12}	36-144	28-70	10-93
GCP	4	40-100	28-70	12-70	{4,5,6,7}	40-150	28-70	12-105
Sudoku	9	81	58-79	47-64	{9,10,12,15,16}	81-256	30-68	47-148
Sudoku finetune	24	576	30-70	173-403	{15,16,24,25}	225-625	30-70	68-314

5.5 Experiments

The goal of our experiments is to evaluate the effectiveness of our two proposed methods for achieving value-set invariance. We compare our models with a generic neural constraint learner, NLM [Dong et al., 2019]. [2] We experiment on datasets generated from Lifted CSPs of three different puzzles, *viz.*, Sudoku, Futoshiki, and Graph Coloring. We train each model on data generated from a fixed value-set, and test on instances generated from larger value-sets. In addition, we fine-tune our multi-valued model for Sudoku on 8000 puzzles of size 24 and test it on puzzles of different board sizes. Table 5.1 contains the details of both train and test data for the different experiments. Below we describe the three tasks and their datasets in detail.

5.5.1 Task Description and Datasets

5.5.1.1 Futoshiki

Task: As described in the previous chapter, this is a number puzzle in which we have to place numbers $\{1 \dots k\}$ on a $k \times k$ grid, such that no two cells in a row or column contain the same number. In addition, there may be an ordering constraint between two cells, which needs to be honored in the final solution. The input has some of the grid cells already filled with a number and the task is to complete the grid, respecting the additional ordering constraint where ever it exists.

Dataset: We train our model on $12,300$ puzzles of 6 x 6 board size, with the percentage of missing cells varying uniformly between $28 - 70\%$. We test our models on puzzles with board size ranging between 6×6 to 12×12, with the same percentage of missing cells. See table 5.2 for exact details. We note that it becomes increasingly difficult to find puzzles with unique solution as the board size increases. Therefore, we are forced to reduce the maximum percentage of masked (unassigned) cells with increasing board size. The number of ordering constraints is twice the board size.

[2] Our aim is not to directly compete with SOTA SAT solvers, which are much more scalable than neural methods. Refer to appendix for a discussion on comparison with them as well as neural SAT solvers.

Table 5.2: Test Data statistics for all three tasks

k	#Puzzles	#(Variables)	#(Missing Variables)	Mask (%)
		Futoshiki		
6	4100	36	10-25	28-70
7	4091	49	14-34	29-70
8	3578	64	19-44	30-70
9	3044	81	24-56	30-70
10	2545	100	30-66	30-66
11	2203	121	36-82	30-68
12	1882	144	43-93	30-65
		GCP		
4	9102	40-150	12-105	28-70
5	9102	40-150	12-105	28-70
6	6642	40-120	12-84	28-70
7	3362	40-80	12-56	28-70
		Sudoku		
9	18000	81	47-64	58-79
10	2317	100	30-62	30-62
12	1983	144	43-84	30-58
15	1807	225	67-128	30-57
16	1748	256	76-148	30-58
24	1000	576	172-289	30-50
25	1000	625	187-314	30-50

Data Generation: To generate data, we first randomly generate a completely filled $k \times k$ board and then randomly mask $m\%$ of the cells. We search for its all possible completions using backtracking to ensure that it has only one solution. Finally, we insert ordering constraints between $2k$ randomly picked pairs of adjacent cells. The entire process is repeated by varying k and m to generate both the test and train dataset.

5.5.1.2 Graph Coloring (GCP)

Table 5.3: Range for p for given k and n for GCP data generation

k \ n	**40-55**	**56-70**	**71-80**	**81-100**	**101-130**	**131-150**
4	(0.1, 0.2)	(0.05, 0.1)	(0.05, 0.1)	(0.05, 0.1)	(0.02, 0.05)	(0.02, 0.05)
5	(0.2, 0.25)	(0.1, 0.2)	(0.1, 0.2)	(0.075, 0.12)	(0.075, 0.1)	(0.05, 0.075)
6	(0.2, 0.25)	(0.15, 0.25)	(0.17, 0.2)	(0.15, 0.18)	(0.12, 0.16)	-
7	(0.325, 0.375)	(0.275, 0.325)	(0.22, 0.3)	-	-	-

Task: In this task we are given a partially colored graph along with the number of colors

k, and the objective is to color rest of the nodes using k colors such that no two adjacent nodes have the same color.

Dataset: We train our model on randomly generated 4−colorable graphs, and test on k'−colorable graphs, with $k' \in \{4, 5, 6, 7\}$. The training data has around 25 thousand 4-colorable graphs with graph order varying uniformly between $40 - 120$, and the percentage of unassigned (masked) variables varying uniformly between $28 - 70\%$ depending on the graph order. The exact details of the testing data for different chromatic number (value-set size) are provided in table 5.2.

Data Generation: To create non-trivial problems for the dataset, we always attempt to color graphs with the smallest possible number of colors, *i.e.*, the chromatic number of the graph. We follow Erdős–Rényi (ER) model to generate random graphs. It takes number of nodes, n, and an edge probability, p, as input, and adds an edge independent of the other edges with probability p. We note that to sample a graph with n nodes and a given chromatic number k, we need to carefully adjust the range from which edge probability p is sampled. The exact range from which p is sampled uniformly for each chromatic number k and a range of nodes n is given in Table 5.3. We use a CSP solver [Perron and Furnon, 2019] to determine the chromatic number of a given graph, which becomes a bottleneck while generating graphs with higher chromatic number. As a result, we are not able to generate graphs with more than 80 nodes for chromatic number 7, in a reasonable amount of time.

5.5.1.3 Sudoku

Dataset: We randomly select $10,000$ training queries from the 9×9 dataset introduced in Palm et al. [2018]. It has a uniform distribution of number of masked cells, varying between 17 and 33. Our test set has $k' \times k'$ puzzles, with $k' \in \{9, 10, 12, 15, 16\}$. For standard 9×9 board, we use the same test data as used in RRN [Palm et al., 2018]. The test data for the board-sizes between 10 and 16 has the distribution of missing cells varying between $30 - 68\%$ depending on the board size. See table 5.2 for the exact details.

Data Generation: For standard 9×9 board, we use the same test data as used in RRN [Palm et al., 2018] [3]. To generate test data for the board-sizes between 10 and 16, we follow the methodology similar to Futoshiki, except that we use the GSS library [Pieters, 2019], instead of backtracking, to check the solution validity and uniqueness. For the experiment where we fine-tune our models on 24×24 puzzles, both the train and test data for board size 24 and 25 are generated following the methodology similar to Futoshiki. In this setup, we were not able to verify the uniqueness of solution through GSS library as it

[3] Available at: *https://data.dgl.ai/dataset/sudoku-hard.zip*

doesn't scale to such large sizes.

5.5.1.4 Solution Multiplicity

An input query in GCP may have more than one solution, out of which only one is given at train time. But the network may discover a new valid solution, and computing loss of the discovered solution *w.r.t.* the given solution may unnecessarily penalize the model. To avoid this, we algorithmically verify if a prediction is a valid coloring or not, and compute loss *w.r.t.* to this discovered solution, instead of the given one. This is equivalent to solving a One-of-Many Learning problem as defined in chapter 4 with all possible colorings given at training time. The same phenomenon of solution multiplicity exists for Sudoku puzzles of size 24 and 25, as verifying the uniqueness of puzzles on such large board-size became computationally infeasible.

5.5.2 Experimental Setup & Baselines

In both our models, nodes are initialized with learnable 96 dimensional embeddings. In multi-valued model, $k_{\max} = 12, 7$, and 16 embeddings are learnt for Futoshiki, GCP, and Sudoku respectively. Message passing on G in binarized model runs for 32 steps. Message passing on RMPG, $\tilde{G}$ and CMPG, G in the multi-valued model runs for $\tilde{T} = 1$ and $T = 32$ steps respectively. The message passing functions in both the models are 3 layer MLPs, similar to those in RRN, with a difference that there is a separate function for each edge type. In both the models, a layer normalized LSTM cell with hidden dimension 96 acts as state update functions. All models are trained on K40 GPU nodes with 12GB memory. We take simple average of model weights stored at multiple points [Izmailov et al., 2018]. All checkpoints obtained after flattening of the learning curve are selected for computing average. See section C.1.2 in appendix for details.

5.5.2.1 Baselines

For Futoshiki, we train two versions of NLM by varying *depth*: the number of Logic Machines that are stacked on top of each other. As in chapter 4, we train one 30 layer deep NLM model with residual connections for Futoshiki, but unlike them, we assume access to constraint graph, which we provide as a binary predicate input to the model. NLM with 30 depth could not fit puzzles with board-size greater than 10 within 12GB memory of K40 GPU. Hence, we train another version by reducing the depth to 15. For GCP, we train a model with depth 24. For Sudoku, on increasing depth beyond 14, we could not fit even one 9×9 train sample within GPU memory. Note that the maximum depth

Table 5.4: Futoshiki: Mean (Std. dev.) of Board and Pointwise accuracy on different board sizes. MV and BIN correspond to Multi-valued Model and Binarized Model, respectively.

	6	7	8	9	10	11	12
Board Accuracy							
NLM15	73.37 (1.34)	56.98 (1.47)	48.71 (1.96)	44.16 (1.72)	37.54 (2.74)	32.50 (2.84)	-
NLM30	85.72 (0.39)	69.61 (0.57)	63.52 (1.20)	60.73 (1.29)	55.94 (0.85)	-	-
MV	99.62 (0.18)	90.18 (2.38)	71.58 (4.66)	54.85 (6.89)	38.51 (5.62)	24.18 (4.49)	11.97 (5.54)
BIN	**99.86 (0.01)**	**97.92 (1.27)**	**93.39 (4.08)**	**89.39 (6.03)**	**83.48 (10.7)**	**76.14 (15.83)**	**68.15 (22.08)**
Pointwise Accuracy							
NLM15	96.72 (0.16)	93.9 (0.26)	93.43 (0.26)	93.86 (0.28)	94.07 (0.29)	94.29 (0.31)	-
NLM30	97.88 (0.05)	95.32 (0.10)	95.09 (0.14)	95.48 (0.08)	95.68 (0.03)	-	-
MV	99.91 (0.03)	98.84 (0.24)	97.09 (0.46)	96.07 (0.60)	95.17 (0.53)	94.52 (0.41)	93.99 (0.60)
BIN	**99.97 (0.00)**	**99.63 (0.13)**	**99.02 (0.37)**	**98.60 (0.47)**	**98.23 (0.68)**	**97.85 (0.98)**	**97.66 (1.31)**

Table 5.5: GCP: Mean (Std. dev.) of coloring and pointwise accuracy on graphs with different chromatic number.

	4	5	6	7
Board Accuracy				
NLM24	81.34 (5.93)	70.78 (7.45)	71.25 (8.35)	73.20 (7.58)
MV	97.80 (0.03)	**97.72 (0.37)**	94.03 (2.54)	72.21 (11.17)
BIN	**99.09 (0.07)**	96.69 (2.61)	**95.7 (4.04)**	**94.35 (4.82)**
Pointwise Accuracy				
NLM24	99.47 (0.13)	98.58 (0.34)	97.95 (0.54)	97.26 (0.68)
MV	**99.96 (0.00)**	**99.89 (0.00)**	99.50 (0.23)	96.22 (1.55)
BIN	**99.96 (0.01)**	99.85 (0.03)	**99.76 (0.08)**	**99.48 (0.16)**

chosen for the graph experiments reported in [Dong et al., 2019] is 8. This is because they work with much smaller graphs (up to maximum 50 nodes), whereas smallest graph in Futoshiki has $6^3 = 216$ binary nodes, warranting creation of much deeper models.

5.5.2.2 Evaluation Metrics

For each experiment, we report the mean and standard deviation over three runs by varying random seed. We report two metrics: *board accuracy* and *point-wise accuracy* for all our experiments. In the former, we consider output of the model as correct only if it satisfies the underlying CSP, whereas in the later, we give partial credit even for assigning some of the variables correctly, formally defined below.

Definition 2. *Pointwise Accuracy: Let $Y_{\mathbf{x}}$ be the set of possible solutions for an input $\mathbf{x}$ with k variables, and let $\hat{\mathbf{y}}$ be the model's prediction. Pointwise accuracy of the prediction $\hat{\mathbf{y}}$ with respect to the solution $\mathbf{y} \in Y_{\mathbf{x}}$, denoted as, $PointAcc(\mathbf{y}, \hat{\mathbf{y}})$, is defined to be the fraction of variables that match between the $\mathbf{y}$ and $\hat{\mathbf{y}}$: $PointAcc(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}\{\mathbf{y}[i] == \hat{\mathbf{y}}[i]\}$, where $\mathbb{1}\{.\}$ is the indicator function.*

Given above, we define pointwise accuracy for a prediction $\hat{\mathbf{y}}$ of an input $\mathbf{x}$ with respect to a solution set $Y_{\mathbf{x}}$ to be the maximum among the pointwise accuracy with respect to each

Table 5.6: Sudoku: Mean (Std. dev.) of board and pointwise accuracy on different board-sizes. Both models trained on 9×9 puzzles

	Board Accuracy				
	9	**10**	**12**	**15**	**16**
MV	92.78 (0.08)	99.65 (0.15)	88.30 (6.08)	29.33 (13.71)	19.70 (14.03)
BIN	**99.13 (0.14)**	**99.91 (0.04)**	**99.63 (0.10)**	**63.05 (45.71)**	**27.31 (23.81)**
	Pointwise Accuracy				
MV	98.52 (0.05)	99.96 (0.02)	99.43 (0.26)	**97.03 (0.71)**	**96.30 (0.90)**
BIN	**99.87 (0.02)**	**99.99 (0.00)**	**99.96 (0.01)**	95.55 (6.60)	88.39 (14.25)

of the solutions in the set $Y_{\mathbf{x}}$. *Mathematically,* $PointAcc(Y_{\mathbf{x}}, \hat{\mathbf{y}}) = max_{\mathbf{y} \in Y_{\mathbf{x}}} PointAcc(\mathbf{y}, \hat{\mathbf{y}})$.

For Sudoku and Futoshiki, since there is a unique solution, we can easily compute pointwise accuracy as the target set $Y_{\mathbf{x}}$ is singleton. For the GCP task, whenever the model returns a valid coloring, pointwise accuracy is 1, otherwise, in the absence of access to complete $Y_{\mathbf{x}}$, we report a lower bound by performing a local search, using Google's OR-Tools [4], for a valid coloring closest to the model prediction. Same is true for sudoku puzzles on 24 and 25 board size.

Why care about point-wise accuracy? In our settings, the generalization problem can be hard, especially when there is a large difference between the sizes of the value-sets for the train and test domains. Given that we are defining a novel task, and it is important to measure progress even when problems are hard, we compare the two models using a simpler metric (pointwise accuracy) as well, in addition to board accuracy. This additional metric can help us detect progress, and also compare the relative performance of underlying models.

5.5.3 Results and Discussion

We report the accuracies over different sizes of value-set for Futoshiki, GCP and Sudoku in Table 5.4, 5.5, and 5.6, respectively. We first observe that NLM fails to train on Sudoku, and its performance is worse than one or both of our models for all experimental settings in Futoshiki and GCP. As expected, in Futoshiki, NLM

Table 5.7: Sudoku: Mean (Std. dev.) of board and pointwise accuracy of models fine-tuned on 24 board-size

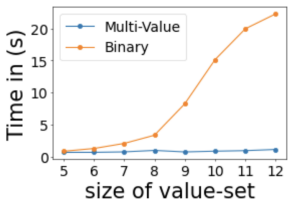
	Board Accuracy			
	15	**16**	**24**	**25**
MV	**91.03 (3.25)**	**90.39 (3.49)**	**54.57 (21.25)**	**43.77 (14.42)**
BIN	63.05 (45.71)	27.31 (23.81)	0.0 (0.0)	0.0 (0.0)
	Pointwise Accuracy			
MV	**99.43 (0.16)**	**99.46 (0.15)**	**99.30 (0.12)**	**99.10 (0.09)**
BIN	95.55 (6.60)	88.39 (14.25)	7.85 (0.63)	7.44 (0.43)

[4] https://developers.google.com/optimization

model with depth 30 fails to run on board sizes 11 and 12 and depth 15 model fails to run on size 12. Note that both NLM and our binarized model work by binarizing the underlying puzzle, but we observe that binarized model shows significantly better generalization across value-sets. We note that NLM performs decently well for GCP even for the test graphs with chromatic number $k' = 7$. We attribute this to the fact that in our test data for $k' = 7$, graphs are relatively small, with max 80 graph nodes, resulting in total 560 binary objects in NLM, which is similar to the max 400 binary objects that it trains over (k=4, max 100 nodes).

Comparison between binarized model and multi-valued model: We first observe that both our models achieve similar performance on the value-set over which they are trained. We observe that the standard deviation of the *board accuracy* increases significantly as the size of value-set increases, whereas the *pointwise accuracy* is relatively stable. This is due to the high sensitivity of the board accuracy to pointwise accuracy: even if a single variable is incorrectly assigned in a puzzle, its contribution towards board accuracy goes to 0, whereas it still contributes positively towards pointwise accuracy. When trained on small sizes, binarized model shows better generalization. But as the problem size increases, the computational graph for binarized model fails to fit in the available GPU memory and thus its performance degrades. On the other hand, multi-valued model being memory efficient, scales much better. To demonstrate this, Table 5.7 reports the performance of multi-valued model further finetuned on sudoku puzzles of board-size 24, and tested on board-sizes varying between 15 and 25. We couldn't finetune the binarized model as its computational graph doesn't fit in the GPU. The binarized model trained on puzzles of board-size 9 gives 0.0 board accuracy on size 24 and 25. The performance of multi-valued model is better than binarized model not only on board-size 25, but also on board-sizes smaller than 24. This also demonstrates that the poor performance of the same multi-valued model trained on smaller board-size is not due to any lack of *representation power*, but due to difficulty in learning additional embeddings: when training k' embeddings from puzzles of board-size k, multi-valued model never gets to see all k' value embeddings together. Moreover, the different combinations of k out of k' embeddings increase exponentially with $(k' - k)$, making it further difficult to train. To validate this, we train a multi-valued model with only 7 learnable embeddings for Futoshiki and observe that the board accuracy on 7 board-size increases to 97.82% (at par with binarized model) from 90.18% which is achieved when trained with 12 embeddings.

Computational complexity: In fig. 5.3, for the two models, we compare the average inference time and the GPU memory occupied by a batch of 32 Futoshiki puzzles over value-sets of



(a) Runtime

varying sizes. As expected, the multi-valued model is much more efficient, both in terms of time and memory.

5.6 Conclusion

We have looked at the novel problem of value-set invariance in combinatorial puzzles, formally defined using lifted CSPs and proposed two different neural solutions extending RRNs. Our experiments demonstrate the superior performance of our models compared to an existing neural baseline. We discuss the relative strengths and weaknesses of our proposed models.

In the last two chapters, we discussed neural solvers for combinatorial problems that are instances of a (lifted) CSP with unknown constraints. These models implicitly represent the rules or constraints in their weights. One issue with these models is the lack of interpretability of the underlying rules. In the next part of the thesis, we will explore methods for learning constraints or rules with explicit representation, *e.g.*, as linear inequalities of an Integer Linear Program (ILP), or as horn clauses. This also allows us to use state-of-the-art commercial solvers, such as Gurobi [Gurobi Optimization, LLC, 2022], for symbolic reasoning. One advantage of integrating solvers in the model is that once correct rules are learnt, such models will generalize better on unseen examples, irrespective of their difficulty. We discuss this in chapter 6.

In addition to solving CSPs with unknown constraints, explicit representation of constraints in the form of horn clauses may also act as plausible explanations for a particular prediction of a neural model. We explore this in chapter 7 for the task of Knowledge Base Completion (KBC) using Tensor Factorization (TF) based models.

Part IV

Explicit and Unknown Constraints

Chapter 6

Scalable Learning of Constraints in Neural ILP Architectures

In the last two chapters we focused on neural models for combinatorial problems and addressed a couple of issues with them: solution multiplicity and value-set (or output space) invariance. Main characteristic of the neural models for reasoning (such as RRN [Palm et al., 2018] and NLM [Dong et al., 2019]) is that the representation of constraints is implicit in their weights, making them a black-box reasoner. One can circumvent this by learning constraints explicitly in the language of a symbolic reasoner, *e.g.* linear constraints for Integer Linear Programs (ILP). The benefit of learning explicit constraint is two–fold: in addition to making the system more interpretable, it gives an opportunity to exploit the symbolic solvers. Moreover, once the correct constraints are learned, symbolic reasoners would generalize much better than neural reasoners.

In addition to creating purely neural models for symbolic reasoning tasks, there is a recent focus on designing architectures that have an ILP layer within a neural model (referred to as *Neural ILP* here). Such architectures are suitable for pure reasoning tasks that require data-driven constraint learning or for tasks requiring both perception (neural) and reasoning (ILP). A recent SOTA approach for end-to-end training of Neural ILP explicitly defines gradients through the ILP black box [Paulus et al., 2021] – this trains extremely slowly, owing to a call to the underlying ILP solver for every training data point in a minibatch. In response, we present an alternative training strategy that is *solver-free*, *i.e.*, does not call the ILP solver at all at training time. Neural ILP has a set of trainable hyperplanes (for cost and constraints in ILP), together representing a polyhedron. Our key idea is that the training loss should impose that the final polyhedron separates the positives (all constraints satisfied) from the negatives (at least one violated constraint or a suboptimal cost value), via a soft-margin formulation. While positive example(s) are

provided as part of the training data, we devise novel techniques for generating negative samples. Our solution is flexible enough to handle equality as well as inequality constraints. Experiments on several problems, both perceptual as well as symbolic, which require learning the constraints of an ILP, show that our approach has superior performance and scales much better compared to purely neural baselines and other state-of-the-art models that require solver-based training. In particular, we are able to obtain excellent performance in 9×9 Visual Sudoku, to which the other Neural ILP solver is not able to scale.

6.1 Introduction

In the last couple of chapters, we discussed various neural models for solving combinatorial optimization problems that require complex reasoning over discrete symbols. Many of these problems can be expressed in the form an underlying Integer Linear Program (ILP). Two different kinds of problem (input) settings have been considered in the literature: (a) purely symbolic and (b) combination of perceptual and symbolic. Solving an n-Queens problem given the partial assignment of queens on the board as input would be an example of the former, and solving a sudoku puzzle given the image of a partially filled board as input would be an example of latter. While the first setting corresponds to a pure reasoning task, second involves a combination of perception and reasoning tasks which need to be solved in a joint fashion. Existing literature proposes various approaches to handle one or both these settings. One line of work that we discussed in part III, proposes purely neural models [Palm et al., 2018, Nandwani et al., 2021, Dong et al., 2019] for solving these tasks representing the underlying constraints and costs implicitly. While standard CNNs are used to solve the perceptual task, neural models such as Graph Neural Networks take care of the reasoning component. In an alternate view, one may want to solve these tasks by explicitly learning the constraints and cost of the underlying ILP. While the perceptual reasoning would still be handled by using modules such as CNN, reasoning is taken care of by having an explicit representation in the form of an ILP layer representing constraints and costs. Such an approach would have the potential advantage of being more interpretable, and also being more accurate if the underlying constraints could be learned in a precise manner. Some recent approaches take this route, and include works by [Paulus et al., 2021, Pogancic et al., 2020, Berthet et al., 2020]. We refer to the latter set of approaches as *neural ILP* architectures. [1]

Learning in neural ILP architectures is complicated by the fact that there is a discrete

[1] Although these methods can also train a Neural-ILP-Neural architecture, studying this is beyond our scope.

optimization (in the form of an ILP layer) at the end of the network, which is typically non-differentiable, making the end-to-end learning of the system difficult. One of the possible ways is to instead use an iterative ILP solving algorithm such as a cutting-plane method [Gomory, 2010] that uses a continuous relaxation in each iteration which is shown to be differentiable due to the introduction of continuous variables [Ferber et al., 2020, Wilder et al., 2019]. Most of these works are concerned with learning only the cost function and assume that the constraints are given. Recent work by Paulus et al. [2021] has proposed an approach to directly pass the gradients through a black-box ILP solver. Specifically, they rely on Euclidean distances between the constraint hyperplanes and the current solution obtained by the ILP solver to approximate the gradients. Though this approach improves the quality of results compared to earlier works involving continuous relaxations, scalability gets severely affected since an ILP has to be solved at every learning iteration, making the training extremely slow.

We are interested in answering the following question: is there a way to train a neural ILP architecture in an end-to-end manner, which does not require an access to an underlying solver? Such an approach, if exists, could presumably result in significantly faster training times, resulting in scalability. In response, we propose a novel technique to differentiate through the learnable constraints as well as the learnable cost function of an unknown Integer Linear Program (ILP). During training, our technique doesn't solve the ILP to compute the gradients. Instead, we cast the learning of ILP constraints (and cost) as learning of a polyhedron, consisting of a set of hyperplanes, such that points inside the polyhedron are treated as positive, and points outside as negative. While a positive point needs to be classified as positive by each of the hyperplanes, a negative point needs to be classified as negative by at least one of the hyperplanes. Our formulation incorporates the learning of ILP cost also as learning of one of the hyperplanes in the system. We formulate a novel margin-based loss to learn these hyperplanes in a joint fashion. A covariance based regularizer that minimizes the cosine similarity between all pairs of hyperplanes ensures that learned constraints (hyperplanes) are not redundant. Since the training data comes only with positive examples, i.e., solutions to respective optimization problems, we develop several techniques for sampling the negatives, each of which is central to the effective learning of our hyperplanes in our formulation.

We present several experiments on problems which require learning of ILP constraints and cost, with both symbolic as well as perceptual input. These include solving symbolic sudoku as well as visual sudoku in which we are given an image of a partially filled board [Wang et al., 2019] (perceptual); ILPs with random constraints (symbolic), Knapsack from sentence description (perceptual) and key-point matching (perceptual) from [Paulus et al., 2021]. Our closest competitor, CombOptNet [Paulus et al., 2021], can not solve

even the smallest of the sudoku boards sized 4×4, whereas we can easily scale to 9×9, getting close to 100% accuracy. We are slightly better on key-point matching, and obtain significantly better accuracy on random ILPs and knapsack, especially on large problem sizes. We also outperform purely neural baselines (wherever applicable).

6.2 Related Works

In part III, we looked at neural models such as NLM [Dong et al., 2019], RRN [Palm et al., 2018], that replace a combinatorial solver altogether and *implicitly* encodes the rules or constraints in their weights, and learns them from data. Other than that, Selsam et al. [2019], Amizadeh et al. [2019a,b] train a GNN for solving any CSP when constraints are *explicitly* given in a standard form such as CNF, DNF or Boolean Circuits. This is clearly different from the current work since we are interested in explicit learning the constraints of the underlying optimization problem. Inverse Optimization [Chan et al., 2021] aims to learn the constraints (cost) of a linear program (LP) from the observed optimal decisions. Recently [Tan et al., 2019, 2020] use the notion of Parameterized Linear Programs (PLP) in which both the cost and the constraints of an LP are parameterized by unknown weights and a feature vector. [Gould et al., 2019] show how to differentiate through continuous constrained optimization problem using the notion of a '*declarative node*' that uses implicit function theorem [Mingari Scarpello and Ritelli, 2002]. Similar to this are other works [Amos and Kolter, 2017, Agrawal et al., 2019] that define methods for differentiating through other continuous problems like convex quadratic programs (QPs) or cone programs. While all these techniques are concerned with learning the constraints (cost) for optimization problems defined over continuous variables, our focus is on learning constraints (cost) for an ILP which involves optimization over discrete variables and can be a significantly harder problem.

In the space of learning cost for an ILP, several approaches have been proposed recently. [Pogancic et al., 2020] define gradient w.r.t. linear cost function that is optimized over a given discrete space. [Rolínek et al., 2020a,b] exploit it for the task of Deep Graph Matching, and for differentiating through rank based metrics such as Average Precision/Recall respectively. [Berthet et al., 2020] replace the black-box discrete solver with its smooth perturbed version during training and exploit 'perturb-and-MAP' [Papandreou and Yuille, 2011] to compute the gradients. On similar lines, [Niepert et al., 2021] also exploit perturb-and-MAP but propose a different noise model for perturbation and also extends to the case when the output of the solver may feed into a downstream neural model. These methods assume the constraints to be given, and only learn the cost for an ILP.

Other methods use relaxations of specific algorithms for the task of learning the constraints and cost for an ILP. Ferber et al. [2020] backprop through the KKT conditions of the LP relaxation created iteratively while using cutting-plane method [Gomory, 2010] for solving a MILP; Wilder et al. [2019] add a quadratic penalty term to the continuous relaxation and use implicit function theorem to backprop through the KKT conditions, as done in Amos and Kolter [2017] for smooth programs; Mandi and Guns [2020] instead add a log-barrier term to get the LP relaxation and differentiate through its homogeneous self-dual formulation linking it to the iterative steps in the interior point method. Wang et al. [2019] uses a low rank SDP relaxation of MAXSAT and defines how the gradients flow during backpropagation. Presumably, these approaches are limited in their application since they rely on specific algorithmic relaxations.

Instead of working with a specific algorithm, some recent works differentiate through a black-box combinatorial solver for the task of learning constraints [Paulus et al., 2021]. The solver is called at every learning step to compute the gradient. This becomes a bottleneck during training, since the constraints are being learned on the go, and the problem could become ill-defined resulting in a larger solver time, severely limiting scalability of such approaches. In contrast, we would like to perform this task in a solver-free manner. [Pan et al., 2020] propose learning of constraints for the task of structured prediction. They represent the linear constraints compactly using a specific two layer Rectifier Network [Pan and Srikumar, 2016]. A significant limitation is that their method can only be used for learning constraints, and not costs. Further, their approach does not do well experimentally on the benchmark domains that we compare with. [Meng and Chang, 2021] propose a non-learning based approach for mining constraints in which for each training data $(\mathbf{c}, \mathbf{y})$, a new constraint is added which essentially imply that no other target can have better cost than the given target $\mathbf{y}$ for the corresponding cost $\mathbf{c}$. We are specifically interested in learning not only the constraints, but also the cost of an ILP from data. Convex Polytope Machines (CPM) [Kantchelian et al., 2014] learns a non linear binary classifier in the form of a polytope from a dataset of positive and negative examples. In contrast, our goal is to learn the cost as well as constraints for an ILP where both the constraints and the cost could be parameterized by another neural network. Also, we do not have access to any negative samples.

6.3 Differentiable ILP Loss

6.3.1 Background and Task Description

We are interested in learning how to solve combinatorial optimization problems that can be expressed as an Integer Linear Program (ILP) with equality as well as inequality constraints:

$$\arg\min_{\mathbf{z}} \mathbf{c}^T\mathbf{z} \;\text{ subject to }\quad \mathbf{U}\mathbf{z}=\mathbf{v};\;\; \mathbf{G}\mathbf{z}+\mathbf{h}\geq \mathbf{0};\;\; \mathbf{z}\in\mathbb{Z}^n \tag{6.1}$$

Here $\mathbf{c}\in\mathbb{R}^n$ represents an n dimensional cost vector, the matrix $\mathbf{U}\in\mathbb{R}^{m_1\times n}$ and $\mathbf{v}\in\mathbb{R}^{m_1}$ represent m_1 linear equality constraints, and the matrix $\mathbf{G}\in\mathbb{R}^{m_2\times n}$ and $\mathbf{h}\in\mathbb{R}^{m_2}$ represent m_2 linear inequality constraints, together defining the feasible region.

Without loss of generality, one can replace an equality constraint $\mathbf{u}_i^T\mathbf{z}=\mathbf{v}_i$ by two inequality constraints: $\mathbf{u}_i^T\mathbf{z}\geq\mathbf{v}_i$ and $-\mathbf{u}_i^T\mathbf{z}\geq-\mathbf{v}_i$. Here $\mathbf{u}_i$ and $\mathbf{v}_i$ represent the i^{th} row of the matrix $\mathbf{U}$ and i^{th} element of the vector $\mathbf{v}$ respectively. Using this, one can reduce ILP in eq. (6.1) to an equivalent form with just inequality constraints:

$$\arg\min_{\mathbf{z}} \mathbf{c}^T\mathbf{z} \;\text{ subject to }\quad \mathbf{A}\mathbf{z}+\mathbf{b}\geq\mathbf{0}, \mathbf{z}\in\mathbb{Z}^n \tag{6.2}$$

Here matrix $\mathbf{A}\in\mathbb{R}^{m\times n}$ is row-wise concatenation of the matrices $\mathbf{G},\mathbf{U}$ and $-\mathbf{U}$. Vector $\mathbf{b}\in\mathbb{R}^m$ is a row-wise concatenation of vectors $\mathbf{h},-\mathbf{v}$ and $\mathbf{v}$, and $m=2m_1+m_2$ is the total number of inequality constraints. We represent the i^{th} constraint $\mathbf{a}_i^T\mathbf{z}+\mathbf{b}_i\geq 0$, as $[\mathbf{a}_i|\mathbf{b}_i]$. The integer constraints $\mathbf{z}\in\mathbb{Z}^n$ make the problem NP hard.

Neural ILP Architecture: In a neural ILP architecture, the constraint matrix $\mathbf{A}$, vector $\mathbf{b}$, and the cost vector $\mathbf{c}$ are neural functions $f_{\mathbf{A}}$, $f_{\mathbf{b}}$, and $f_{\mathbf{c}}$ (of input x) parameterized by learnable $\Theta=(\theta_{\mathbf{A}},\theta_{\mathbf{b}},\theta_{\mathbf{c}})$, *i.e.*, $\mathbf{A}=f_{\mathbf{A}}(x;\theta_{\mathbf{A}})$; $\mathbf{b}=f_{\mathbf{b}}(x;\theta_{\mathbf{b}})$; $\mathbf{c}=f_{\mathbf{c}}(x;\theta_{\mathbf{c}})$. This results in a different ILP for each input x. For an input x, the solution given by a neural ILP model, $\mathbf{M}_\Theta(x)$, is nothing but the optima of the following paramterized ILP where $\mathbf{A},\mathbf{b},\mathbf{c}$ are replaced by corresponding neural functions $f_{\mathbf{A}},f_{\mathbf{b}},f_{\mathbf{c}}$ evaluated at input x:

$$\mathbf{M}_\Theta(x)=\arg\min_{\mathbf{z}} \left(f_{\mathbf{c}}(x;\theta_{\mathbf{c}})\right)^T\mathbf{z} \;\text{ subject to }\quad f_{\mathbf{A}}(x;\theta_{\mathbf{A}})\mathbf{z}+f_{\mathbf{b}}(x;\theta_{\mathbf{b}})\geq\mathbf{0}, \mathbf{z}\in\mathbb{Z}^n \tag{6.3}$$

Example of visual-sudoku: In $k\times k$ visual-sudoku, the input x is an image of a sudoku puzzle and $\mathbf{y}^*\in\{0,1\}^n$ is the corresponding solution, represented as a $n=k^3$ dimensional binary vector: the integer in each of the k^2 cells is represented by a k dimensional one-hot binary vector. Function $f_{\mathbf{c}}(x;\theta_{\mathbf{c}})$ parameterizing the cost is nothing but a neural digit classifier that classifies the content of each of the k^2 cells into one of the k classes. The neural functions $f_{\mathbf{A}}$ and $f_{\mathbf{b}}$ are independent of the input x as the constraints are the same for every $k\times k$ sudoku puzzle. Therefore, $f_{\mathbf{A}}(x;\theta_{\mathbf{A}})=\theta_{\mathbf{A}}$, and $f_{\mathbf{b}}(x;\theta_{\mathbf{b}})=\theta_{\mathbf{b}}$, where $\theta_{\mathbf{A}}$ is just a learnable matrix of dimension $m\times n$ and $\theta_{\mathbf{b}}$ is a learnable vector of dimension m. See Bartrlett et al. [2008] for an ILP formulation of $k\times k$ sudoku with $4k^2$

equality constraints in a $n = k^3$ dimensional binary space $\{0,1\}^n$.

Learning Task: Given a training dataset $\mathcal{D} = \{(x_s, \mathbf{y}^*_s) \mid s \in \{1 \dots S\}\}$ with S training samples, the task is to learn the parameters $\Theta = (\theta_{\mathbf{A}}, \theta_{\mathbf{b}}, \theta_{\mathbf{c}})$, such that $\mathbf{y}^*_s = \mathbf{M}_\Theta(x_s)$ for each s. To do so, one needs to define derivatives $\frac{\partial L}{\partial \mathbf{A}}$, $\frac{\partial L}{\partial \mathbf{b}}$, and $\frac{\partial L}{\partial \mathbf{c}}$ *w.r.t.* $\mathbf{A}, \mathbf{b}$, and $\mathbf{c}$ respectively, of an appropriate loss function L. Once such a derivative is defined, one can easily compute derivatives *w.r.t.*$\theta_{\mathbf{A}}, \theta_{\mathbf{b}}$, and $\theta_{\mathbf{c}}$ using the chain rule: $\frac{\partial L}{\partial \theta_{\mathbf{A}}} = \frac{\partial L}{\partial \mathbf{A}} \frac{\partial f_{\mathbf{A}}(x;\theta_{\mathbf{A}})}{\partial \theta_{\mathbf{A}}}$; $\frac{\partial L}{\partial \theta_{\mathbf{b}}} = \frac{\partial L}{\partial \mathbf{b}} \frac{\partial f_{\mathbf{b}}(x;\theta_{\mathbf{b}})}{\partial \theta_{\mathbf{b}}}$; and $\frac{\partial L}{\partial \theta_{\mathbf{c}}} = \frac{\partial L}{\partial \mathbf{c}} \frac{\partial f_{\mathbf{A}}(x;\theta_{\mathbf{c}})}{\partial \theta_{\mathbf{c}}}$. Hence, in the formulation below, we only worry about computing gradients *w.r.t.* the constraint matrix $\mathbf{A}$, vector $\mathbf{b}$, and the cost vector $\mathbf{c}$.

The existing approaches, *e.g.*, [Paulus et al., 2021], explicitly need access to current model prediction $\mathbf{M}_\Theta(x)$ This requires solving the ILP in eq. (6.3), making the learning process extremely slow. In contrast, we present a *'solver-free'* framework for computation of an appropriate loss and its derivatives *w.r.t.* $\mathbf{A}, \mathbf{b}$, and $\mathbf{c}$. Our framework does not require solving any ILP while training, thereby making it extremely scalable as compared to existing approaches.

6.3.2 A Solver-free Framework

Conversion to a constraint satisfaction problem: As a first step, we convert the constraint optimization problem in eq. (6.2) to an equivalent constraint satisfaction problem by introducing an additional '*cost–constraint*': $\mathbf{c}^T\mathbf{z} \leq \mathbf{c}^T\mathbf{y}^*$, equivalent to $\mathbf{a}_{m+1} = -\mathbf{c}$ and $\mathbf{b}_{m+1} = \mathbf{c}^T\mathbf{y}^*$.

Note that the above cost–constraint separates the solution $\mathbf{y}^*$ of the original ILP in eq. (6.2) from the other feasible integral points. This is because $\mathbf{y}^*$ must achieve the minimum objective value $\mathbf{c}^T\mathbf{y}^*$ amongst all the feasible integral points and hence no other feasible integral point $\mathbf{z}$ can obtain objective value less than $\mathbf{c}^T\mathbf{y}^*$. The new constraint $[\mathbf{a}_{m+1}|\mathbf{b}_{m+1}]$ together with the original m constraints guarantee that $\mathbf{y}^*$ is the only solution satisfying all of the $(m+1)$ constraints. This results in the following equivalent linear constraint satisfaction problem:

$$\arg\min_{\mathbf{z}} \mathbf{0}^T\mathbf{z} \;\text{ subject to }\; \mathbf{A}\mathbf{z} + \mathbf{b} \geq \mathbf{0} \;\;;\;\; \mathbf{c}^T\mathbf{z} \leq \mathbf{c}^T\mathbf{y}^* \;\;;\;\; \mathbf{z} \in \mathbb{Z}^n \tag{6.4}$$

Constructing such an equivalent satisfaction problem requires access to the solution $\mathbf{y}^*$ of the original ILP in eq. (6.2) and that is already available to us for the training data samples. By rolling up the cost vector $\mathbf{c}$ into an additional constraint with the help of $\mathbf{y}^*$, we have converted our original objective of learning both the cost and constraints to just learning of constraints in eq. (6.4).

The main intuition behind our framework comes from the observation that each of the m linear constraints defining the feasible region are essentially m linear binary classifiers (hyperplanes) in $\mathbb{R}^n$ separating the ground truth $\mathbf{y}^*$ from the infeasible region. The

additional cost–constraint in eq. (6.4) separates $\mathbf{y}^*$ from other integral samples feasible *w.r.t.* the original m constraints. Learning constraints of an ILP is akin to simultaneously learning m linear binary classifiers separating $\mathbf{y}^*$ from other infeasible points of the original ILP along with learning a classifier for cost–constraint, separating $\mathbf{y}^*$ for other feasible integral points.

For a vector $\mathbf{y}^*$ to lie inside the feasible region, *all* the classifiers need to classify it positively, and hence it acts as a positive data point for all the $m+1$ binary classifiers. In the absence of explicitly provided negative samples, we propose a couple of strategies for sampling them from $\mathbb{Z}^n$ for each ground truth $\mathbf{y}^*_s$. We discuss them in detail in section 6.3.3. For now, let $\mathcal{N}_{\mathbf{y}^*}$ be the set of all the negative samples generated by all our sampling techniques for a ground truth positive sample $\mathbf{y}^*$. In contrast to a positive point which needs to satisfy *all* the $m+1$ constraints, a negative point becomes infeasible even if it violates *any one* of the $m+1$ constraints. As a result, it would suffice if any one of the $m+1$ classifiers correctly classify it as a negative sample. (See fig. 6.1 for an illustration). While learning, one cannot assign a negative sample to any specific classifier as their parameters are being updated continuously. Instead, we make a soft assignment depending upon the Euclidean distance of the negative sample from the hyperplane. With this intuition, we now formally define our ILP–Loss.

Formulating solver-free ILP–Loss: Let $d(\mathbf{z}; [\mathbf{a}_i|\mathbf{b}_i]) = \frac{\mathbf{a}_i^T\mathbf{z}+\mathbf{b}_i}{|\mathbf{a}_i|}$ represent the signed Euclidean distance of a point $\mathbf{z} \in \mathbb{R}^n$ from the hyperplane corresponding to the constraint $[\mathbf{a}_i|\mathbf{b}_i]$. We want the signed distance from all the hyperplanes to be positive for the ground truth samples and negative from at least one hyperplane for all the sampled negative points. We operationalize this via a margin based loss function:

$$L(\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{y}^*|\mathcal{N}_{\mathbf{y}^*}) = \lambda_{pos}L_+ + \lambda_{neg}L_- + \lambda_{cov}L_o \quad \text{where} \tag{6.5}$$

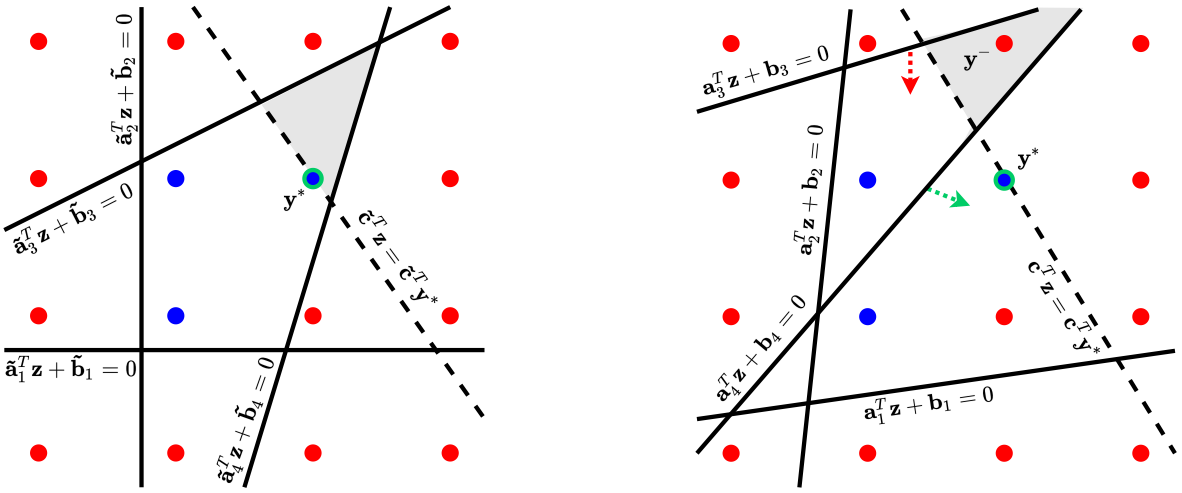
$$L_+ = \frac{1}{m}\sum_{i=1}^{m} \max\{0, \mu^+ - d(\mathbf{y}^*; [\mathbf{a}_i|\mathbf{b}_i])\} \tag{6.6}$$

$$L_- = \frac{1}{|\mathcal{N}_{\mathbf{y}^*}|}\sum_{\mathbf{y}^-\in\mathcal{N}_{\mathbf{y}^*}}\sum_{i=1}^{m+1} w^i_{\mathbf{y}^-} \max\{0, \mu^- + d(\mathbf{y}^-; [\mathbf{a}_i|\mathbf{b}_i])\} \tag{6.7}$$

$$L_o = \sum_{i,j=1;i\neq j}^{m} \frac{\mathbf{a}_i^T\mathbf{a}_j}{|\mathbf{a}_i||\mathbf{a}_j|} \simeq \left(\sum_{i=1}^{m}\frac{\mathbf{a}_i}{|\mathbf{a}_i|}\right)^2 \tag{6.8}$$

$$w^i_{\mathbf{y}^-} = \frac{e^{(-d(\mathbf{y}^-;[\mathbf{a}_i|\mathbf{b}_i])/\tau)}}{\sum_{j=1}^{m+1} e^{(-d(\mathbf{y}^-;[\mathbf{a}_j|\mathbf{b}_j])/\tau)}} \tag{6.9}$$

We call L_+, L_-, and L_o as the positive, negative and covariance loss respectively. The average in L_+ ensures that a ground truth sample is positively classified by *all* the classifiers. μ^+ and μ^- are the hyperparameters representing the margins for the positive



(a) Ground Truth ILP

(b) Intermediate Learnt ILP

Figure 6.1: An illustration of our framework. Figure on the left shows 4 ground truth constraints that need to be learnt. Blue dots are the only feasible integral points w.r.t. the 4 constraints. Shaded area containing only the dot with green border is the feasible region after we add the cost constraint (dashed line). Figure on the right shows an intermediate scenario while learning. The green-bordered dot (positive) is outside the intermediate 4^{th} constraint and the red dot (negative) is inside the intermediate 3^{rd} constraint. Positive and negative losses encourage the 4^{th} and the 3^{rd} hyperplanes to move in the direction shown by the green and red dotted arrows respectively.

and the negative points respectively. $w^i_{\mathbf{y}^-}$ in L_- represents the soft assignment of $\mathbf{y}^-$ to the i^{th} constraint $[\mathbf{a}_i|\mathbf{b}_i]$ and is computed by temperature annealed softmax over the negative distances in eq. (6.9). Softmax ensures that the hyperplane which is most confident to classify it as negative gets the maximum weight. When $\mathbf{y}^-$ lies inside the feasible region, then the most confident classifier is the one closest to $\mathbf{y}^-$. To avoid the pathological behaviour of decreasing the loss by changing the weights, we ensure that gradients do not flow through $w^i_{\mathbf{y}^-}$ in eq. (6.7).

The temperature parameter τ needs to be annealed as the training progresses. A high temperature initially can be seen as '*exploration*' for the right constraint that will be violated by $\mathbf{y}^-$. This is important as the constraints are also being learnt and are almost random initially, so a given negative $\mathbf{y}^-$ should not commit to a particular hyperplane. Additionally, this encourages multiple constraints to be violated for each negative, which leads to a robust set of constraints. As the training progresses, we reduce the temperature τ, ensuring that the most confident classifier with least signed distance gets almost all the weight. If $\mathbf{y}^-$ is correctly classified as a negative with a margin μ^- by *any* classifier *i.e.*, $d(\mathbf{y}^-;[\mathbf{a}_i|\mathbf{b}_i]) \leq -\mu^-$ for some i, then the corresponding negative loss, $\max\{0, \mu^- + d(\mathbf{y}^-;[\mathbf{a}_i|\mathbf{b}_i])\}$, becomes zero, and a low value of τ ensures that it gets all the

weight.

The last term, L_o acts as a regularizer and tries to ensure that no two learnt constraints are similar. This is done by minimizing the cosine similarity between all the pairs of constraints. The weights $\lambda_{pos}, \lambda_{neg}$, and λ_{cov} are computed automatically with a multi-loss weighing technique using coefficient of variations as described in Groenendijk et al. [2021]. Intuitively, the loss term with maximum variance over the learning iterations adaptively gets most of the weight.

Other details:

Parameterization of equality constraints: Recall that we replace an equality $\mathbf{u}_i^T\mathbf{z} = \mathbf{v}_i$ by two inequality constraints: $\mathbf{v}_i \leq \mathbf{u}_i^T\mathbf{z} \leq \mathbf{v}_i$. In practice, to enhance learnability, we add a small margin of ϵ on both sides: $\mathbf{v}_i - \epsilon \leq \mathbf{u}_i^T\mathbf{z} \leq \mathbf{v}_i + \epsilon$. We pick an ϵ small enough so that the probability of the new feasible region to include an infeasible integral point is negligible.

Known boundaries: In many cases, the boundary conditions on the output variables are known, *i.e.*, $\mathbf{z} \in \mathcal{Y} = [l, u]^n$, where l and u are the lower and upper bounds on each dimension of $\mathbf{z}$. We handle this by adding known boundary constraints: $l \leq \mathbf{z}_i \leq u, \forall i \in \{1, \ldots, n\}$.

Over-parameterization of constraints: As done in Paulus et al., we also over-parameterize each constraint hyperplane by an additional learnable offset vector $\mathbf{o}_i$ which can be viewed as its own local origin. Radius $\mathbf{r}_i$ represents its distance from its own origin $\mathbf{o}_i$, resulting in the following hyperplane in the base coordinate system: $\mathbf{a}_i^T\mathbf{z} + \mathbf{r}_i - \mathbf{o}_i^T\mathbf{a}_i/|\mathbf{a}_i| \geq 0$.

Initialization of A: The way we initialize the constraints may have an impact on the learnability. While [Paulus et al., 2021] propose to sample each entry of $\mathbf{a}_i$ uniformly (and independently) between $[-0.5, 0.5]$, we also experiment with a standard Gaussian initialization. The latter results in initial hyperplanes with their normal directions ($\mathbf{a}_i$'s) uniformly sampled from a unit hyper-sphere. In expectation, such initialization achieves minima of L_o that measures total pairwise covariance.

6.3.3 Negative Sampling

A meaningful computation of ILP–Loss in eq. (6.5) depends crucially on the negative samples $\mathcal{N}_{\mathbf{y}}^*$. Randomly sampling negatives from $\mathbb{Z}^n$ is one plausible strategy, but it may not be efficient from the learning perspective: any point which is far away from any of the classifiers will easily be classified as negative and will not contribute much to the loss function. In response, we propose multiple alternatives for sampling the negative points:

1. **Integral k-hop neighbours:** To generate negatives, we sample the integral neighbours that are at a Hamming Distance of k from $\mathbf{y}^*$. For small k, these form the

hardest negatives as they are closest to the positive point. Note that it is possible for a few integral neighbours to be feasible *w.r.t.* the m constraints, but they must have worse cost than the given ground truth $\mathbf{y}^*$. Such samples contribute towards learning the cost parameters $\theta_{\mathbf{c}}$.

2. **Project and sample:** In this method, we project the ground truth $\mathbf{y}^*$ on each of the m hyperplanes and then randomly sample a neighbour of the projection, generating a total of m negatives. Sampling probability in the j^{th} dimension depends on the j^{th} coordinate of the projection: if value of the j^{th} coordinate is $r \notin \mathbb{Z}$, then we sample $floor(r)$ and $ceil(r)$ with probability $ceil(r) - r$ and $r - floor(r)$ respectively. If $r \in \mathbb{Z}$, then we sample r with probability 1. Projection samples are close to the boundary of the currently learnt polyhedron, thus taking the training progress into account. Further, each hyperplane is likely to be assigned to a close-by negative due to projection sampling.

3. **Batch negatives:** In this strategy, we consider every other ground truth $\mathbf{y}^*_{s'}$ $s' \neq s$ in the minibatch as a potential negative sample for $\mathbf{y}^*_s$. We call it as a 'batch–negative'. This is particularly useful for learning the cost parameters $\theta_{\mathbf{c}}$ when the learnable constraints are the same for all the ground truth samples, such as in sudoku. In such cases, a batch–negative $\mathbf{y}^*_{s'}$ being a feasible point of the original ILP formulation, must always satisfy all of the m learnable constraints of the original ILP in eq. (6.2). Hence, the only way for $\mathbf{y}^*_{s'}$ to be correctly classified as a negative for $\mathbf{y}^*_s$ is by violating the cost constraint in eq. (6.4), Learning cost parameters $\theta_{\mathbf{c}}$ that result in violation of the cost constraint for every batch–negative helps in ensuring that the ground truth $\mathbf{y}^*_s$ indeed has the minimum cost.

4. **Solver based:** Although our approach is motivated by the objective of avoiding solver calls, our framework is easily extensible to solver-based training by using the solution to the currently learnt ILP as a negative sample. This is useful when the underlying neural networks parameterizing $\mathbf{A}, \mathbf{b}$ or $\mathbf{c}$ are the bottleneck instead of the ILP solver. While we do not use solver negatives by default, we demonstrate their effectiveness in one of our experiments where the network parameterizing $\mathbf{c}$ indeed takes most of the computation time.

6.4 Experiments

The goal of our experiments is to evaluate the effectiveness and scalability of our proposed approach compared to the SOTA black-box solver based approach, i.e., CombOptNet

[Paulus et al., 2021]. We experiment on 4 problems: symbolic and visual sudoku [Wang et al., 2019], and three problems from [Paulus et al., 2021]: random constraints, knapsack from sentence description and key-point matching. We also compare with an appropriately designed neural baseline for each of our datasets. To measure scalability, in each of the domains, we compare the performance of each of the algorithms in terms of training time, as well as accuracy obtained for varying problem sizes. For each of the algorithms, we report time till the epoch that achieves the best val set accuracy and excludes time taken during validation. We kept a maximum time limit of 12 hours for the training of each algorithm. We next describe the details of each of these datasets, appropriate baselines, and our results. To solve the learnt ILPs, we use Gurobi ILP solver [Gurobi Optimization, LLC, 2022] available under 'Named-user academic license'. All our experiments were run on 11 GB 'GeForce GTX 1080 Ti' GPUs installed on a machine with 2.60GHz Intel(R) Xeon(R) Gold 6142 CPU. For each of the algorithms, we kept a maximum time limit of 12 hours for training.

6.4.1 Symbolic and Visual Sudoku

This task involves learning the rules of the sudoku puzzle. For symbolic sudoku, the input x is a matrix of digits whereas for a visual sudoku, x is an image of a sudoku puzzle where a digit is replaced by a random MNIST image of the same class. A $k \times k$ sudoku puzzle can be viewed as an ILP with $4k^2$ equality constraints over k^3 binary variables [Bartrlett et al., 2008]. For symbolic sudoku, each of the k^2 cells of the puzzle is represented by a k dimensional binary vector which takes a value of 1 at i^{th} dimension if and only if the cell is filled by digit i, resulting in a k^3 dimensional representation of the input $\mathbf{x} \in \mathbb{R}^n$ where $n = k^3$. On the other hand, for visual sudoku, each of the digit images in the k^2 cells are decoded (using a neural network) into a k dimensional real vector, resulting in a k^3 dimensional **learnable** cost $\mathbf{c} \in \mathbb{R}^n$. The k^3 dimensional binary output vector (solution) $\mathbf{y}^*$ is created analogous to symbolic input. Our objective is to learn the constraint matrix $\mathbf{A} = f_{\mathbf{A}}(x; \theta_{\mathbf{A}}) = \theta_{\mathbf{A}}$ and vector $\mathbf{b} = f_{\mathbf{b}}(x; \theta_{\mathbf{b}}) = \theta_{\mathbf{b}}$, representing the linear constraints of sudoku. For symbolic sudoku, the cost vector $\mathbf{c} = -\mathbf{x}$ is known, where as for visual sudoku, the cost vector $\mathbf{c} = f_{\mathbf{c}}(x; \theta_{\mathbf{c}})$ is a function of the input image x, parameterized by $\theta_{\mathbf{c}}$ which also needs to be learned.

Dataset: We experiment with 3 different datasets for $k = 4, 6$, and 9. We first build symbolic sudoku puzzles with k digits, and use MNIST to convert a puzzle into an image for visual sudoku. For 9×9 sudokus, we use a standard dataset from Kaggle [Park, 2017], for $k = 4$ we use publically available data from Arcot and Kalluraya [2019], and for $k = 6$ we use the data generation process described in Nandwani et al. [2022a]. We randomly

Table 6.1: Board accuracy and training time for different board sizes of symbolic and visual sudoku (for CombOptNet, "-" denotes time-out after 12 hours)

	Symbolic Sudoku						Visual Sudoku					
	Board Accuracy (%)			Training Time (min)			Board Accuracy (%)			Training Time (min)		
	4x4	6x6	9x9	4x4	6x6	9x9	4x4	6x6	9x9	4x4	6x6	9x9
Neural (RRN)	**100.0**	99.1	91.3	5	7	110	**99.8**	97.5	71.1	120	65	97
CombOptNet	0.0	0.0	0.0	-	-	-	0.0	0.0	0.0	-	-	-
SATNet	100.0	96.8	28.5	1	74	299	98.0	80.8	17.8	79	89	205
ILP–Loss (Ours)	**100.0**	**100.0**	**100.0**	**1**	**2**	**52**	99.7	**98.8**	**98.3**	**3**	**11**	**92**

select $10,000$ samples for training, and 1000 samples for testing for each k. To generate the input images for visual-sudoku, we use the official train and test split of MNIST[Deng, 2012]. Digits in our train and test splits are randomly replaced by images in the MNIST train and test sets respectively, avoiding any leakage.

Hyperparameters and other design choices: (a) # of learnable constraints: We keep $m = (n+1)/2$ as the number of learnable equality constraints where $n = k^3$ is the number of binary variables. (b) Margin: We find that a margin of 0.01 works well across domains and problem sizes. (c) Temperature: we start with a temperature $\tau = 1$ and anneal it by a factor of 0.1 whenever the performance on a small held out set plateaus. (d) Early Stopping: we early stop the training based on validation set performance, with a timeout of 12 hours for each experiment. We bypass the validation bottleneck of solving the ILPs from scratch by providing the gold solutions as hints when invoking Gurobi. (e) Negative Sampling: For sampling neighbors, we select all the n one hop neighbors, and an equal number of randomly selected 2,3 and 4 hop neighbors, resulting in a total of $4n$ neighbors. (f) Initialization: we initialize $\mathbf{a}_i$ from a standard Gaussian distribution for CombOptNet and our method.

Baselines: Our neural baseline is Recurrent Relational Networks (RRN) [Palm et al., 2018]: a purely neural approach for reasoning based on Graph Neural Networks. We use the default set of parameters provided in their paper for training RRNs. We note that the RRN baseline uses additional information in the form of graph structure which is not available to our solver. We make this comparison to see how well we perform compared to one of the SOTA techniques for this problem. We also compare against another neuro-symbolic architecture using SATNet [Wang et al., 2019] as the reasoning layer. The neural component in all the neuro-symbolic methods is a CNN that decodes images into digits.

Results: Table 6.1 compares the performance and training time of our method against the different baselines described above. CompOptNet fails miserably on this problem, not being able to complete training for any of the sizes in the stipulated time. The purely

neural model gives a decent performance and is competitive with ours for smaller board sizes. But for larger board size of 9×9, we beat RRN based model by a significant margin of about 25 points and 9 points in visual and symbolic sudoku, respectively.

While SATNet performs comparable on smaller 4×4 sudoku puzzles, its performance degrades drastically to 17.8% for 9×9 board size on visual sudoku. Wang et al. use a different set of 9×9 puzzles for training and testing sudoku and report 63.2% accuracy on visual sudoku, different from what it obtains on our dataset. Hence we trained both SATNet and our model on the dataset available on SATNet's Github repo. Interestingly, on their visual sudoku dataset which we believe to be easier (as shown by performance numbers), our run of SATNet achieves 71.0% board accuracy whereas our method achieves 98.3%.

6.4.2 Random constraints

This is the synthetic dataset borrowed from Paulus et al.. The training data $\mathcal{D} = \{(\mathbf{c}_s, \mathbf{y}^*_s) | s \in \{1, \ldots, S\}\}$ is created by first generating a random polyhedron $\mathcal{P} \subseteq \mathcal{Y}$ by sampling m' hyperplanes $\{[\mathbf{a}'_i | \mathbf{b}'_i] \mid \mathbf{a}'_i, \mathbf{b}'_i \in \mathbb{R}^n, i \in \{1, \ldots, m'\}\}$ in an n dimensional bounded continuous space $\mathcal{Y}$. A cost vector $\mathbf{c}_s \in \mathbb{R}^n$ is then randomly sampled and corresponding $\mathbf{y}^*_s$ is obtained as $\mathbf{y}^*_s = \arg\min_{\mathbf{z} \in \mathcal{P}} \mathbf{c}_s^T \mathbf{z}, \mathbf{z} \in \mathbb{Z}^n$. Objective is to learn a constraint matrix $\mathbf{A}$ and vector $\mathbf{b}$ such that for a ground truth $(\mathbf{c}, \mathbf{y}^*)$ pair, $\mathbf{y}^* = \arg\min_{\mathbf{z} \in \mathcal{Y}} \mathbf{c}^T \mathbf{z}$, $\mathbf{z} \in \mathbb{Z}^n$, and $\mathbf{A}\mathbf{z} + \mathbf{b} \geq \mathbf{0}$.

Dataset: Restricting $\mathcal{Y}$ to $[0, 1]^n$ and $[-5, 5]^n$ results in two variations, referred as 'binary' and 'dense' settings, respectively. For both of the output spaces, we experiment with four different settings in $n = 16$ dimensional space by varying the number of ground truth constraints as $m' = 1, 2, 4$, and 8. For each m', we experiment with all the 10 datasets provided by Paulus et al. and report mean and standard error over the 10 models. The training data in each case consists of 1600 $(\mathbf{c}, \mathbf{y})$ pairs and model performance is tested for 1000 cost vectors.

Hyperparameters and other design choices: We keep number of learnable constraints as twice the number of ground truth constraints, *i.e.*, $m = 2m'$, as Paulus et al. report best performance in most of the settings with it. Here we initialize $\mathbf{a}_i$ uniformly between $[-0.5, 0.5]$. Rest of the hyperparamters are set as in the case of sudoku.

Baseline: Here we compare only against CombOptNet. A neural baseline (an MLP) is shown to perform badly for this symbolic problem in Paulus et al.. Hence we exclude this in our experiments.

Results: Table 6.2 presents the comparison of the two algorithms in terms of accuracy as well as time taken for training. While we perform marginally better than CombOptNet

Table 6.2: Mean of the vector accuracy ($\mathbf{M}_\Theta(x) = \mathbf{y}^*$) and training time over the 10 random datasets in each setting of Random constraints. Number of learnable constraints is twice the number of ground truth constraints. See section D.1 for std. err. over the 10 runs.

		Vector Accuracy (%)				Training Time (min)			
		1	2	4	8	1	2	4	8
Binary	CombOptNet	97.6	95.3	84.3	63.4	8.2	13.5	26.5	40.8
	ILP–Loss (Ours)	**97.8**	**96.0**	**92.8**	**87.8**	**7.3**	**11.6**	**18.1**	**27.5**
Dense	CombOptNet	89.3	74.8	34.3	2.0	9.9	16.8	24.7	48.2
	ILP–Loss (Ours)	**96.6**	**86.3**	**74.0**	**41.5**	**7.3**	**15.6**	**17.6**	**20.6**

in terms of training time, our performance is significantly better, on almost all problems in the dense setting, and the larger problems in the binary setting. We are roughly 25 and 40 accuracy points better than CombOptNet on the largest problem instance with 8 constraints in the binary and dense settings respectively. This result demonstrates that our approach is not only faster in terms of training time, but also results in better solutions compared to the baseline, validating the effectiveness of our approach.

6.4.3 Knapsack from Sentence Descriptions

This task, also borrowed from Paulus et al., is based on the classical NP-hard Knapsack problem. Each input consists of N sentences describing the price and weight of N items and the objective is to select a subset of items that maximizes their total price while keeping their total weight less than the knapsack's capacity C: $\arg\max \mathbf{p}^T\mathbf{z}$ s.t. $\mathbf{w}^T\mathbf{z} \leq C, \mathbf{z} \in \{0,1\}^N$, where $\mathbf{p}, \mathbf{w} \in \mathbb{R}^N$ are the price and weight vectors, respectively. Each sentence has been converted into a $d = 4096$ dimensional dense embedding using Conneau et al. [2017], so that each input is $N \times d$ dimensional vector $\mathbf{x}$. The corresponding output is $\mathbf{y}^* \in \{0,1\}^N$. The knapsack capacity C is fixed for all instances in a dataset. The task is to learn the parameters $\theta_\mathbf{c}$, $\theta_\mathbf{A}$ and $\theta_\mathbf{b}$ of a neural network that extracts the cost and the constraints from $\mathbf{x}$. Note that here both the cost and the constraints need to be inferred from the input.

Dataset: The dataset in Paulus et al. has 4500 train and 500 test instances with fixed $N = 10$ items, extracted from a corpus containing $50,000$ sentences and their embeddings. In addition to experimenting with the original dataset, to demonstrate the scalability of our method, we also bootstrap new datasets with $N = 15, 20, 25$ and 30, by randomly selecting sentences from the original corpus. Each bootstrapped dataset has 4500 train and 500 test instances.

Table 6.3: Mean accuracy and train-time over 10 runs for various knapsack datasets with different number of items in each instance. For $N = 25, 30$, "-" represents timeout of 12 hours and we evaluate using the latest snapshot of the model obtained within 12 hours of total training. See section D.1 for std. err. over the 10 runs.

	Vector Accuracy (mean in %)					Training Time (mean in min)				
	10	15	20	25	30	10	15	20	25	30
CombOptNet	63.2	48.2	30.1	2.6	0.0	**41.0**	61.4	153.0	-	-
ILP–Loss (Ours)	**71.4**	**58.5**	**48.7**	**41.0**	**28.4**	44.0	**51.0**	**82.2**	**106.1**	**111.6**

Hyperparameters and other design choices: Following Paulus et al., we use a two-layer MLP with hidden-dimension 512 to extract $\mathbf{c}$ and $\mathbf{A}$ from N d-dimensional sentence embeddings, and keep $\mathbf{b} = 1$. Number of constraints m is set to 4, a setting which achieves best performance in Paulus et al.. A notable difference is in the output layer of our MLP. Paulus et al. assume access to the ground truth price and weight range ($[10, 45]$ and $[15, 35]$ respectively), and use a sigmoid output non-linearity with suitable scale and shift to produce $\mathbf{A}$ and $\mathbf{c}$ in the correct range. We do the same for CombOptNet, but for ILP–Loss we simply use a linear activation at the output. We note that training CombOptNet with linear activation without access to the ground truth ranges gives poorer results.

Results: Table 6.3 presents a comparison between the training time and accuracy of our method against CombOptNet. We significantly outperform the baseline in terms of accuracy across all the datasets. For the smallest problem size with only 10 items, our training time is comparable to the baseline. This is expected as for small problems, ILP may not be the bottleneck, and the relative speedup obtained by solver free method gets offset by the increased number of iterations it may require to train. Our relative gain increases significantly with increasing problem size. CombOptNet fails to complete even a single epoch in the stipulated time and results in 0% accuracy on the largest problem.

6.4.4 Keypoint Matching

Here we experiment on a real-world task of matching key points between two images of different objects of the same type Rolínek et al. [2020b]. The input x consists of a pair of images (I_1, I_2) of the same type along with a shuffled list of coordinates (pixels) of k keypoints in both images. The task is to match the same keypoints in the two images with each other. The ground truth constraint enforces a bijection between the keypoints in the two images. The output $\mathbf{y}$ is represented as a $k \times k$ binary permutation matrix, *i.e.*, entries in a row or column should sum to 1. The cost $\mathbf{c}$ is a $k \times k$ sized vector parameterized by

$\theta_{\mathbf{c}}$. The goal is to learn $\mathbf{A}$, $\mathbf{b}$ and $\theta_{\mathbf{c}}$.

Dataset: We experiment with image pairs in SPair-17k [Min et al., 2019] dataset used for the task of keypoint matching in [Rolínek et al., 2020b]. Since the neural ILP models can deal with a fixed size output space, we artificially create 4 datasets for 4, 5, 6 and 7 keypoints from the original dataset. While generating samples for k keypoints dataset, we randomly sample any k annotated pairs from input image pairs that have more than k keypoints annotated. See table 6.4 for details.

Hyperparameters and other design choices: For each k, the number of learnable constraints is set to $2k$: same as the number of ground truth constraints. For keypoints 5,6 and 7, in addition to random initialization, we also experiment by initializing the backbone cost parameters $\theta_{\mathbf{c}}$ with the one obtained by training it on 4 keypoints and pick the one which obtains better accuracy on val set. This happens for all the methods for keypoints 6 and 7.

Table 6.4: Keypoint Matching: Number of train and test samples for datasets with different keypoints

Num Keypoints	**4**	**5**	**6**	**7**
#Test	10,474	9,308	7,910	6,580
#Train	43,916	37,790	31,782	26,312

For ILP–Loss with only solver and batch negatives (ILP–Loss + Sol.), we start with a temperature $\tau = 0.1$ and anneal it by a factor of 0.5 whenever performance on a small validation set plateaus. For ILP–Loss with only solver–free negatives, we start with a temperature $\tau = 0.5$ and anneal it by a factor of 0.2 at 10^{th} and 30^{th} epoch. As done in Paulus et al. [2021], we initialize $\mathbf{a}_i$ uniformly between $[-0.5, 0.5]$. Rest of the hyperparmeters are same as those used for sudoku.

Baselines: We compare against a strong neural baseline which is same as the backbone model parameterizing the k^2 dimensional cost vector in [Rolínek et al., 2020b]. It is trained by minimizing BCE loss between negative learnt cost ($-\mathbf{c}$) and target $\mathbf{y}^*$. We create an additional baseline by doing constrained inference with ground truth constraints and learnt cost ('Neural + CI') in table 6.5.

Results: Table 6.5 presents the percentage of keypoints matched correctly by different models. In this experiment, the bottleneck *w.r.t.* time is the backbone neural model instead of the ILP solver. Therefore, we also experiment with solver based negatives in our method (ILP–Loss + Solver). While ILP–Loss using solver-free negatives performs somewhat worse than CombOptNet in terms of its accuracy, especially for smaller problem sizes, using solver based negatives helps ILP–Loss surpass CombOptNet in terms of both the accuracy and training efficiency for large problem sizes (and makes it comparable on others). This is because a solver based negative sample is guaranteed to be incorrectly classified (as positive) by the current hyperplanes, and hence provides a very strong

Table 6.5: Average point-wise accuracy and training times over 3 runs with different random seeds for varying # of keypoints. Neural + CI denotes ILP inference with known constraints over the cost learnt by neural model. See section D.1 for std. err. over the 3 runs.

	Pointwise Accuracy (in %)				Training Times (in min)			
	4	5	6	7	4	5	6	7
Neural	80.88	78.04	75.39	73.49	148	**37**	**3**	**40**
Neural + CI	82.42	79.99	77.64	75.88	148	**37**	**30**	**40**
CombOptNet	83.86	**81.43**	78.88	76.85	**41**	67	144	279
ILP–Loss (Ours)	81.76	79.59	77.84	76.18	115	92	106	109
ILP–Loss + Solver (Ours)	**84.64**	81.27	**79.51**	**78.59**	43	73	99	174

training signal compared to solver-free negatives in ILP–Loss. We obtain a gain of up to 5 points over the Neural baseline, which is roughly double the gain obtained by Neural + CI.

6.5 Discussion

Training of Neural-ILP-Neural Architecture: In this work, we presented a formulation to train Neural-ILP architectures. However, our formulation is not capable of training *neural-ILP-neural* architectures where the output of ILP is an input to a downstream neural model. In the current formulation, availability of the solution to the ground truth ILP is important for our solver-free approach to work. Specifically, it is required to: 1.) convert the constrained optimization problem to a constraint satisfaction problem by including the cost-constraint eq. (6.4), and 2.) to calculate the positive loss eq. (6.6). However, in a Neural-ILP-Neural architecture, the intermediate supervision for only the Neural-ILP part (*i.e.*, solution of the ground truth ILP) is not available.

On the other hand, solver based methods such as CombOptNet, do not require access to the solution of the ground-truth ILP. Instead, they rely on the solution of the current intermediate ILP (during learning) to compute the gradients of the loss and thus their approach is not solver free. We note that even though in princple they can train Neural-ILP-Neural architectures, their experiments are only in the Neural-ILP settings.

Extending our current work for Neural-ILP-Neural architectures is an important direction of future work. One plausible approach could be to train an auxiliary inverse network that converts a given output of Neural-ILP-Neural architecture to a predicted symbolic target of Neural-ILP component. This predicted target can be used as a proxy to the ground truth solution of the Neural-ILP part. Similar ideas of using an inverse

network have been explored in [Agarwal et al., 2021], albeit under the setting where ILP is known and only the neural encoder and decoder needs to be learnt.

6.6 Conclusion

We have presented a solver–free framework for scalable training of a neural ILP architecture. Our method learns the linear constraints by viewing them as linear binary classifiers, separating the positive points (inside the polyhedron) from the negative points (outside the polyhedron). While given ground truth acts as positives, we propose multiple strategies for sampling negatives. A simple trick using the available ground truth outputs in the training data, converts the cost vector into a constraint, enabling us to learn the cost vector and constraints in a similar fashion.

Till now, we have looked at training purely neural and neuro-symbolic methods for solving combinatorial problems, which can also be seen as instances of a constraint satisfaction problem with unknown constraints. In this chapter, we expressed constraints explicitly as linear inequalities over the output variables.

In the next work, we take a completely different view of constraints as explanations instead of constraints as the underlying rules of a reasoning problem. Specifically, we generate explicit horn-clauses as explanations for predictions of a black-box Knowledge Base Completion model.

Chapter 7

OxKBC: Outcome Explanation for Knowledge Base Completion

Till now, we have focused on learning with known and unknown constraints. In the last chapter, we developed methods to differentiate through explicit but unknown linear constraints over the output variables. In this chapter, we train a model to output explicit constraints as horn clauses over the variables in a transductive Tensor Factorization based Knowledge Base Completion (KBC) model, such as DistMult [Yang et al., 2015], ComplEx [Trouillon et al., 2016]. We view these horn clauses as an explanation for the predictions made by the KBC model.

State-of-the-art models for KBC for large KBs (such as FB15k and YAGO) are based on tensor factorization (TF), *e.g.*, DistMult, ComplEx. While they produce good results, they cannot expose any rationale behind their predictions, potentially reducing the trust of a user in the outcome of the model. Previous works have explored creating an inherently explainable model, e.g. Neural Theorem Proving (NTP), DeepPath, MINERVA, but explainability in them comes at the cost of performance. Others have tried to create an auxiliary explainable model having high fidelity with the underlying TF model, but unfortunately, they do not scale well to large KBs. In this work, we propose OxKBC – an **O**utcome e**X**planation engine for **KBC**, which provides a post-hoc explanation for every triple inferred by a (uninterpretable) factorization based model. It first augments the underlying Knowledge Graph by introducing weighted edges between entities based on their similarity given by the underlying model. It then defines a notion of human-understandable explanation paths along with a language to generate them. Depending on the edges, the paths are aggregated into second–order templates for further selection. The best template with its grounding is then selected by a neural selection module that is trained with minimal supervision by a novel loss function. Experiments over Mechanical

Turk demonstrate that users overwhelmingly find our explanations more trustworthy compared to rule mining.

7.1 Introduction

Knowledge bases (KBs) store facts in the form of triples $r(s, o)$, which indicate that the head entity s is related to the tail entity o via the relation r, *e.g.*: $capitalOf(WashingtonDC, USA)$. These KBs are typically incomplete [Bollacker et al., 2008] and the task of KB Completion (KBC) aims to infer new facts from a given KB.

Many state-of-the-art models for KBC on large-scale KBs use tensor factorization (TF) [Sedghi and Sabharwal, 2018, Lacroix et al., 2018, Jain et al., 2018, Xue et al., 2018]. They learn embeddings for each entity and relation in a continuous vector space and define a scoring function to evaluate the validity of a given triple. At test time, given a query $r(s, ?)$ they output a ranked list of entities o by sorting the model scores for $r(s, o)$ for all possible tail entities. Though TF models achieve excellent task performance, they are inherently opaque – it is very difficult to uncover a rationale for why the model scores an entity high for a query, because neither the dimensions nor the scoring function are human interpretable. Since users are often the final consumers of a KB (e.g., via a KB-based QA task, or an entity retrieval task [Xiao et al., 2016]), we believe providing a rationale behind a prediction to them is an important sub-task for building trust in these systems.

Contributions: We present OxKBC: a post-hoc **O**utcome **eX**planation engine, which is faithful to a given TF-based **KBC** model (M). It works on the underlying Knowledge Graph, KG, corresponding to the facts given in the Knowledge Base, KB. It first augments the KG with weighted edges between entities based on their similarity as defined by the underlying model M. It then posits that an explanation for a prediction o for query $r(s, ?)$ is a path between s and o in this augmented graph.

A key contribution of our work is that similar paths are grouped into second-order templates and OxKBC trains a neural template selection module for explaining a given prediction. We define novel unsupervised and semi-supervised loss functions, so that this module can be trained with no or minimal training data. OxKBC outputs the highest scoring path within the selected template as an explanation to the user. We evaluate OxKBC explanations via a user study over Amazon Mechanical Turk. We find that workers overwhelmingly prefer OxKBC explanations compared to rule-mining techniques proposed earlier.

7.2 Related Work

KBC: Other than TF based models, research has also explored path based approaches for KBC. They use random walk with restarts [Lao and Cohen, 2010] and more recently, reinforcement learning [Das et al., 2018, Xiong et al., 2017] to construct multi-hop paths in the knowledge graph and use those to make predictions. The paths may offer natural interpretability [Nickel et al., 2016], but the performance of these models is significantly lower than TF, especially on large KBs like FB15K [Bordes et al., 2013], and YAGO [Suchanek et al., 2007]. See Section E.1.1 for a detailed comparison. [Rocktäschel and Riedel, 2017] inductively learn first order rules to generate paths by using a differentiable version of backward chaining. Unfortunately, it does not scale beyond 10K facts in KB.[1] In contamporaneous work, [Minervini et al., 2020] speed up neural theorem proving for this task. Unfortunately, we could not get their method to complete even after a week of continuous computation on FB15K, suggesting that their method still is not scalable to large KBs. Earlier, research had also considered symbolic Horn-clause style inference [Lao and Cohen, 2010, Schoenmackers et al., 2010], though these are no longer competitive with modern neural methods.

TF methods, like DistMult [Yang et al., 2015] and ComplEx [Trouillon et al., 2016], calculate the score $S^M(s, r, o)$, of a triple $r(s, o)$ via a latent tensor factorization over entity embeddings $(\overline{s}, \overline{t})$ and relation embeddings $(\overline{r})$. Key idea in these methods is to learn embeddings such that the score S^M of the facts present in the KB is high. In this case, DistMult [Yang et al., 2015] aligns relation embeddings to the Hadamard product of the embeddings of head and tail entitites. It uses $S^M(s, r, o) = \overline{r}^T(\overline{s} \bullet \overline{o})$, where $\bullet$ is the element-wise product, also known as Hadamard product.

While in principle OxKBC may be adapted to explain any TF model's predictions, in our experiments, we use TypedDistMult [2] [Jain et al., 2018] as our underlying model M. The key idea is to augment the scoring function used by DistMult to reduce type errors. Two new terms are introduced to account for head and tail entity type compatibility. The approach requires no explicit specification of type hierarchies or type supervision. The scoring function S^M scores $r(s, o)$ as $S^M(s, r, o) = \sigma(\overline{r}_w^T(\overline{s}_w \bullet \overline{o}_w))\sigma(\overline{r}_{ts}^T\overline{s}_t)\sigma(\overline{r}_{to}^T\overline{o}_t)$, where σ is the sigmoid function. The subscript w denotes the world knowledge embedding, t denotes the type embedding of an entity and ts and to denote head and tail type embeddings of a relation, respectively.

Explainability in KBC: There are broadly two approaches for explainable AI. One is to design a transparent model, which can inherently reveal its functioning (e.g., linear

[1] https://www.akbc.ws/2017/slides/sebastian-riedel-slides.pdf
[2] Code taken from https://github.com/dair-iitd/KBI

models, decision trees, Horn-clause inference for KBC). Unfortunately, transparent models often result in compromised performance in general [Weld and Bansal, 2018], as well as specifically for KBC (as discussed earlier in this section). An alternative is to provide post-hoc explanations for the behaviour of a high-quality black box model [Wachter et al., 2017]. This approach is further divided into two categories: 'Model Explanation' (Mx) and 'Outcome Explanation' (Ox) [Costabello et al., 2019].

In Mx, once the black–box model is learnt, a global explainable auxiliary model is built using the predictions of the black-box model such that the auxiliary model has high fidelity with the original black-box model. *Rule mining* [Yang et al., 2015] is an Mx approach, which mines first order rules based on embeddings learnt by M. It identifies paths (of length ≤ 3) using the notion of support and confidence borrowed from the frequent itemset mining literature [Pei et al., 2000]. Search space is pruned by considering only those relations which are close to each other in the embedding space. The extracted first order rules may be used to explain a prediction. We compare against this approach in our experiments. The other Mx approach is known as the *pedagogical approach* – it trains a global auxiliary interpretable model (weighted Horn clause rules) from the output of the non-interpretable but accurate TF model [Gusmão et al., 2018]. Its experiments were on a toy dataset with only 13 relations, and unfortunately, we were unable to scale this to the size of our dataset (see Section E.1.2).

Ox, attempts to achieve a higher fidelity between explainable model and M by creating local model approximations around each prediction independently, e.g. [Ribeiro et al., 2016]. This approach often trains a new classifier for each prediction, which is slow if a large number of predictions need to be explained. In response, OxKBC takes a novel Ox approach, which decomposes the explanation engine into two components: a global trained model that, for a given prediction, selects the most plausible second-order explanation template (which quantifies over both relations and entities), and an explanation generator that outputs the best instantiation of the template for this prediction. This approach scales better, since the trained model is global and easy to train, and no retraining is required per prediction.

Finally, a related task to Ox is *fact checking* – verifying the truthfulness of a (predicted) triple $r(s, o)$ based on existing facts in the KB. Many fact-checking approaches [Shiralkar et al., 2017, Shi and Weninger, 2016] use path based techniques to check the veracity of a triple and generate paths as evidence. However, these methods do not use algorithms that use information from the underlying TF model. Thus, while these paths may be useful evidence, they should not be used as explanations, as they are not faithful to the TF model. In the future, it may be interesting to combine fact checking ideas with statistical information in TF models to develop a better outcome explanation engine.

7.3 OxKBC: The Outcome Explanation Engine

We are given a knowledge base of facts, KB, with relations R and entities E, and a model M, trained on KB. M scores each triple $r(s, o)$ using the score $S^M(s, r, o)$. The goal of OxKBC is to show an end-user a reason why M predicts a tail entity o for the query $r(s, ?)$. It does so by finding an explanation path between s and o which entails the given relation r.

7.3.1 Language for generating Explanation Paths

A KB can be seen as a knowledge-graph KG, where there is a directed edge from s to o labeled by r for every KB fact $r(s, o)$. We now define the notion of an explanation path in this KG. Once the model M is learnt, we augment KG with additional edges that correspond to similarity between any two entities e and e'. We represent such an edge as $\sim_M (e, e')$ and call them 'entity similarity edges'. Here, the operator $\sim_M$ represents that M considers these two entities similar. We call the original edges of KG as 'relation edges'. An explanation path, $P(s, o)$, is a path between two entities, s and o, in the augmented KG. Here, we call s as $Head(P)$ and o as $Tail(P)$. A path is a sequence of edges concatenated by a ",". Following grammar generates an explanation path recursively:

1. **Relation Edges:** $P \leftarrow r(s, o) \quad \forall\, r(s, o) \in KB$
2. **Pre-fix a similarity edge:** $P \leftarrow \sim_M (s', s)\ , P_1\ s.t.\ Head(P_1) = s.$
3. **Post-fix a similarity edge:** $P \leftarrow P_1, \sim_M (o, o')\ \ s.t.\ Tail(P_1) = o.$
4. **Concatenation of two paths:** $P \leftarrow P_1, P_2 \quad s.t. \quad Tail(P_1) = Head(P_2).$

P_1 and P_2 on the R.H.S. themselves denote explanation paths generated using the same rules recursively. An explanation path may contain both types of edges: 'relation edges' and 'entity similarity edges'. For each entity similarity edge between two entities e and e', we may define its weight as $sim_M(e, e')$ where sim_M function captures the degree of similarity between the entities as given by the underlying model M. This is easy to obtain from any TF model, e.g., for TypedDistMult [Jain et al., 2018], we use $cos(\overline{r}_w, \overline{r}'_w)cos(\overline{r}_{ts}, \overline{r}'_{ts})cos(\overline{r}_{to}, \overline{r}'_{to})$, where cos is the cosine similarity. For a relation edge, we consider its weight as 1 as we include relation edges for only those triples that exist in KB. Another alternative is to use the model score $S^M(s, r, o)$ here as well, like similarity edges, and that would make all our edges probabilistic. But we leave that for future work. Using this notion of edge weights, we define $EdgeScore(P)$ of a path P as product of all the edge weights in it. For P to be a good explanation, its $EdgeScore$ should be high. Next, we define an operator, $RelComposition(P)$, which performs a composition of all the relations in the path P. It returns a vector in the same space as the relation embeddings

learnt by the model. We follow previous work to use a Hadamard product of the embeddings of all relations in the path [Guu et al., 2015]. For example, for $P(s,o) = r_1(s,u_1), r_2(u_1,o)$, $RelComposition(P) = r_1 \bullet r_2$.

Now, let the model M predict an entity o for the query $r(s,?)$. To quantify the plausibility of an explanation path $P(s,o)$ for the prediction $r(s,o)$, we define a plausibility score as:

$$Plausibility(P(s,o), r(s,o)) = sim_M\left(RelComposition\left(P\left(s,o\right)\right), r\right) \cdot EdgeScore\left(P\right) \tag{7.1}$$

In the above equation, $EdgeScore(P)$ captures all the entity similarities in the path P and $sim_M(RelComposition(P), r)$ captures whether composition of relations in the path entail the given relation r or not. sim_M function is as defined earlier.

Task of OxKBC is to select the most plausible explanation path. However, one can easily observe that $Plausibility$ score might not be comparable for two paths of unequal lengths. Even for two equal length paths, it could be difficult to compare due to different edge types. For example, consider two paths P_1 and P_2 with length two but different edge types: $P_1(s,o) = \sim_M (s,u_1),\ r_1(u_1,o)$ and $P_2(s,o) = r_1(s,v_1),\ r_2(v_1,o)$. P_1 has a similarity edge followed by a relation edge and P_2 has two relation edges. Clearly $Plausibility(P_1, r(s,o)) = sim_M(s,u_1) \cdot sim_M(r_1,r)$ and $Plausibility(P_2, s, r(s,o)) = sim_M(RelComposition(r_1,r_2), r)$ are not comparable as former involves product of two similarities whereas the latter has only one term. To resolve such discrepancies, we define the notion of a second–order template which aggregates comparable explanation paths. Once we have defined a template, then OxKBC would be left with the job of selecting the most appropriate template and ground it to generate an explanation.

7.3.2 Templates: Aggregation of similar Explanation Paths

Depending on the sequence of edge types in it, we classify an explanation path into a second–order template in which we quantify over both relations and entities. Each template corresponds to a fixed sequence of edge types. This aggregation is required because the entire space of explanation paths is huge and it is not feasible to compare all of them. Hence, to operationalize the selection process, we aggregate them into templates. We use these templates to explain the prediction o for a query $r(s,?)$:

T1. **Relation Similarity**: $r'(s, o) \in KB$ and $r \sim_M r'$
T2. **Entity Similarity**: $r(s', o) \in KB$ and $s \sim_M s'$
T3. **Entity & Relation Similarity**: $s \sim_M s'$ and $r'(s', o) \in KB$ and $r \sim_M r'$, $r \neq r'$
T4. **Two Length Relation Similarity**: $r_1(s, u_1) \in KB$ and $r_2(u_1, o) \in KB$ and $r \sim_M RelComposition(r_1, r_2)$

Preliminary investigation of potential explanations for our datasets reveal that these templates are able to explain most of the predictions. Our work naturally extends to other templates, such as "Entity & two-length Relation Similarity", but we do not implement them for our datasets.

We note that while other works have considered rules based on relation similarity earlier, e.g. [Yang et al., 2015] mine first order one length and two length rules (T1 and T4 respectively), we are the first one to use the notion of entity similarity for generating explanations. Intuitively, entity similarity captures an aggregation of many 2 length paths: a high similarity score between e and e' implies that they can be used interchangeably, i.e. $\exists r \in R$ and $u \in E$ *s.t.* both $r(e, u), r(e', u) \in KB$, i.e. $P(e, e') = r(e, u), s(u, e')$ is a path in KB if $s = r^{-1}$ is also in the set of relations R, and this holds for many choices of r and u, implying existence of many such paths. While rule mining may discover just one path between e and e', entity similarity may hop directly using an entity similarity edge, e.g. for second fact in Table 7.4.

We also note that since our templates are second order, they have the representation power to subsume first order rules. For example, T1, which infers that $r(s, o)$ because $r'(s, o) \in KB$ and "r is similar to r'" subsumes one length rule "$r' \implies r$" as learnt in many other rule mining systems.

While annotating the data, we observed that in few cases o is a prediction for a given query $r(s, ?)$ because in the KB it is the most frequent entity for the relation r. For example, most of the websites (s) in the KB are in 'Language' (r) 'English' (o), and hence model learns to predicts 'English' whenever asked for 'Language' of any website. To account for such explanations, we consider a frequency based template as well:

T5. **Frequent for Relation**: o is a highly frequent tail entity for the relation r in KB

These templates may not be able to explain every prediction, hence, we also introduce a default template (T0) that corresponds to *'no explanation'*. Table 7.4 lists sample explanations (generated by OxKBC) on FB15K test data along with template IDs.

Template Scoring: The *Compatibility* function between a path P and a prediction $r(s, o)$ as defined in equation 7.1 offers a natural way of scoring an explanation path P

for the given prediction. Since a given similarity based template represents comparable paths with same sequence of edge types in it, we can quantify it with the explanation path having maximum score. Accordingly, we define template score, T_i^S, for a similarity based template Ti as:

$$T_i^S(s,r,o) \;=\; \max_{P \in Ti} Compatibility(P, r(s,o)) \quad \forall i \in \{1,\ldots,4\} \tag{7.2}$$

Inspired by the notion of selecting the best amongst all paths in a given similarity based template, we define score of frequency based templates T5 as the normalized frequency of the corresponding relation, tail entity pair (r,o) in KB:

$$T_5^S(s,r,o) \;=\; \frac{|\{s' : r(s',o) \in KB\}|}{|\{(s',o') : s', r(s',o') \in KB\}|} \tag{7.3}$$

These goodness scores act as important features in the template selection module (section 7.3.3). Once a template is selected, an explanation in English is produced as shown in Table 7.4. For templates T1-T4, we ground them with the entities/relations in the explanation path corresponding to the *argmax* of the *Compatibility* score.

Faithfulness of Explanation Paths to the TF Model: Multiplicative models like TypeDistMult use three-way products to compute scores of a triple. So, a high dot product (and therefore, cosine score) between two relation embeddings r and r' (or two entity embeddings) fundamentally represents that model M considers the two relations (entities) as somewhat replaceable. This idea is exploited in our similarity templates. Consider, as an example, a query $r(s,?)$ and the corresponding prediction, o, for the tail entity by the model M. Say OxKBC explains the prediction using T1 by finding the best fact $r'(s,o)$ in the KB such that $r \sim_M r'$ (cosine score between the two is high). Now, since $r'(s,o)$ is in the KB, model score for $r'(s,o)$ must be trained to be high, and hence, Hadamard product of the embeddings of s and o is likely to be closely aligned with the embedding of r'. Because $r \sim_M r'$ Hadamard product of the embeddings of s and o is also closely aligned with that of r, resulting in a high model score for $r(s,o)$. Thus, this explanation is one reasonable clue why model M may have decided to make this prediction in the first place. Similar arguments can be made for T2 T3 and T4.

7.3.3 Selection Module

OxKBC's selection module (SM) decides which template to select for explaining a given prediction. It uses a 2 layer MLP for this task. For each template i, it takes an input feature vector and outputs a score S_i^{SM}, representing SM's belief that this template is a good explanation for the prediction. The scores S_i^{SM} are converted into probabilities P_i^{SM} through a $softmax$ layer. OxKBC chooses the template with the highest probability.

Table 7.1: Input Features for a prediction $r(s, o)$ from i^{th} template Ti

Global Features for a query $r(s, ?)$		**Prediction specific features for** $r(s, o)$	
max score	$\max_{u \in E} T_i^S(s, r, u)$	score	$T_i^S(s, r, o)$
distr. mean	$\text{mean}_{u \in E} T_i^S(s, r, u)$	similarity	$sim_M\left(o, \arg\max_{u \in E} T_i^S(s, r, u)\right)$
distr. std	$\text{std}_{u \in E} T_i^S(s, r, u)$	rank	$\text{rank}_{\text{o}}{}_{u \in E} T_i^S(s, r, u)$

Input Features: For a given query $r(s, ?)$ and a template i, we compute template goodness scores $T_i^S(s, r, u)\ \forall\ u \in E$. This defines a distribution of goodness scores over tail entities for a given input query $r(s, ?)$. To explain a prediction $r(s, o)$, we construct a feature vector such that it captures the relative score $T_i^S(s, r, o)$ of o w.r.t the distribution $T_i^S(s, r, u)$ over $u \in E$. We claim that this feature vector has all the information to decide if this template is good for explaining the given prediction or not. It maintains the distribution level global features w.r.t. the query $r(s, ?)$: max, mean, std deviation of the distribution, and prediction specific features: score, rank and other statistics of the specific prediction o. Table 7.1 enumerates all the template features used as input to the Selection Module. Since we do not have goodness scores for T0, SM considers its features as trainable parameters.

Training: Manual labeling of explanations is time-consuming, so we primarily train SM in unsupervised[3] and semi-supervised settings. For completeness, we also report on a fully supervised setting on our small annotated dataset. For the unsupervised setting, the key insight is that for most triples known to be true, we may expect some explanation, but we do not know which template to select. However, for a randomly selected negative sample, the selection module must always select T0 as we do not expect any explanation for it. In the absence of template level annotation, we rely on distant supervision from this weak signal. SM formulates it in a multi-instance learning (MIL) framework [Dietterich et al., 1997] where feature vector for each template is an instance, and supervision is at the bag level (positive or negative labels on a triple). It uses a two layer MLP (with 90 and 40 neurons in two hidden layers) to compute a score for each template Ti. A softmax over the scores of all the templates converts them into a probability distribution over the space of templates, and the Selection Module selects the one with maximum probability as the best explanation template. Figure 7.1 shows the overall architecture.

Loss Function: We operationalize this through an unsupervised loss function. When classifying a positive triple, SM gains reward (R_{pos}) proportional to the highest template probability amongst templates other than T0. To account for the case that sometimes

[3]We use a very small set of 20 labeled samples as dev set for hyperparameter tuning.

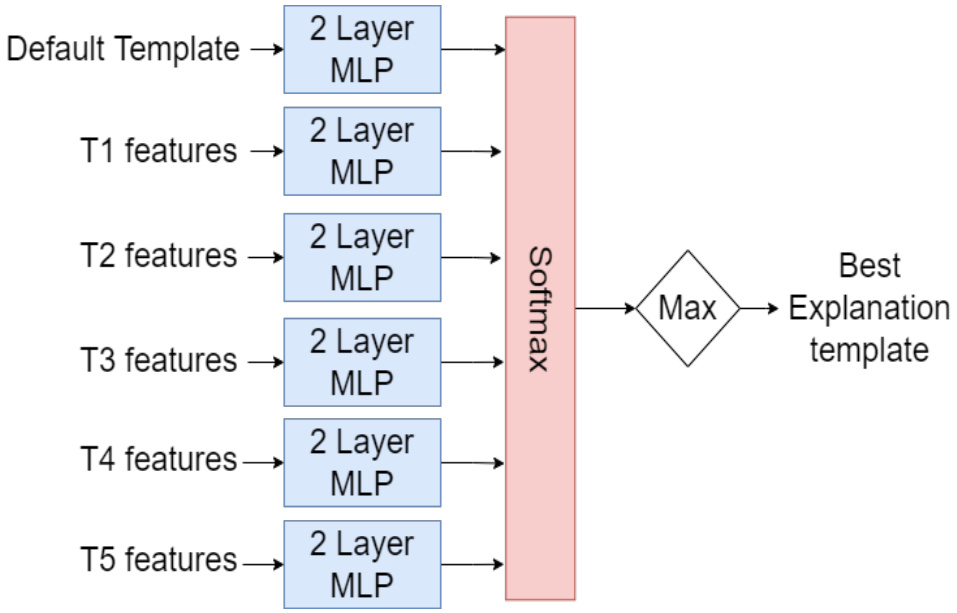


Figure 7.1: Architecture of our Selection Module. The underlying two-layer MLP is same when predicting score for every template.

no template might be available even for a correct triple, SM gets a lower reward even for selecting T0 (eq. (7.4)). For negative samples, the module is penalized (R_{neg}) if it selects any template other than T0 – it gets a negative reward proportional to total probability of all the explanation templates other than T0, and positive reward proportional to probability of T0 (eq. (7.5)).

$$R_{pos} = \rho_{pos} p_0^{\text{SM}} + \max_{i \neq 0} p_i^{\text{SM}} \quad (7.4)$$

$$R_{neg} = p_0^{\text{SM}} - \rho_{neg} \sum_{i \neq 0} p_i^{\text{SM}} \quad (7.5)$$

$$L_{\text{unsup}} = -R_{pos} - R_{neg} \quad (7.6)$$

$$L_{\text{semi}} = L_{\text{unsup}} + \lambda_1 L_{\text{sup}} + \lambda_2 L_{\text{PR}} \quad (7.7)$$

In semi-supervised setting, we assume access to a small train set of labeled explanations, i.e., the correct template for a fact is annotated. We add a supervised loss term (L_{sup}), which is the standard cross-entropy loss over the set of labeled examples. Additionally, we project the template label distribution from labeled set into unlabeled data through posterior regularization [Ganchev et al., 2010]. We add a KL divergence loss (L_{PR}) between labeled probability distribution over templates in train set and learned distribution. All hyperparameters ρ and λ are tuned on a small devset.

7.4 Experiments

The goal of our experiments is to answer the following research questions: (1) What is the benefit of adding a small amount of supervision on the overall performance of our model? (2) How good are our explanations compared to existing approaches? (3) Are our explanations faithful to the underlying TF model? To answer the first question, we perform an internal evaluation using the annotations made by the authors. To answer the second question, we perform a human evaluation using workers on Mechanical Turk. To answer the third question, we design an experiment to observe the impact of removing a fact from the KB which is part of an explanation on the quality of prediction made by the TF model. Next, we describe our experiments in detail.

Datasets: Our experiments use the standard KBC datasets, FB15K [Bordes et al., 2013] and YAGO3-10 [Dettmers et al., 2018]. FB15K is a subset of the original Freebase [Bollacker et al., 2008] and has $592,213$ facts with $14,951$ entities and $1,345$ relations. A large fraction of this knowledge base describes facts about actors, awards, movies and sports. YAGO3-10 is a subset of YAGO [Suchanek et al., 2007] and has $1,089,040$ facts with $123,128$ entities and 37 relations. Most of the facts describe attributes like citizenship, gender, profession, etc. for various people.

Models: We use TypedDistMult [Jain et al., 2018] as our underlying tensor factorization model M, since its code is freely available and it has near state-of-the-art performance on these datasets. To train it, we retain the exact train, dev and test folds used in previous work.

Training Data: For training SM, we randomly select 100 samples from the original dev set, where M makes correct prediction at the top. Two authors annotate each sample with the template producing the best explanation with inter-annotator agreement (Cohen's Kappa) of 0.87, and resolve conflicts through discussion. This labeling took about 3 hours. Out of 100 training samples, 20 samples is used as dev set for hyper parameter tuning. Remaining 80 samples are used as train set in semi-supervised and supervised setting to compute supervised loss (L_{sup}). The entire train data of the original model M is used for computing the unsupervised loss (L_{unsup}) in semi-supervised and unsupervised setting.

Templates: We evaluate only those templates via SM that are chosen at least once in our manual annotation. For FB15K, this includes T1, T2, T4 and T5 and for YAGO, only T1, T2 and T5.

7.4.1 Internal Evaluation

Similar to labelled training data, we annotate around 135 randomly sampled facts from the original test data with the template producing the best explanation. We note that in our internal evaluation, we use only those test queries where the true fact was scored at the top by our model since it is not quite clear how to evaluate the explanation of an incorrect tuple. We report micro-F1 for these templates over this unseen labelled test data as our evaluation metric. We repeat each experiment three times by changing random seeds and report the mean and standard deviation (See Section E.2 for details on hyperparameters, training time and computing infrastructure).

Table 7.2: Micro-F1 score of OxKBC (mean ± std deviation)

Dataset	Unsupervised	Supervised	Semi-supervised
FB15K	0.537 ± 0.012	0.421 ± 0.089	0.641 ± 0.011
YAGO	0.699 ± 0.384	0.965 ± 0.010	0.968 ± 0.004

Unsupervised training is done using only the weak supervision on the unlabeled train data (eq. (7.6), whereas, the supervised training is done using only 80 labeled data points. Individually, both may not achieve as high performance as our semi-supervised setting, which uses the entire training data as unlabeled data along with manually annotated 80 labeled data points. It also includes additional evidence using the KL divergence term that is computed over unlabeled data.

Table 7.2 reports the results from this evaluation. We observe that semi-supervised learning indeed outperforms both other settings. We also note that unsupervised setting can have a huge variance, such as for the YAGO dataset. This is not surprising, since it can be very difficult to train the model using only distant supervision in the form of unsupervised loss term. As a result, in a couple of random runs, it learns a degenerate model where all predictions collapse randomly to any one of the templates. Adding KL divergence term along with a small amount of labeled data (80 samples) in the overall loss term helps in maintaining the same posterior distribution over the output space and ensures that it does not learn a degenerate model, thus reducing variance for semi-supervised setting.

The scores on YAGO are particularly high, because there are very few relations for which M makes correct predictions at the top, making the role of OxKBC easy. Overall, our results are in line with our expectation that semi-supervised learning will outperform purely supervised or unsupervised learning, by exploiting statistical regularities present in large amounts of unlabeled data.

7.4.2 User Evaluation

Since explanations are ultimately for end users, we compare the quality of our explanations over other proposed approaches, by a user study over Amazon Mechanical Turk[4] on FB15K dataset.

Baseline: For implementing rule mining baseline, we adapt the procedure as defined in [Yang et al., 2015] to mine closed-path logical rules upto 3 length and use embeddings trained by M to prune the exponential search space. The rules are ordered by confidence, defined as the ratio of number of correct predictions using the rule and the total number of predictions possible using body of the rule. Refer to section 5 in [Yang et al., 2015] for more details. In the first version of our experiments, we noticed that for many predictions, rules did not provide any explanation, which led to workers preferring OxKBC almost always over rule mining. To fix this, we reduced the thresholds, but that resulted in selection of longer rules much more frequently because of their rarity (making their confidence high). Experiments with this baseline resulted in workers preferring OxKBC over rules 84% of the time. But for a given prediction, we notice that a rule of length 1 is almost always more convincing than a rule of length 2 and so on. Hence we modify the rule selection procedure to prefer a lower length rule whenever present. Within same length, rule with highest confidence in selected. We report our results using this strong version of the baseline.

7.4.2.1 Quality of Explanations

We ask workers to compare explanations generated by OxKBC with explanations derived from the baseline. Both the systems use the same underlying base model M to generate their explanations. We randomly select 450 different facts from the test data and generate explanations from the two explanation engines. Out of these, 97 have the exact same explanation from both engines, and 21 have no explanation from both engines. For the remaining, we ask five different workers to select the better explanation. They are also given an option to reject or accept both explanations. In cases, where one engine offers no explanation, we ask workers whether the other explanation is better or worse than not having an explanation. For explaining entity similarity, we also generate a context for why two entities are similar and the user can hover to see the facts where the entities are interchangeable to see the rationale for why M considers two entities similar. Finally, we use majority voting to decide which explanation is better for a fact.

We present the results in Table 7.3 and split the analysis into two parts: one when model's top prediction matches the gold annotation in the data, denoted as GT@1 (Gold Truth@1)

[4]**Selecting Workers**: See Section E.3 for details on how we select quality workers for our experiments.

Table 7.3: OxKBC vs. rule mining under different scenarios: **Overall:** when either of the two systems give an explanation; **Both:** When both give an explanation; **OxKBC:** When only OxKBC gives an explanation (explained above); Similarly for 'Rules'. When only one of the systems produces an explanation (last two rows), we ask workers whether it is better or worse than having no explanation. If it is selected as being worse, we mark the other system, producing 'no explanation', as better.

	OxKBC Better	**Rules Better**	**Tie**	**Total**
Overall	**145**	49	18	212
Both	**118**	40	18	176
OxKBC	**2**	2	0	4
Rules	25	**7**	0	32

(a) Comparison between explanations for GT@1

	OxKBC Better	**Rules Better**	**Tie**	**Total**
Overall	**94**	16	10	120
Both	**76**	10	10	96
OxKBC	**0**	3	0	3
Rules	18	**3**	0	21

(b) Comparison of explanations for N-GT@1

in Table 7.3a, and the other when it doesn't match the gold: N-GT@1 (Not–Gold Truth@1) in Table 7.3b. We see that not only are the explanations from OxKBC preferred over rule mining for GT@1, but even for the predictions for samples in N-GT@1, OxKBC gives a more probable explanation than rule mining. Note that the workers are never told if the given fact is true or not. Table 7.4 lists some facts, where OxKBC outperformed the explanations generated by rule mining and vice-versa.

7.4.2.2 Qualitative analysis of explanations for incorrect predictions

From Table 7.3b, we observe that workers prefer OxKBC's explanations even when the prediction of the model M is not marked as correct (N-GT@1). We randomly select 100 samples from N-GT@1 and manually analyze the explanations generated by OxKBC. For 77 out of 100 samples, OxKBC generated convincing explanations. On further investigation, we discovered that 52 out of 77 predictions were indeed correct in real-world but the model was penalized because the fact was not present in the KB or dev/test sets. This is an issue of the incompleteness of KB and demonstrates that good explanation can help in augmenting the KB with more facts (examples in Table 7.5). In the second case, we looked at remaining 25 odd examples where top model prediction was actually incorrect (in real-world). For 15 out of 25 samples, explanations looked reasonable even though the facts were not true. Table 7.6 lists few examples. While statistically a good prediction, in this particular case, the facts were untrue. We consider that these errors should not be used for model improvement, since the explanations are reasonably convincing even if the facts are untrue. The remaining 10 examples were the cases where the model made genuine errors (e.g., type related errors) and we did not get any proper explanation. These are the cases where we can potentially improve the model, e.g. by reranking until we get

Table 7.4: OxKBC was preferred to Rule Mining for first two facts, rule mining won in third fact, and none of the explanations were accepted in the last example.

Fact	Explanation from OxKBC	Explanation from Rule Mining
(Academy Award for Best Sound Mixing, has nomination for, WarGames)	[T4] (Academy Award for Best Sound Mixing, has nominee, Willie D. Burton) and (Willie D. Burton, was an award nominee for, WarGames)	(Academy Award for Best Sound Mixing, has nomination for, On Golden Pond) and (On Golden Pond, was nominated for, Academy Award for Best Cinematography) and (Academy Award for Best Cinematography, has nomination for, WarGames)
(The Last King of Scotland, has genre, Drama)	[T2] (The Lives of Others, has genre, Drama) and "The Last King of Scotland" is similar to "The Lives of Others"	(The Last King of Scotland, has actor, Forest Whitaker) and (Forest Whitaker, won an award for, Bird) and (Bird, has genre, Drama)
(47th Annual Grammy Awards, had an award category, Grammy Awards for Song of the Year)	[T2] (50th Annual Grammy Awards, had an award category, Grammy Awards for Song of the Year) and "47th Annual Grammy Awards" is similar to "50th Annual Grammy Awards"	(47th Annual Grammy Awards , is an instance of repeating event:, Grammy Awards) and (Grammy Awards, category , Grammy Awards for Song of the Year)
(Actor, is the profession of, John Lithgow)	[T2] (Musician, is the profession of, John Lithgow) and "Actor" is similar to "Musician"	(Actor, is the profession of, Henry Winkler) and (Henry Winkler, has profession, Writer) and (Writer, is the profession of, John Lithgow)

Table 7.5: OxKBC explanations from N-GT@1 when the top prediction is correct in real world.

Fact	Explanation from OxKBC
(Sound Mixer, had some job associated with film, The Motorcycle Diaries)	(Supervising Sound Editor, had some job associated with film, The Motorcycle Diaries) and ("Sound Mixer" is similar to "Supervising Sound Editor")
(Composer, is the profession of, Danny Elfman)	(Composer, is the profession of, John Kander) and (John Kander, is an award co-nominee with, Danny Elfman)
(2010 Winter Olympics, has a participating country, Czech Republic)	(2002 Winter Olympics, has a participating country, Czech Republic) and ("2010 Winter Olympics" is similar to "2002 Winter Olympics")

a plausible explanation. Doing this experimentation further as part of a controlled user study is a direction for future work.

Table 7.6: OxKBC explanations for samples from N-GT@1 when the predictions are actually incorrect.

Fact	Explanation from OxKBC
(Sex and the City, has actor, Fergie)	(Sex and the City, is an award nominating work of, Fergie)
(A Fish Called Wanda, was nominated for Academy Award for, Best Picture)	(A Fish Called Wanda, is an award nominating work of, Michael Shamberg) and (Michael Shamberg, was nominated for, Academy Award for Best Picture)
(Bobby McFerrin, music artist has genre, Rock music)	(Billy Joel, music artist has genre, Rock music) and ("Bobby McFerrin" is similar to "Billy Joel")

7.4.3 Faithfulness of Explanations

In these set of experiments, we set out to ask if the explanations generated by OxKBC are faithful to the underlying TF model or not? We note that explanations consist of supporting facts, e.g., for a prediction $r(s, o)$, if OxKBC produces an explanation using template T1, i.e., $r(s, o)$ *because* $r'(s, o)$ and $r' \sim r$, then $r'(s, o)$ is the supporting fact from KB used for grounding the template. We ask how important are the supporting facts in underlying TF model's decision making? Specifically, we observe the change in the quality of predictions caused by removing such supporting facts from the KB, compared to the change observed when a random fact connected with the prediction (control) is removed. Greater the resulting change, higher our confidence that the model used supporting facts as evidence for the prediction, and that our explanations are indeed faithful.

Table 7.7: MRR over 100 samples when the probability of the supporting fact in the explanation is reduced. Control MRR reports the MRR when probability of any other related fact in the KB is reduced.

	MRR			Control MRR		
Steps	**T1**	**T2**	**T4**	**T1**	**T2**	**T4**
1	1.00	1.00	1.00	1.00	1.00	1.00
1	0.50	0.60	0.40	0.94	0.95	0.96
2	0.30	0.34	0.24	0.88	0.89	0.89
5	0.20	0.18	0.13	0.69	0.57	0.63

To achieve the above objective, we perform the following experiment. For each template, we randomly select 100 correct predictions for which OxKBC outputs a grounding of that template as an explanation. We then take a few gradient steps to reduce the probability of the supporting fact(s) selected by OxKBC, i.e. $r'(s, o)$ in the above example, and compute the rank of the entity o for query $r(s, ?)$. As a control experiment, for each prediction $r(s, o)$, we randomly select a fact from KB changing either the subject s, object o or the relation r from the triple and take a few gradient steps to reduce the probability

of thus selected fact. We again compute the rank of the entity o for query $r(s, ?)$. We report the MRR for the two settings (non-control vs. control) over the 100 randomly selected predictions for each template in Table 7.7.

Our results show that there is a reduction in 'Control MRR' as well. This is expected as the facts selected in the control experiment are not completely random but in the neighborhood of the prediction and may have some impact on the model score for the prediction. However, as can be seen from the table, reducing the model's belief for the supporting fact results in a much sharper drop in its confidence of the correct prediction (as shown by sharp decrease in MRR vs control MRR). In other words, if the supporting facts were not correct, then model would not have ranked the object o high enough for the query $r(s, ?)$. This validates the hypothesis that explanations generated by OxKBC are indeed faithful to underlying TF model.

7.5 Discussion

OxKBC currently only outputs one best explanation: the top grounding of the selected template. To generate alternative explanations, it can always generate the next best groundings of the selected template. However, alternative good explanations may also come from a different template. This will require us to learn a Selection Module that can compare the second best grounding of a top template with the best grounding of the second best template (and so on). Given that our design of OxKBC assumes minimal available supervision, more thought is needed on how to compare these.

Sub–graph as an explanation: We also note that while explanations are often paths in a KG, sometimes they may be not, and we may need to generalize the notion of explanation as a "path" to explanation as a "sub–graph". We believe that our general approach may be extensible to subgraph explanations. We can extend the explanation grammar to allow for subgraph rules. We can then define new templates and provide a compatibility function for the template. The Selection Module should follow as is. For instance, $isSpouseOf(X, Y)$ because $isFatherOf(X, Z)$ and $isMotherOf(Y, Z)$ could be an instance of some template where $P \leftarrow P_1, P_2 \;\; s.t. \;\; Head(P_1) = Head(P), Head(P_2) = Tail(P)$ *and* $Tail(P_1) = Tail(P_2)$. We did not study this in depth because, in our preliminary analysis on our datasets, paths came out as natural explanations, and more generic subgraphs were not found essential. We leave these two directions for future research.

7.6 Conclusions

We present OxKBC, a novel approach for providing post-hoc explanations for tensor factorization based KBC models. We introduce the idea of entity similarity edges in the knowledge graph and use paths in the augmented knowledge graph as explanations. We quantify over both relations and entities to define second–order templates for aggregating explanation paths and train a neural selection module to pick the best template for a given prediction. The neural model can be trained in a minimally supervised fashion due to a novel loss function. Based on a user study done on mechanical turk, we find that explanations generated by our system are significantly better than those generated from rule mining. We perform additional experiments showing that the explanations generated by OxKBC are faithful to the predictions of the underlying model. In the future, we hope to extend the explanation engine for debugging the KBC model and improving its performance.

Part V

Epilogue

Chapter 8

Conclusion and Future Work

8.1 Conclusion and Summary

In this thesis, we focused on training deep models with symbolic constraints. We classified our work on the two axes of constraint representation (*explicit* vs. *implicit*) and knowledge of constraints (*known* vs. *unknown*). In most parts of the thesis, we considered constraints to be hard, except in the last chapter where constraints form plausible explanations but may not always hold.

We first discussed learning with *explicitly known* constraints, where we argued that in addition to the training data, domain knowledge could be a good source of information for training deep neural models. We observed that constraints may naturally represent the underlying domain knowledge in the form of a Boolean expression over the output space. We then presented a constraint language for discrete output space that converted a Boolean expression into an arithmetic expression involving the output probabilities over symbols. We then introduced a primal-dual formulation for enforcing constraints during training and demonstrated that learning with constraints is better than constrained inference.

Next, we focused on learning constraints automatically. Instead of using constraints to improve the performance of a neural model, we focused on constraints in a combinatorial problem where they form the core of a reasoning system. We formulated the issue of solution multiplicity and the lack of output space invariance in neural reasoners and proposed appropriate solutions to tackle them.

In the case of training under solution multiplicity, we first formally defined the task of 1oML and observed that when more than one solution exists for a training input, it may not be optimal to compute loss *w.r.t.* all the solutions. Instead, one should appropriately select one of the correct solutions for computing loss. We then showed that the greedy

method of selecting the appropriate target might get stuck in local minima and proposed exploration-based methods to skip local minima and improve the training. In particular, we proposed two exploration techniques: I-ExplR and SelectR. While I-ExplR relies on the model probabilities to decide the exploration, SelectR jointly trains a separate RL agent that outputs exploration probabilities.

Next, we observed that existing neural reasoners either suffer from a lack of output space invariance, *i.e.*, can't solve larger problems after training on smaller problems (RRN [Palm et al., 2018]), or, do not scale well (NLM [Dong et al., 2019]). In response, we proposed two scalable novel architectures based on RRN that can train on smaller puzzles and test on larger puzzles. Using our scalable architectures, we could train using smaller-sized sudokus and test with sudokus up to size 25×25, whereas NLM [Dong et al., 2019] could not run on even 9×9 sudokus. Our first architecture, binarized model, is inspired by the classical method of converting any CSP into a binary CSP and converts a multi-class node classification problem into a binary node classification problem. Our second architecture, multi-valued model, exploits the permutation invariance of the output labels and learns k_{max} embeddings from training instances with k labels. We concluded by comparing the strengths and weaknesses of our two models.

As a natural next step after studying neural reasoners, we worked towards integrating symbolic reasoners with neural models to create an end-to-end differentiable neuro–symbolic architecture. In particular, we focused on Integer Linear Programs (ILP) that express constraints *explicitly* as linear inequalities. Representing the constraints explicitly make them more amenable to interpretability. Moreover, once the correct constraints are learned, symbolic reasoners may generalize better than neural reasoners. We build on an existing method [Paulus et al., 2021] for learning constraints of an ILP in a Neural-ILP architecture. We observed that it calls the ILP solver in every learning iteration, making the training process very slow. In response, we proposed a solver-free framework for training. It looks at linear constraints as binary linear classifiers separating the given output from the rest of the space. We propose novel ways of sampling the negatives for training such binary linear classifiers. To learn the cost, we exploit the given output and convert it into a linear constraint encoding the fact that no other feasible data point can have a better cost than the given output.

In the last chapter, we took a completely different view of constraints as explanations for the predictions of a black-box neural model, specifically for the task of Knowledge Base Completion (KBC). In particular, we proposed OxKBC(Outcome Explanation Engine for KBC): a framework to generate explanations in the form of horn-clauses over the entities, relations, and known facts in the KB. OxKBC first augments the Knowledge Graph with entity similarity edges and then groups explanation paths into second-order templates. A

neural Selection Module selects the most appropriate template and grounds it to generate a horn-clause constraint as an explanation of a model prediction.

8.2 Future Work

In this thesis, we touched on various aspects of constraints in deep learning. A natural direction for future work would explore a combination of these different aspects. We discuss some of them below.

8.2.1 Combining Known and Unknown Constraints

In our work, we looked at both the scenarios of known and unknown constraints separately. Under known constraints, we mainly focused on problems where knowing only the constraints may not solve the task but may boost the performance of the underlying model. Usually, a domain expert would enumerate these constraints for us. On the other hand, under unknown constraints, we focused on combinatorial problems where constraints form the core of the problem and knowing them would actually solve the task. Is it possible to combine known and unknown constraints for both scenarios? We discuss each of them below.

In the first scenario where constraints are primarily used to improve the performance, Pan et al. [2020] already show that learning unknown constraints over the model outputs via Rectifier Networks [Pan and Srikumar, 2016] improve the performance for structured prediction tasks such as entity and relation extraction, and citation field extraction. However, they do not experiment under scenarios where a subset of constraints might already be known. Alternatively, it may happen that the enumeration of known constraints by the experts is not exhaustive and deploying methods to learn them explicitly may further improve the performance. For example, Tandon et al. [2018] use many commonsense constraints such as "*a non–existent entity can't move*" to solve procedural text comprehension task [Dalvi et al., 2018]. It is quite likely that certain constraints are still missing from their set of constraints, and deploying methods for learning constraints may further boost the performance.

In the second scenario, where constraints form the core of the reasoning problems, certain constraints might be already known for many combinatorial problems and providing them upfront while training may either boost the performance or enhance the learnability of the overall system. For example, in sudoku and futoshiki, the constraint that one cell can have only one digit is quite obvious and can be assumed to be known; in N-Queens, one may assume that total number of queens is N. It would be interesting to observe the

impact of the known constraints while training models for combinatorial problems.

8.2.2 Solution Multiplicity

Solution Multiplicity in Neural Execution of Graph Algorithms: In our work on solution multiplicity, we primarily focused on neural models that were trained to directly predict the solution of a combinatorial problem. However, for many of the combinatorial problems on graphs, a classical graph algorithm, such as Prim's shortest path algorithm, may provide intermediate supervision. In a recent work, Veličković et al. [2020] train GNNs to imitate the steps of an algorithm by aligning the embeddings of the graph nodes at an intermediate step k with the k^{th} step of the algorithm. Though they experiment with many graph structures, such as ladder graphs, two dimensional grid graphs, trees, but they randomly initialize the edge weights so that the k^{th} step is always uniquely defined. However, in a real–world scenario, this might not be the case. For example, many of the neighbouring edges can have the same weight and following any of them forms a valid step in Prim's algorithm. This falls under the scenario of solution multiplicity, where not only the final outputs are many, but there could be multiple paths leading to the same output. It will be interesting to address the issue of multiple paths while learning from intermediate supervision provided by the steps of an algorithm.

Solution Multiplicity while training Neural–ILP Architectures: In chapter 6, we developed a scalable method for training Neural–ILP architectures and proposed an appropriate loss function for it. Our current formulation of ILP–Loss crucially assumed a unique positive solution for each input. If the assumption of a unique solution for a given input is dropped, we may need to modify ILP–Loss. Some of the modifications are trivial, for example, all of the given solutions must satisfy all the constraints, and the expression for positive loss should account for it. However, sampling negatives now becomes difficult as we may sample one of the positives as a negative. It may become particularly challenging when all the positive solutions are not even available during training, resulting in learnability issues. Studying the impact of such a change on the learnability of the model is an exciting direction for future work.

Once the model has been trained, we observed in in chapter 4 that its aggregate performance deteriorates with an increase in the number of correct solutions. However, we hypothesize that a Neural–ILP architecture may not suffer from such degradation during inference. Irrespective of the number of possible correct solutions for an input, a symbolic solver in a Neural–ILP model will always output a correct solution once the correct constraints have been learnt. We leave the study of Neural–ILP architecture under solution multiplicity as a potential direction for future work.

8.2.3 Output Space Invariance

In our work, we used both CSP and ILP to formally represent a combinatorial problem. Both these representations are amenable to first–order lifting, and thereby can represent size–varying instances of the same puzzles using the same lifted expressions. For example, as we saw in section 5.3, sudoku of any size can be represented with a lifted first–order constraint $\texttt{Nbr}(c_1, c_2) \rightarrow \texttt{Neq}(\texttt{Value}(c_1), \texttt{Value}(c_2))$, where the predicate `Nbr` represents whether two cells are in the same row, column or box, and relation `Neq` represents whether two values are unequal or not. Similarly, one may construct lifted linear constraints to represent a generic sudoku. For example, if we represent a $N \times N$ sudoku as a 3D binary tensor $\mathbf{x} \in \{0,1\}^{N \times N \times N}$, then $\sum_{j=1}^{N} \mathbf{x}[i,j,k] = 1 \forall i,k \in \{1 \cdots N\}$ represent the constraint that each row i should have at most one occurrence of k.

In chapter 5, to design purely neural models that are agnostic to the size of the output space, we exploited permutation invariance of the output space and size invariance of GNNs. On the other hand, in chapter 6, we learnt fixed dimensional constraint matrix $\mathbf{A}$ and $\mathbf{b}$ that can represent puzzles of a fixed size. A natural question arises: is it possible to learn lifted linear constraints and parameterize the constraint matrix $\mathbf{A}$ and $\mathbf{b}$ based on the size of the problem? In chapter 6, we did not assume any knowledge of the underlying problem structure. Would knowing additional structures, such as notion of a row, column and a grid in sudoku, help in learning lifted ILPs? Would such a parameterization and knowledge of structure also result in more interpretable constraints? In the future, we would like to develop methods to represent and learn the linear constraints in their first–order formulation instead of a fixed dimensional constraint matrix or cost vector.

8.2.4 Neuro–symbolic Architectures

Human interpretable constraints in Neural–ILP models: While developing methods for training Neural–ILP architectures, we argued that the ground–truth linear constraints of an ILP are more interpretable than the weights of a black-box neural reasoner. However, we observed that the learned linear constraints are very different from the natural constraints that are assumed to be unknown. For example, there is no obvious direct correspondence between the natural linear one–hot constraints of sudoku as discussed in Bartrlett et al. [2008] and the constraints learnt by our system, beyond the fact that each constraint represents a hyper–plane in a high dimensional space. In the future, we would like to develop methods that distill the learned constraints into human–interpretable form or directly learn natural interpretable constraints. Assuming structure and learning lifted constraints, as discussed in the previous section, may also result in human–interpretable

constraints. We leave it as a promising direction for future work.

Neural–ILP–Neural tasks: Our scalable method of training Neural-ILP architectures relies heavily on the availability of symbolic output of the ILP layer. However, such symbolic output might not always be available, for example, in neural-symbolic-neural architectures where the symbolic output of an ILP layer is the input to a downstream neural model. Although CombOptNet proposed by Paulus et al. [2021] can be used to train neural-symbolic-neural architectures, it comes at the cost of scalability. Extending our scalable approach to the training of such neural-symbolic-neural architectures is a potential direction for future work. It would be interesting to find a complex real–world application for neural-symbolic-neural architectures.

Bibliography

Ananye Agarwal, Pradeep Shenoy, and Mausam. End-to-end neuro-symbolic architecture for image-to-image reasoning tasks. *CoRR*, abs/2106.03121, 2021. URL https://arxiv.org/abs/2106.03121.

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9558–9570, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9ce3c52fc54362e22053399d3181c638-Abstract.html.

Christian Agrell, Simen Eldevik, Andreas Hafver, Frank Børre Pedersen, Erik Stensrud, and Arne Bang Huseby. Pitfalls of machine learning for tail events in high risk environments. *Safety and Reliability – Safe Societies in a Changing World*, 2018.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019a. URL https://openreview.net/forum?id=BJxgz2R9t7.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. PDP: A general neural framework for learning constraint satisfaction solvers. *CoRR*, abs/1903.01969, 2019b. URL http://arxiv.org/abs/1903.01969.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017. URL http://proceedings.mlr.press/v70/amos17a.html.

Vaibhav Arcot and Samarth Kalluraya. Data set of 4x4 sudoku puzzles and solutions. https://github.com/Black-Phoenix/4x4-Sudoku-Dataset, 2019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Aniket Bajpai, Sankalp Garg, and Mausam. Transfer of deep reactive policies for MDP planning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10988–10998, 2018.

Andrew Bartrlett, Tim Chartier, Amy Langville, and Timothy Rankin. Integer programming model for the sudoku problem. *"J. Online Math. & Its Appl."*, 8(May 2008), May 2008. URL "https://www.maa.org/press/periodicals/loci/joma/integer-programming-model-for-the-sudoku-problem".

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9508–9519. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6bb56208f672af0dd65451f869fedfd9-Paper.pdf.

Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. Size-invariant graph representations for graph classification extrapolations. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pages 837–851, 2021. URL http://proceedings.mlr.press/v139/bevilacqua21a.html.

Esben Jannik Bjerrum. SMILES enumeration as data augmentation for neural network modeling of molecules. *CoRR*, abs/1703.07076, 2017. URL http://arxiv.org/abs/1703.07076.

Nikolay Bogoychev and Pinzhen Chen. Terminology-aware translation with constrained decoding and large language model prompting. In Philipp Koehn, Barry Haddon, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation, WMT 2023, Singapore, December 6-7, 2023*, pages 890–896. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.wmt-1.80.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376746. URL http://doi.acm.org/10.1145/1376616.1376746.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2787–2795, USA, 2013. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999792.2999923.

Antoine Bordes, Y-Lan Boureau, and Jason Weston. Learning end-to-end goal-oriented dialog. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=S1Bb3D5gg.

Andrea Borghesi, Federico Baldo, and Michela Milano. Improving deep learning models via constraint-based domain knowledge: a brief survey. *CoRR*, abs/2005.10691, 2020. URL https://arxiv.org/abs/2005.10691.

Johan Bos, Valerio Basile, Kilian Evang, Noortje Venhuizen, and Johannes Bjerva. The groningen meaning bank. In Nancy Ide and James Pustejovsky, editors, *Handbook of Linguistic Annotation*, volume 2, pages 463–496. Springer, 2017.

Luke Breitfeller, Aakanksha Naik, and Carolyn P. Rosé. STAGE: tool for automated extraction of semantic time cues to enrich neural temporal ordering models. *CoRR*, abs/2105.07314, 2021. URL https://arxiv.org/abs/2105.07314.

Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 73–82, 2010. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2098&proceeding_id=26.

Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1Xw62kRZ.

Vivien Cabannes, Alessandro Rudi, and Francis Bach. Structured prediction with partial labelling through the infimum loss. *CoRR*, abs/2003.00920, 2020. URL https://arxiv.org/abs/2003.00920.

Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL https://aclanthology.org/W05-0620.

Mario Lezcano Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3794–3803, 2019. URL http://proceedings.mlr.press/v97/lezcano-casado19a.html.

Timothy C. Y. Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications, 2021. URL https://arxiv.org/abs/2109.03920.

Kai-Wei Chang, Rajhans Samdani, and Dan Roth. A constrained latent variable model for coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 601–612, 2013. URL http://aclweb.org/anthology/D/D13/D13-1057.pdf.

Jianshu Chen and Li Deng. A primal-dual method for training recurrent neural networks constrained by the echo-state property. *arXiv preprint arXiv:1311.6091*, 2013.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4): 834–848, 2018. doi: 10.1109/TPAMI.2017.2699184. URL https://doi.org/10.1109/TPAMI.2017.2699184.

Nuri Cingillioglu and Alessandra Russo. Deeplogic: Towards end-to-end differentiable logical reasoning. In *Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019) Stanford University, Palo Alto, California, USA, March 25-27, 2019., Stanford University, Palo Alto, California, USA, March 25-27, 2019*, volume 2350 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL http://ceur-ws.org/Vol-2350/paper21.pdf.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 670–680. Association for Computational Linguistics, 2017. doi: 10.18653/v1/d17-1070. URL https://doi.org/10.18653/v1/d17-1070.

Luca Costabello, Freddy Lecue, Fosca Giannotti, Riccardo Guidotti, Pascal Hitzler, Pasquale Minervini, and Kamruzzaman Sarker. On explainable ai: From theory to motivation, applications and limitations, 2019. A tutorial at AAAI 2019.

Timothée Cour, Benjamin Sapp, and Ben Taskar. Learning from partial labels. *J. Mach. Learn. Res.*, 12:1501–1536, 2011. URL http://dl.acm.org/citation.cfm?id=2021049.

Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1144. URL https://www.aclweb.org/anthology/N18-1144.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2018.

Johan de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 290–296, 1989. URL http://ijcai.org/Proceedings/89-1/Papers/046.pdf.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1182–1191. PMLR, 2018. URL http://proceedings.mlr.press/v80/denton18a.html.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy I/O. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 990–998. PMLR, 2017. URL http://proceedings.mlr.press/v70/devlin17a.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. URL https://aclweb.org/anthology/papers/N/N19/N19-1423/.

Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell.*, 89(1-2):31–71, 1997.

Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artif. Intell.*, 244:143–165, 2017a. doi: 10.1016/j.artint.2015.08.011. URL https://doi.org/10.1016/j.artint.2015.08.011.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, pages 920–923, 2017b. doi: 10.1109/ICMLA.2017.00-37. URL https://doi.org/10.1109/ICMLA.2017.00-37.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=B1xY-hRctX.

Gregory Druck, Kuzman Ganchev, and João Graça. Rich prior knowledge in learning for nlp, 2011. URL https://www.aclweb.org/anthology/P11-5005.

Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic sat-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 1169–1177, 1997. URL http://ijcai.org/Proceedings/97-2/Papers/055.pdf.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018. doi: 10.1613/jair.5714. URL https://doi.org/10.1613/jair.5714.

Jun Feng, Minlie Huang, Li Zhao, Yang Yang, and Xiaoyan Zhu. Reinforcement learning for relation classification from noisy data. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5779–5786. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17151.

Lei Feng and Bo An. Partial label learning with self-guided retraining. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3542–3549. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33013542. URL https://doi.org/10.1609/aaai.v33i01.33013542.

Lei Feng, Jiaqi Lv, Bo Han, Miao Xu, Gang Niu, Xin Geng, Bo An, and Masashi Sugiyama. Provably consistent partial-label learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/7bd28f15a49d5e5848d6ec70e584e625-Abstract.html.

Aaron M. Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1504–1511. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5509.

Thomas Frerix, Matthias Nießner, and Daniel Cremers. Linear inequality constraints for neural network activations. *CoRR*, abs/1902.01785, 2019. URL http://arxiv.org/abs/1902.01785.

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1300–1309, 1999. URL http://ijcai.org/Proceedings/99-2/Papers/090.pdf.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 413–422. International World Wide Web Conferences Steering Committee / ACM, 2013. doi: 10.1145/2488388.2488425. URL https://doi.org/10.1145/2488388.2488425.

Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md. Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. Bias and fairness in large language models: A survey. *CoRR*, abs/2309.00770, 2023. doi: 10.48550/ARXIV.2309.00770. URL https://doi.org/10.48550/arXiv.2309.00770.

Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, 2010. URL http://portal.acm.org/citation.cfm?id=1859918.

Sankalp Garg, Aniket Bajpai, and Mausam. Symbolic network: Generalized neural policies for relational mdps. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3397–3407. PMLR, 2020.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 10932–10952. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.674.

Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs *and* an algorithm for the mixed integer problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 77–103. Springer, 2010. doi: 10.1007/978-3-540-68279-0_4. URL https://doi.org/10.1007/978-3-540-68279-0_4.

Ke Gong, Xiaodan Liang, Dongyu Zhang, Xiaohui Shen, and Liang Lin. Look into person: Self-supervised structure-sensitive learning and a new benchmark for human parsing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6757–6765. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.715. URL https://doi.org/10.1109/CVPR.2017.715.

Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *CoRR*, abs/1909.04866, 2019. URL http://arxiv.org/abs/1909.04866.

Rick Groenendijk, Sezer Karaoglu, Theo Gevers, and Thomas Mensink. Multi-loss weighting with coefficient of variations. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pages 1468–1477. IEEE, 2021. doi: 10.1109/WACV48630.2021.00151. URL https://doi.org/10.1109/WACV48630.2021.00151.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 192–202, 2016. URL http://aclweb.org/anthology/D/D16/D16-1019.pdf.

M.M. Gupta and J. Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy Sets and Systems*, 40(3):431–450, 1991. ISSN 0165-0114. doi: https://doi.org/10.1016/0165-0114(91)90171-L. URL https://www.sciencedirect.com/science/article/pii/016501149190171L. Fuzzy Logic and Uncertainty Modelling.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

Arthur Colombini Gusmão, Alvaro Henrique Chaim Correia, Glauber De Bona, and Fábio Gagliardi Cozman. Interpreting embedding models of knowledge bases: A pedagogical approach. In *Proceedings of the ICML Workshop on Human Interpretability in Machine Learning*, 2018.

Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 318–327, 2015.

Rujun Han, Qiang Ning, and Nanyun Peng. Joint event and temporal relation extraction with shared representations and structured prediction. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 434–444. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1041. URL https://doi.org/10.18653/v1/D19-1041.

Aria HasanzadeZonuzy, Archana Bura, Dileep M. Kalathil, and Srinivas Shakkottai. Learning with safety constraints: Sample complexity of reinforcement learning for constrained mdps. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7667–7674. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16937.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Mikael Henaff, Junbo Jake Zhao, and Yann LeCun. Prediction under uncertainty with error-encoding networks. *CoRR*, abs/1711.04994, 2017. URL http://arxiv.org/abs/1711.04994.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14

(8):2, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Alfred Horn. On sentences which are true of direct unions of algebras. *J. Symb. Log.*, 16 (1):14–21, 1951. doi: 10.2307/2268661. URL https://doi.org/10.2307/2268661.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016a. URL http://aclweb.org/anthology/P/P16/P16-1228.pdf.

Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric P. Xing. Deep neural networks with massive learned knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1670–1679, 2016b. URL http://aclweb.org/anthology/D/D16/D16-1173.pdf.

Zhiting Hu, Zichao Yang, Ruslan R. Salakhutdinov, Lianhui Qin, Xiaodan Liang, Haoye Dong, and Eric P. Xing. Deep generative models with learnable knowledge constraints. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 10522–10533, 2018. URL http://papers.nips.cc/paper/8250-deep-generative-models-with-learnable-knowledge-constraints.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

Kazuo Iwama and Shuichi Miyazaki. Sat-variable complexity of hard combinatorial problems. In *In Proceedings of the World Computer Congress of the IFIP*, pages 253–258, 1994.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In

Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018, pages 876–885, 2018. URL http://auai.org/uai2018/proceedings/papers/313.pdf.

Prachi Jain, Pankaj Kumar, Mausam, and Soumen Chakrabarti. Type-sensitive knowledge base inference without explicit type supervision. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 2, pages 75–80, 2018.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. *CoRR*, abs/2310.20410, 2023. doi: 10.48550/ARXIV.2310.20410. URL https://doi.org/10.48550/arXiv.2310.20410.

Chi Jin, Praneeth Netrapalli, and Michael I. Jordan. Minmax optimization: Stable limit points of gradient descent ascent are locally optimal. *CoRR*, abs/1902.00618, 2019. URL http://arxiv.org/abs/1902.00618.

Rong Jin and Zoubin Ghahramani. Learning with multiple labels. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 897–904. MIT Press, 2002. URL http://papers.nips.cc/paper/2234-learning-with-multiple-labels.

David Joslin and Amitabha Roy. Exploiting symmetries in lifted csps. In *The 14th National Conference on Artificial Intelligence (AAAI)*, pages 197–202. AAAI Press, 1997.

Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

Alex Kantchelian, Michael Carl Tschantz, Ling Huang, Peter L. Bartlett, Anthony D. Joseph, and J. Doug Tygar. Large-margin convex polytope machine. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3248–3256, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/b147a61c1d07c1c999560f62add6dbc7-Abstract.html.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In Hugo Larochelle,

Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/49f85a9ed090b20c8bed85a5923c669f-Abstract.html.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

Hoel Kervadec, Jose Dolz, Meng Tang, Eric Granger, Yuri Boykov, and Ismail Ben Ayed. Constrained-cnn losses for weakly supervised segmentation. *Medical Image Analysis*, 54:88–99, 2019. doi: 10.1016/j.media.2019.02.009. URL https://doi.org/10.1016/j.media.2019.02.009.

Erich-Peter Klement, Radko Mesiar, and Endre Pap. Triangular norms. position paper I: basic analytical and algebraic properties. *Fuzzy Sets Syst.*, 143(1):5–26, 2004. doi: 10.1016/j.fss.2003.06.007. URL https://doi.org/10.1016/j.fss.2003.06.007.

Patrick Knobelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. End-to-End Training of Hybrid CNN-CRF Models for Stereo. In *CVPR*, 2017.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193, 9780262013192.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc V. Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7: 453–466, 2019.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *Proceedings of the 35th International Conference*

on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, pages 2869–2878, 2018.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270, 2016. URL http://aclweb.org/anthology/N/N16/N16-1030.pdf.

Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Mach. Learn.*, 81(1):53–67, October 2010. ISSN 0885-6125. doi: 10.1007/s10994-010-5205-8. URL http://dx.doi.org/10.1007/s10994-010-5205-8.

Nada Lavrac and Luc De Raedt. Inductive logic programming: A survey of european research. *AI Commun.*, 8(1):3–19, 1995. doi: 10.3233/AIC-1995-8101. URL https://doi.org/10.3233/AIC-1995-8101.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. URL http://yann.lecun.com/exdb/mnist/.

Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime G. Carbonell. Gradient-based inference for networks with output constraints. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 4147–4154, 2019. doi: 10.1609/aaai.v33i01.33014147. URL https://doi.org/10.1609/aaai.v33i01.33014147.

Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *ACL, ACL 2019, 2019*, 2019.

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. A logic-driven framework for consistency of neural models. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3922–3933. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1405. URL https://doi.org/10.18653/v1/D19-1405.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall F. Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 806–814, 2015. doi: 10.1109/CVPR.2015.7298681. URL https://doi.org/10.1109/CVPR.2015.7298681.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *7th International Conference on Learning Representations (ICLR)*, May 2019. URL https://openreview.net/pdf?id=BklHpjCqKm.

Abhijit Mahabal, Jason Baldridge, Burcu Karagol Ayan, Vincent Perot, and Dan Roth. Text classification with few examples using controlled generalization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 3158–3167, 2019. URL https://aclweb.org/anthology/papers/N/N19/N19-1319/.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+optimisation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/51311013e51adebc3c34d2cc591fefee-Abstract.html.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3753–3763, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html.

Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. Imposing hard constraints on deep networks: Promises and limitations. *CoRR*, abs/1706.02025, 2017. URL http://arxiv.org/abs/1706.02025.

Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, 23:190–217, 2012.

Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime G. Carbonell. Towards semi-supervised learning for deep semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4958–4963, 2018. URL https://aclanthology.info/papers/D18-1538/d18-1538.

Tao Meng and Kai-Wei Chang. An integer linear programming framework for mining constraints from data. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7619–7631. PMLR, 2021. URL http://proceedings.mlr.press/v139/meng21a.html.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michaël Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.7753.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11): 39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL http://doi.acm.org/10.1145/219717.219748.

Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Spair-71k: A large-scale benchmark for semantic correspondence. *CoRR*, abs/1908.10543, 2019. URL http://arxiv.org/abs/1908.10543.

Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022. doi: 10.1109/TPAMI.2021.3059968.

Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, 2020. URL http://arxiv.org/abs/1912.10824.

Giovanni Mingari Scarpello and Daniele Ritelli. A historical outline of the theorem of implicit functions. *Divulgaciones Matemáticas*, 10, 01 2002.

Amina Mollaysa, Alexandros Kalousis, Eric Bruno, and Maurits Diephuis. Learning to augment with feature side-information. In Wee Sun Lee and Taiji Suzuki, editors,

Proceedings of The 11th Asian Conference on Machine Learning, ACML 2019, 17-19 November 2019, Nagoya, Japan, volume 101 of *Proceedings of Machine Learning Research*, pages 173–187. PMLR, 2019. URL http://proceedings.mlr.press/v101/mollaysa19a.html.

Shikhar Murty, Patrick Verga, Luke Vilnis, and Andrew McCallum. Finer grained entity typing with typenet. *arXiv preprint arXiv:1711.05795*, 2017.

Shikhar Murty, Patrick Verga, Luke Vilnis, Irena Radovanovic, and Andrew McCallum. Hierarchical losses and new resources for fine-grained entity typing and linking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 97–109, 2018. URL https://aclanthology.info/papers/P18-1010/p18-1010.

Mehrab Najafian, Mohammad Hesam Tadayon, and Morteza Esmaeili. Construction of strongly mutually distinct sudoku tables and solid sudoku cubes by cyclotomic cosets. *IEEE Transactions on Games*, PP:1–1, 11 2018. doi: 10.1109/TG.2018.2880953.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 3075–3081. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14636.

Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. A primal dual formulation for deep learning with constraints. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12157–12168, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/cf708fc1decf0337aded484f8f4519ae-Abstract.html.

Yatin Nandwani, Deepanshu Jindal, Mausam, and Parag Singla. Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. In *International Conference on Learning Representations (ICLR)*, 2021.

Yatin Nandwani, Vidit Jain, Mausam, and Parag Singla. Neural models for output-space invariance in combinatorial problems. In *International Conference on Learning Representations (ICLR)*, 2022a. URL https://arxiv.org/abs/2202.03229.

Yatin Nandwani, Rishabh Ranjan, Mausam, and Parag Singla. A solver-free framework for scalable learning in neural ILP architectures. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022b. URL http://papers.nips.cc/paper_files/paper/2022/hash/3469b211b829b39d2b0cfd3b880a869c-Abstract-Conference.html.

Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015435. URL https://doi.org/10.1145/1015330.1015435.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33, 2016.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: Backpropagating through discrete exponential family distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14567–14579. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/7a430339c10c642c4b2251756fd1b484-Paper.pdf.

Qiang Ning, Zhili Feng, Hao Wu, and Dan Roth. Joint reasoning for temporal and causal relations. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2278–2288. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1212. URL https://aclanthology.org/P18-1212/.

Qiang Ning, Sanjay Subramanian, and Dan Roth. An improved neural baseline for temporal relation extraction. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 6202–6208. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1642. URL https://doi.org/10.18653/v1/D19-1642.

Vilém Novák. First-order fuzzy logic. *Studia Logica*, 46(1):87–109, 1987. doi: 10.1007/BF00396907. URL https://doi.org/10.1007/BF00396907.

Ozan Oktay, Enzo Ferrante, Konstantinos Kamnitsas, Mattias P. Heinrich, Wenjia Bai, Jose Caballero, Stuart A. Cook, Antonio de Marvao, Timothy Dawes, Declan P. O'Regan, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Anatomically constrained neural networks (acnns): Application to cardiac image enhancement and segmentation. *IEEE Trans. Med. Imaging*, 37(2):384–395, 2018. doi: 10.1109/TMI.2017.2743464. URL https://doi.org/10.1109/TMI.2017.2743464.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 3372–3382, 2018. URL http://papers.nips.cc/paper/7597-recurrent-relational-networks.

Xingyuan Pan and Vivek Srikumar. Expressiveness of rectifier networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2427–2435. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/panb16.html.

Xingyuan Pan and Vivek Srikumar. Learning to speed up structured output prediction. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3993–4002, 2018. URL http://proceedings.mlr.press/v80/pan18b.html.

Xingyuan Pan, Maitrey Mehta, and Vivek Srikumar. Learning constraints for structured prediction using rectifier networks. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4843–4858. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.438. URL https://doi.org/10.18653/v1/2020.acl-main.438.

George Papandreou and Alan L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc Van Gool, editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 193–200. IEEE Computer Society, 2011. doi: 10.1109/ICCV.2011.6126242. URL https://doi.org/10.1109/ICCV.2011.6126242.

Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2249–2255, 2016. URL http://aclweb.org/anthology/D/D16/D16-1244.pdf.

Kyubyong Park. 1 million sudoku games. https://www.kaggle.com/datasets/bryanpark/sudoku, 2017.

Kyubyong Park. Can convolutional neural networks crack sudoku puzzles? https://github.com/Kyubyong/sudoku, 2018.

Deepak Pathak, Philipp Krähenbühl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1796–1804, 2015. doi: 10.1109/ICCV.2015.209. URL https://doi.org/10.1109/ICCV.2015.209.

Anselm Paulus, Michal Rolínek, Vít Musil, Brandon Amos, and Georg Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8443–8453. PMLR, 2021. URL http://proceedings.mlr.press/v139/paulus21a.html.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=HkAClQgA-.

Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30, 2000.

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017. URL http://arxiv.org/abs/1712.04621.

Laurent Perron and Vincent Furnon. Or-tools, 2019. URL https://developers.google.com/optimization/.

Bart Pieters. Generic sudoku solver. https://github.com/bartp5/gss, 2019.

Marin Vlastelica Pogancic, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, CoNLL '12, pages 1–40, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=2391181.2391183.

Rajkumar Pujari and Dan Goldwasser. Using natural language relations between answer choices for machine comprehension. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4010–4015, 2019. URL https://aclweb.org/anthology/papers/N/N19/N19-1403/.

Vasin Punyakanok, Dan Roth, Wen-tau Yih, and Dav Zimak. Learning and inference over constrained output. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1124–1129. Professional Book Center, 2005. URL http://ijcai.org/Proceedings/05/Papers/1434.pdf.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287, 2008. doi: 10.1162/coli.2008.34.2.257. URL https://doi.org/10.1162/coli.2008.34.2.257.

Pengda Qin, Weiran Xu, and William Yang Wang. Robust distant supervision relation extraction via deep reinforcement learning. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2137–2147. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1199. URL https://www.aclweb.org/anthology/P18-1199/.

Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer, 2008. ISBN 978-3-540-78651-1. doi: 10.1007/978-3-540-78652-8. URL https://doi.org/10.1007/978-3-540-78652-8.

Sathya N. Ravi, Tuan Dinh, Vishnu Sai Rao Lokhande, and Vikas Singh. Constrained deep learning using conditional gradient and applications in computer vision. *CoRR*, abs/1803.06453, 2018. URL http://arxiv.org/abs/1803.06453.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3788–3800, 2017.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129, 2015. URL http://aclweb.org/anthology/N/N15/N15-1118.pdf.

Michal Rolínek, Vít Musil, Anselm Paulus, Marin Vlastelica P., Claudio Michaelis, and Georg Martius. Optimizing rank-based metrics with blackbox differentiation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 7617–7627. Computer Vision Foundation / IEEE, 2020a. doi: 10.1109/CVPR42600.2020.00764. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Rolinek_Optimizing_Rank-Based_Metrics_With_Blackbox_Differentiation_CVPR_2020_paper.html.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*, volume 12373 of *Lecture Notes in Computer Science*, pages 407–424. Springer, 2020b. doi: 10.1007/978-3-030-58604-1_25. URL https://doi.org/10.1007/978-3-030-58604-1_25.

Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In Hwee Tou Ng and Ellen Riloff, editors, *Proceedings of the Eighth Conference on Computational Natural Language Learning, CoNLL 2004, Held*

in cooperation with HLT-NAACL 2004, Boston, Massachusetts, USA, May 6-7, 2004, pages 1–8. ACL, 2004. URL https://aclanthology.org/W04-2401/.

Dan Roth and Wen-tau Yih. Integer linear programming inference for conditional random fields. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 736–743, 2005. doi: 10.1145/1102351.1102444. URL https://doi.org/10.1145/1102351.1102444.

Soumava Kumar Roy, Zakaria Mhammedi, and Mehrtash Harandi. Geometry aware constrained optimization techniques for deep learning. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4460–4469, 2018. doi: 10.1109/CVPR.2018.00469. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Roy_Geometry_Aware_Constrained_CVPR_2018_paper.html.

Gordon Royle. Minimum sudoku. https://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php, 2014.

Alexander M. Rush and Michael Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *J. Artif. Intell. Res.*, 45:305–362, 2012. doi: 10.1613/jair.3680. URL https://doi.org/10.1613/jair.3680.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN 9780134610993. URL http://aima.cs.berkeley.edu/.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, page 901, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/ed265bc903a5a097f61d3ec064d96d2e-Abstract.html.

Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018,*

Montréal, Canada, pages 7310–7321, 2018. URL http://papers.nips.cc/paper/7960-relational-recurrent-neural-networks.

Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel S. Weld. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1088–1098, 2010.

Hanie Sedghi and Ashish Sabharwal. Knowledge completion for generics using guided tensor factorization. *TACL*, 6:197–210, 2018.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1099. URL https://doi.org/10.18653/v1/P17-1099.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Baoxu Shi and Tim Weninger. Discriminative predicate path mining for fact checking in knowledge graphs. *Knowl. Based Syst.*, 104:123–133, 2016. doi: 10.1016/j.knosys.2016.04.015. URL https://doi.org/10.1016/j.knosys.2016.04.015.

Prashant Shiralkar, Alessandro Flammini, Filippo Menczer, and Giovanni Luca Ciampaglia. Finding streams in knowledge graphs to support fact checking. In Vijay Raghavan, Srinivas Aluru, George Karypis, Lucio Miele, and Xindong Wu, editors, *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, pages 859–864. IEEE Computer Society, 2017. doi: 10.1109/ICDM.2017.105. URL https://doi.org/10.1109/ICDM.2017.105.

Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In Peter Auer and Ron Meir, editors, *Learning Theory, 18th Annual Conference on Learning Theory, COLT 2005, Bertinoro, Italy, June 27-30, 2005, Proceedings*, volume 3559 of *Lecture Notes in Computer Science*, pages 545–560. Springer, 2005. doi: 10.1007/11503415_37. URL https://doi.org/10.1007/11503415_37.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: 10.1145/1242572.1242667. URL http://doi.acm.org/10.1145/1242572.1242667.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1139–1147. JMLR.org, 2013. URL http://proceedings.mlr.press/v28/sutskever13.html.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06067.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4949–4953, 2017.

Yingcong Tan, Andrew Delong, and Daria Terekhov. Deep inverse optimization. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 540–556. Springer, 2019. doi: 10.1007/978-3-030-19212-9_36. URL https://doi.org/10.1007/978-3-030-19212-9_36.

Yingcong Tan, Daria Terekhov, and Andrew Delong. Learning linear programs from optimal decisions. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020,*

December 6-12, 2020, virtual, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e44e875c12109e4fa3716c05008048b2-Abstract.html.

Niket Tandon, Bhavana Dalvi, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. Reasoning about actions and state changes by injecting commonsense knowledge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 57–66, 2018. URL https://aclanthology.info/papers/D18-1006/d18-1006.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/b64a70760bb75e3ecfd1ad86d8f10c88-Abstract.html.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003. URL https://aclanthology.org/W03-0419.

Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artif. Intell.*, 70(1-2):119–165, 1994. doi: 10.1016/0004-3702(94)90105-8. URL https://doi.org/10.1016/0004-3702(94)90105-8.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2071–2080, 2016. URL http://jmlr.org/proceedings/papers/v48/trouillon16.html.

Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *IJDWM*, 3(3):1–13, 2007. doi: 10.4018/jdwm.2007070101. URL https://doi.org/10.4018/jdwm.2007070101.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*,

pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

Patrick Verga, Arvind Neelakantan, and Andrew McCallum. Generalizing to unseen entities and entity pairs with row-less universal schema. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 613–622, 2017. URL https://aclanthology.info/papers/E17-1058/e17-1058.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):652–663, 2017. doi: 10.1109/TPAMI.2016.2587640. URL https://doi.org/10.1109/TPAMI.2016.2587640.

Khuong Vo, Tri Nguyen, Dang Pham, Mao Nguyen, Minh Truong, Trung Mai, and Tho T. Quan. Combination of domain knowledge and deep learning for sentiment analysis of short and informal messages on social media. *Int. J. Comput. Vis. Robotics*, 9(5): 458–485, 2019. doi: 10.1504/IJCVR.2019.102286. URL https://doi.org/10.1504/IJCVR.2019.102286.

Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR*, abs/1711.00399, 2017. URL http://arxiv.org/abs/1711.00399.

Toby Walsh. SAT v CSP. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000. doi: 10.1007/3-540-45349-0_32. URL https://doi.org/10.1007/3-540-45349-0_32.

Haoyu Wang, Muhao Chen, Hongming Zhang, and Dan Roth. Joint constrained learning for event-event relation extraction. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 696–706. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.51. URL https://doi.org/10.18653/v1/2020.emnlp-main.51.

Ming-Wei Wang, Nicholas Rizzolo, and Dan Roth. Integer linear programming in NLP - constrained conditional models. In Jason Baldwin, Peter Clark, and Gökhan Tür, editors, *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 1, 2010, Los Angeles, California, USA - Tutorial Abstracts*, pages 9–14. Association for Computational Linguistics, 2010. URL https://aclanthology.org/N10-4005/.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19e.html.

Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1859–1866. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/264.

Yiwei Wang, Muhao Chen, Nanyun Peng, and Kai-Wei Chang. Deepedit: Knowledge editing as decoding with constraints. *CoRR*, abs/2401.10471, 2024. doi: 10.48550/ARXIV.2401.10471. URL https://doi.org/10.48550/arXiv.2401.10471.

Daniel S. Weld and Gagan Bansal. The challenge of crafting intelligible intelligence. *CoRR*, abs/1803.04263, 2018. URL http://arxiv.org/abs/1803.04263.

Lu Wen, Jingliang Duan, Shengbo Eben Li, Shaobing Xu, and Huei Peng. Safe reinforcement learning for autonomous vehicles through parallel constrained policy optimization. In *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*, pages 1–7. IEEE, 2020. doi: 10.1109/ITSC45102.2020.9294262. URL https://doi.org/10.1109/ITSC45102.2020.9294262.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA,*

January 27 - February 1, 2019, pages 1658–1665. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33011658. URL https://doi.org/10.1609/aaai.v33i01.33011658.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

Han Xiao, Minlie Huang, and Xiaoyan Zhu. Knowledge semantic representation: A generative model for interpretable knowledge graph embedding. *CoRR*, abs/1608.07685, 2016. URL http://arxiv.org/abs/1608.07685.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*, 2017. URL https://aclanthology.info/papers/D17-1060/d17-1060.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5498–5507, 2018. URL http://proceedings.mlr.press/v80/xu18h.html.

Ning Xu, Jiaqi Lv, and Xin Geng. Partial label learning via label enhancement. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5557–5564. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33015557. URL https://doi.org/10.1609/aaai.v33i01.33015557.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning linear ranking functions for beam

search with application to planning. *J. Mach. Learn. Res.*, 10:1571–1610, 2009. doi: 10.5555/1577069.1755837. URL https://dl.acm.org/doi/10.5555/1577069.1755837.

Yexiang Xue, Yang Yuan, Zhitian Xu, and Ashish Sabharwal. Expanding holographic embeddings for knowledge completion. In *Advances in Neural Information Processing Systems 31*, pages 4491–4501, 2018.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, 2018.

Zhun Yang, Adam Ishay, and Joohyung Lee. Learning to solve constraint satisfaction problems with recurrent transformer. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=udNhDCr2KQe.

Limin Yao, Sebastian Riedel, and Andrew McCallum. Universal schema for entity type prediction. In *Proceedings of the 2013 workshop on Automated knowledge base construction, AKBC@CIKM 13, San Francisco, California, USA, October 27-28, 2013*, pages 79–84, 2013. doi: 10.1145/2509558.2509572. URL https://doi.org/10.1145/2509558.2509572.

Gilad Yehudai, Ethan Fetaya, Eli A. Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pages 11975–11986, 2021. URL http://proceedings.mlr.press/v139/yehudai21a.html.

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4651–4659. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.503. URL https://doi.org/10.1109/CVPR.2016.503.

Qiang Zeng, Jignesh M. Patel, and David Page. Quickfoil: Scalable inductive logic programming. *Proc. VLDB Endow.*, 8(3):197–208, 2014. doi: 10.14778/2735508.2735510. URL http://www.vldb.org/pvldb/vol8/p197-patel.pdf.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339. PMLR, 2020. URL http://proceedings.mlr.press/v119/zhang20ae.html.

Xiao Zhang, Maria Leonor Pacheco, Chang Li, and Dan Goldwasser. Introducing DRAIL - a step towards declarative deep relational learning. In *Proceedings of the Workshop on Structured Prediction for NLP@EMNLP 2016, Austin, TX, USA, November 5, 2016*, pages 54–62, 2016. doi: 10.18653/v1/W16-5906. URL https://doi.org/10.18653/v1/W16-5906.

Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018. URL http://arxiv.org/abs/1807.05511.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Part VI

Appendices

Appendix A

Appendix to Chapter 3

A.1 Experimental Details

Software environment details:

- **Computational Resources:** All our models are trained on PADUM: Hybrid High Performance Computing Facility at IITD.[1]
- **Software Used:** All models are trained using PyTorch[2]. For NER and SRL experiments, we use Allennlp[3] library which is built on top of PyTorch.
- **Optimizer:** For w updates, we use the same optimizer as used in the base model and for λ updates we use SGD with momentum of 0.9.

A.1.1 Semantic Role Labeling (SRL)

Hyperparameters: In all the experiments, *warmup* iterations and initial value of $l = l_0$ is selected in the same way: to select *warmup* iterations, we train the base model without constraints till convergence and as a rule of thumb, select *warmup* iterations as the iteration number where it reaches around 25% of its peak performance. Initial value of $l = l_0$ is set arbitrarily at 10. Initial learning rate α_Λ^0 is fixed at 0.05. Constant d and learning rate decay

Table A.1: Best hyper-parameters in SRL experiments for different training sizes.

	Constant d	**Decay** β
1% Data	1	1/5
5% Data	10	1
10% Data	10	1

[1] *http://supercomputing.iitd.ac.in*
[2] https://pytorch.org/
[3] https://allennlp.org/

parameters β is selected through a grid search over $\{1, 10\}$ and $\{1, 1/5, 1/10\}$ respectively and we select the best combination based on the performance over dev set. Table A.1 enumerates the best value of these two hyper-parameters for different training sizes.

A.1.2 Named Entity Recognition (NER)

Hyperparameters: *warmup* iterations and initial value of l are selected as in SRL experiments. In these experiments, we do not decay the learning rate and set β to 0. To select the learning rate α_Λ, and constant d, we do a grid search over $\{0.01, 0.05\}$ and $\{1, 5\}$ respectively and select the best combination based on the performance over dev set. Table A.2 enumerates the best value of these two hyper-parameters for different training sizes in both the settings: constrained learning and semi-supervision.

Table A.2: Best hyper-parameters in NER for different training sizes in both scenarios: CL and SCL.

	CL		SCL	
Training Size	**Learning Rate** α_Λ	**Constant** d	**Learning Rate** α_Λ	**Constant** d
400	0.05	5	0.01	5
800	0.05	1	0.01	5
1,600	0.05	1	0.05	1
3,200	0.05	1	0.05	1
6,400	0.01	5	0.01	5
12,800	0.05	1	0.01	5
25,600	0.01	5	0.01	5
37,206	0.01	5	0.01	5

A.1.3 Fine Grained Entity Typing

Hyperparameters: *warmup* iterations and initial value of l are selected as in the above two experiments. As in NER experiments, we do not decay the learning rate and set β to 0. We observed that higher values of the constant d hurts the performance and increase the number of constraint violations as well. Hence, we set it to 0 which gives the best results. To select the learning rate α_Λ, we do a grid search over $\{0.01, 0.02, 0.03, 0.04, 0.05\}$ and select the best value based on the performance over dev set. Table A.3 enumerates its best value for

Table A.3: Best hyper-parameters in Typenet for different training sizes in both scenarios: CL and SCL.

	CL	SCL
Training Size	**Learning Rate** α_Λ	**Learning Rate** α_Λ
5% Data	0.05	0.01
10% Data	0.02	0.03
100% Data	0.04	

different training sizes in both the settings of constrained learning and semi-supervision.

Appendix B

Appendix to Chapter 4

B.1 Experimental Details

Hardware Architecture: Each experiment is run on a 12GB NVIDIA K40 GPU with 2880 CUDA cores and 4 cores of Intel E5-2680 V3 2.5GHz CPUs.

Optimizer: We use Adam as our optimizer in all our experiments. Initial learning rate is set to 0.005 for NLM [Dong et al., 2019] experiments while it is kept at 0.001 for RRN [Palm et al., 2018] experiments. Learning rate for RL phase is kept at 0.1 times the initial learning rate. We reduce learning rate by a factor of 0.2 whenever the performance on the dev set plateaus.

B.1.1 Datasets and Prediction Networks

B.1.1.1 N-Queens

Given a query, *i.e.*, a chess-board of size $N \times N$ and a placement of $k < N$ non-attacking queens on it, the task of N Queens is to place the remaining $N - k$ queens, such that no two queens are attacking each other. We train a Neural Logic Machine (NLM) model [Dong et al., 2019] as the prediction network $\mathbf{M}_\Theta$ for solving queries for this task. To model N-Queens within NLM, we represent a query $\mathbf{x}$ and the target $\mathbf{y}$ as N^2 dimensional Boolean vectors with 1 at locations where a Queen is placed. We use another smaller NLM architecture as the latent model G_ϕ.

We train our model on 10–Queens puzzles and test on 11–Queens puzzles, both with 5 placed queens. This size-invariance in training and test is a key strength of NLM architecture, which we exploit in our experiments.

Data Generation: To generate the train data, we start with all possible valid 10–Queens board configurations. We then generate queries by randomly masking any 5 queens. We

check for all possible valid completions to generate potentially multiple solutions for any given query. Test data is also generated similarly. Training and testing on different board sizes ensures that no direct information leaks from the test dataset to the train dataset. Queries with multiple solutions have a small number of total solutions (2-6), hence we choose $\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}, \forall \mathbf{x}^{(i)}$.

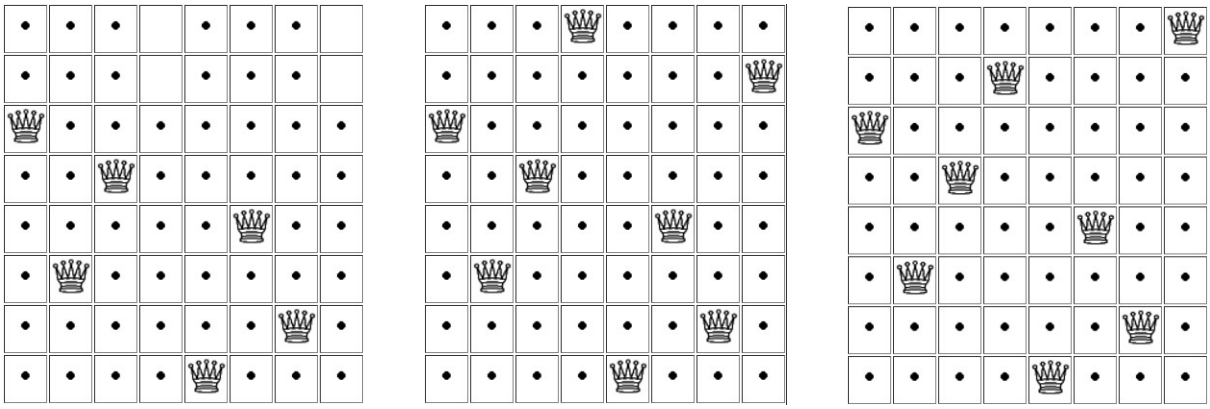


Figure B.1: 8-Queens query along with its two possible solution.[1]

Architecture Details for Prediction Network $\mathbf{M}_\Theta$: We use Neural Logic Machines (NLM)[2] [Dong et al., 2019] as the base prediction network $\mathbf{M}_\Theta$ for this task. NLM consists of a series of basic blocks, called 'Logic Modules', stacked on top of each other with residual connections. Number of blocks in an NLM architecture is referred to as its *depth*. Each block takes grounded predicates as input and learns to represent M intermediate predicates as its output. See [Dong et al., 2019] for further details. We chose an architecture with $M = 8$ and $depth = 30$. We keep the maximum arity of intermediate predicates learnt by the network to be 2.

Input Output for Prediction Network: Input is provided in terms of grounded unary and binary predicates and the architecture learns to represent an unknown predicate in terms of the input predicates. Each cell on the board acts as an atomic variable over which predicates are defined.

1. **Unary Predicates:** To indicate the presence of a Queen on a cell in the input, we use a unary predicate, '*HasQueenPrior*'. It is represented as a Boolean tensor $\mathbf{x}$ of size N^2 with 1 on k out of N^2 cells indicating the presence of a Queen. The output $\mathbf{y}$ of the network is also a unary predicate '*HasQueen*' which indicates the final position of the queens on board.

[1] Image Source: Game play on http://www.brainmetrix.com/8-queens/
[2] Code taken from: https://github.com/google/neural-logic-machines

2. **Binary Predicates:** We use 4 binary predicates to indicate if two cells are in same row, same column, same diagonal or same off-diagonal. The binary predicates are a constant for all board configurations for a given size N and hence can also be thought of as part of network architecture instead of input.

Architecture Details for Selection Module S_ϕ: We use another NLM as our latent model G_ϕ within the selection module S_ϕ. We fix $depth = 4$ and $M = 10$ for the latent model.

Input Output for G_ϕ: Input to G_ϕ is provided in terms of grounded unary and binary predictates represented as tensors just like the prediction network. G_ϕ takes 1 unary predicate as input, represented as an N^2 sized vector, $\mathbf{y}_j^{(i)} - \hat{\mathbf{y}}_-^{(i)}$, where $\hat{\mathbf{y}}_-^{(i)}$ is the prediction from its internal copy of the prediction network ($\mathbf{M}_{\Theta_}$) given the query $\mathbf{x}^{(i)}$. For each $\mathbf{y}_j^{(i)} \in \mathbf{Y}_{\mathbf{x}^{(i)}}$, G_ϕ returns a score which is converted into a probability distribution $Pr_\phi(\mathbf{y}_j^{(i)})$ over $\mathbf{Y}_{\mathbf{x}^{(i)}}$ using a softmax layer.

Hyperparameters:

The list below enumerates the various hyper-parameters with a brief description (whenever required) and the set of its values that we experiment with. Best value of a hyper-parameter is selected based on performance on a held out validation set.

1. **Data Sampling:** Since number of queries with multiple solutions is underrepresented in the training data, we up-sample them and experiment with different ratios of multi-solution queries in the training data. Specifically, we experiment with the ratios of 0.5 and 0.25 in addition to the two extremes of selecting queries with only unique or only multiple solutions. Different data sampling may be used during pre-training and RL fine tuning phases.

2. **Batch Size:** We use a batch size of 4. We selected the maximum batch size that can be accommodated in 12GB GPU memory.

3. ***copyitr*:** We experiment with two extremes of copying the prediction network after every update and copying after every 2500 updates.

4. **Weight Decay in Optimizer:** We experiment with different weight decay factors of 1E-4, 1E-5 and 0.

5. **Pretraining ϕ:** We pretrain G_ϕ for 250 updates.

Training Time: Pre-training takes $10 - 12$ hours while RL fine-tuning take roughly $6 - 8$ hours using the hardware mentioned in the beginning of the section.

B.1.1.2 Futoshiki

This is a logic puzzle in which we are given a grid of size $N \times N$, and the goal is to fill the grid with digits from $\{1 \dots N\}$ such that no digit is repeated in a row or a column. k out of N^2 positions are already filled in the input query $\mathbf{x}$ and the remaining $N^2 - k$ positions need to be filled. Further, inequality constraints are specified between some pairs of adjacent grid positions, which need to be honored in the solution. Our prediction network, and latent model use NLM. Similar to N–Queens, we do size-invariant training – we train our models on 5×5 puzzles with 14 missing digits and test on 6×6 puzzles with 20 missing digits. For both train and test queries we keep up to five inequality constraints of each type: $>$ and $<$.

Data Generation: We start with generating all the possible ways in which we can fill a $N \times N$ grid such that no number appears twice in a row or column. For generating a query we sample any solution and randomly mask out k positions on it. Also we enumerate all the $GreaterThan$ and $LessThan$ relations between adjacent pair of cells in the chosen solution and randomly add q of these relations to the query. We check for all possible valid completions to generate potentially multiple solutions for any given query. Test data is also generated similarly. Queries with multiple solutions have a small number of total solutions (2-6), so we choose $\mathbf{Y}_{\mathbf{x}^{(i)}} = \mathcal{Y}_{\mathbf{x}^{(i)}}, \forall \mathbf{x}^{(i)}$.

Input Output for Prediction Network: Just like N-Queens experiment, the input to the network is a set of grounded unary and binary predicates. We define a grid cell along with the digit to be filled in it as an atomic variable. There are N^2 cells in the grid and each cell can take N values, thus we have N^3 atomic variables over which the predicates are defined.

1. **Unary Predicates:** To indicate the presence of a value in a cell in the input, we use a unary predicate, '*IsPresentPrior*'. It is represented as a Boolean tensor $\mathbf{x}$ of size N^3 with 1 on k positions indicating the presence of a digit in a cell. The output $\mathbf{y}$ of the network is also a unary predicate '*IsPresent*' which indicates the final prediction of grid. Additionally, there are two more unary predicates which represent the inequality relations that need to be honoured. Since inequality relations are defined only between pair of adjacent cells we can represent them using unary predicates.

2. **Binary Predicates:** We use 3 binary predicates to indicate if two vairables are in same row, same column, or same grid cell. The binary predicates are a constant for all board configurations for a given size N.

Input Output for G_ϕ: Same as N-Queens experiment except for the addition of two more unary predicates corresponding to the inequality relations.

Hyperparameters: Same as N-Queens experiment.

Training Time: Pre-training takes roughly $12 - 14$ hours while RL fine-tuning takes $7 - 8$ hours.

B.1.1.3 Sudoku

We also experiment on Sudoku, which has been used as the task of choice for many recent neural reasoning works [Palm et al., 2018, Wang et al., 2019]. We use Relational Recurrent Networks (RRN) [Palm et al., 2018] as the prediction network since it has recently shown state-of-the-art performance on the task. We use a 5 layer CNN as our latent model G_ϕ. Existing Sudoku datasets [Royle, 2014, Park, 2018], do not expose the issues with solution multiplicity. In response, we generate our own dataset that has equal number of queries with unique and multiple solutions. Some $\mathbf{x}^{(i)}$s may have hundreds of solutions, therefore we randomly sample 5 of them from $\mathcal{Y}_{\mathbf{x}^{(i)}}$, i.e., $|\mathbf{Y}_{\mathbf{x}^{(i)}}| \leq 5$ in the train set.

Data Generation: We start with the dataset proposed by Palm et al. [2018]. It has $180k$ queries with only unique solution and the number of givens are uniformly distributed in the range from 17 to 34. [3]. For the queries with unique solution, we randomly sample $10,000$ queries from their dataset, keeping their train, val and test splits. Using the queries with 17-givens from the entire dataset of size $180k$, we use the following procedure to create queries with multiple solutions:

We know that for a Sudoku puzzle to have a unique solution it must have 17 or more givens [McGuire et al., 2012]. So we begin with the set of 17-givens puzzles having a unique solution and randomly remove 1 of the givens, giving us a 16-givens puzzle which necessarily has more than 1 correct solution. We then randomly add 1 to 18 of the digits back from the solution of the original puzzle, while ensuring that the query continues to have more than 1 solution. [4] This procedure gives us multi-solution queries with givens in the range of 17 to 34, just as the original dataset of puzzles with only unique solution. We also observed that often there are queries which have a very large number of solutions (> 100). We found that such Sudoku queries are often too poorly defined to be of any interest. So we filter out all queries having more than 50 solutions. To have the same uniform distribution of number of givens as in the original dataset of puzzles with unique solution, we sample queries from this set of puzzles with multiple solutions such that we have a uniform distribution of number of givens in our dataset. We repeat this procedure to generate our validation and test data by starting from validation and test datasets from Palm et al. [2018].

[3] Available at https://data.dgl.ai/dataset/sudoku-hard.zip

[4] We identify all solutions to a puzzle using http://www.enjoysudoku.com/JSolve12.zip

Architecture Details for Prediction Network $\mathbf{M}_\Theta$: We use Recurrent Relational Network (RRN) [Palm et al., 2018] [5] as the prediction network for this task. RRN uses a message passing based inference algorithm on graph objects. Each cell in grid is represented as a node in the graph. All the cells in the same row, column and box are connected in the graph. Each inference involves 32 steps of message passing between the nodes in the graph and the model outputs a prediction at each step.

Input Output for Prediction Network: Input to the prediction network is represented as a 81×10 matrix with each of the 81 cell represented as a one-hot vector representing the digits ($0-9$, 0 if not given). Output of the prediction network is a $81 \times 10 \times 32$ tensor formed by concatenating the prediction of network at each of the 32 steps of message passing. The prediction at the last step is used for computing accuracy.

Architecture Details for Selection Module S_ϕ: We use a CNN as the latent model G_ϕ. The network consists of four convolutional layers followed by a fully connected layer. The four layers have 100, 64, 32 and 32 filters respectively. Each filter has a size of 3×3 with stride of length 1.

Input Output for G_ϕ: Similar to the other two experiments, the input to G_ϕ is the output $\hat{\mathbf{y}}_{_}^{(i)}$ from the selection module's internal copy $\mathbf{M}_{\Theta_}$ along with $\mathbf{y}_j^{(i)}$. Since the prediction network gives an output at each step of message passing, we modify the G_ϕ and the rewards for S_ϕ accordingly to be computed from prediction at each step instead of relying only on the final prediction.

Hyperparameters:

1. **Data Sampling:** Since number of queries with multiple solutions and queries with unique solution are in equal proportion, we no longer need to upsample multi-solution queries.

2. **Batch Size:** We use a batch size of 32 for training the baselines, while for RL based training we use a batch size of 16.

3. ***copyitr*:** We experiment with $copyitr = 1$ i.e. copying $\mathbf{M}_\Theta$ to $\mathbf{M}_{\Theta_}$ after every update.

4. **Weight Decay in Optimizer:** We experiment with weight decay factor of 1E-4 (same as Palm et al. [2018]).

5. **Pretraining ϕ:** We pretrain G_ϕ for 1250 updates, equivalent to one pass over the train data.

[5] Code taken from: https://github.com/dmlc/dgl/tree/master/examples/pytorch/rrn

B.1.2 Results and Discussions

We repeat all the experiments thrice using different seeds. Below, we compare the performance gains for each of the seed separately.

Seed-wise Comparison for Gains of SelectR over MinLoss In Table B.1 we see that SelectR performs better than MinLoss for each of the three random seeds independently in all the experiments. We note that starting with the same seed in our implementation leads to identical initialization of the prediction network parameters.

Table B.1: Seed wise gains of SelectR over MinLoss across different random seeds and experiments

Seed	Sudoku	NQueens	Futoshiki
42	3.12%	3.40%	0.69%
1729	2.75%	2.53%	1.21%
3120	1.98%	1.39%	1.77%
Avg. Gain	2.61%	2.44%	1.22%

Appendix C

Appendix to Chapter 5

C.1 Models Description

C.1.1 Alternate encoding scheme in Binarized Model

As discussed in section 5.4.1, the main intuition for our binarized modelcomes from 'sparse encoding' of an integer CSP to a SAT. In addition to 'sparse encoding', there is another way of converting integer CSP into SAT, called 'compact encoding' [Ernst et al., 1997, Iwama and Miyazaki, 1994], in which each Boolean SAT variable represents a single bit of the integer value that a CSP variable can take. The final assignment of a CSP variable is given by the integer represented by the $\log k$ Boolean SAT variables corresponding to that variable. Motivated by the 'compact encoding', one can construct another model: instead of a one-hot encoding which requires k nodes (one for each value in $\mathcal{V}$) for each variable, create $\log k$ binary valued nodes for each variable and assign a value $v \in \mathcal{V}$ to the variable based on the integer represented by $\log k$ bits corresponding to it. This results in a graph with $|N_C| \log k$ nodes for a CSP with $|N_C|$ variables and k classes, instead of $|N_C|k$ nodes in the binarized model, and brings it closer to the graph size of $|N_C| + k$ created in multi-valued model. However, such an approach failed to generalize across the size of the value-set in our experiments. In addition, such an encoding has a limitation in its representational capability. It can not encode the relationship between the values effectively. For example, in $k \times k$ Futoshiki, we have an ordinal relationship between the k values representing the numerical numbers 1 to k. In our proposed approaches, we encode this by adding appropriate relational edges between nodes representing different values in $\mathcal{V}$. In the binarized model, it is done for each variable separately, whereas, in the multi-valued model, it is done in the RMPG. In absence of an explicit node for a value in this encoding scheme, it is not clear how to represent such a relationship.

Table C.1: Hyperparameters for different models and tasks

Model	Batch Size	Weight Decay	Orthogonality Loss Factor	Edge Dropout
Futoshiki				
MV	64	0.0001	0.01	0.1
BIN	16	0.0001	-	0.1
NLM15	4	0.0001	-	-
NLM30	2	0.0001	-	-
GCP				
MV	64	0.0001	0.01	0.1
BIN	16	0.0001	-	0.1
NLM24	1	0.0001	-	-
Sudoku				
MV	28	0.0001	0.01	0.1
BIN	3	0.0001	-	0.1

C.1.2 Experimental Details

Computational resources: All models are trained on K40 GPUs with 12 GB memory, available on an HPC cluster.

Hyperparameters: The list below enumerates the various hyperparameters with a brief description and the values chosen.

1. **Batch Size:** For each task, we selected the maximum batch size that can be accommodated in 12GB GPU memory. Refer to Table C.1 for details.

2. **Optimizer:** To minimize the loss, we use Adam optimizer with learning rate 0.0002. As in the original RRN paper, we chose a weight decay factor of 1E-4.

3. **Orthogonality Loss Factor:** To ensure that the multi-valued model learns different embeddings for each value in the value-set, we add an auxiliary loss term, corresponding to the total pairwise dot product of any two embeddings, before and after message passing on the Relation Message Passing Graph (RMPG), $\tilde{G}$. Its weight, α, was chosen amongst $\{0.01, 0.1, 0.5\}$ by cross validating on a devset for Futoshiki, and then fixed afterwards for all our experiments.

4. **Edge Dropout:** While collating the messages from the edges of the same type, we drop 10% of the messages, as done in RRN. Dropout is used in the Message Passing Graph (MPG) of the binarized model, and the Constraint Message Passing Graph (CMPG) of the multi-valued model.

Table C.2: Training cost of different models in terms of number of epochs, gradient updates and clock time

Model	Batch Size	Training Data Size	# Gradient Updates	#Epochs	Time per Epoch (min)	Total Time (Hours)
			Futoshiki			
MV	64	12,300	60,000	312	5	25
BIN	16	12,300	37,500	49	26	21
NLM15	4	12,300	155,000	50	34	43
NLM30	2	12,300	232,500	38	87	66
			GCP			
MV	64	25,010	80,000	205	10	33
BIN	16	25,010	40,000	26	39	17
NLM24	1	25,010	260,000	10	213	37
			Sudoku			
MV	28	10,000	162,000	454	9	68
BIN	3	10,000	168,000	50	74	63

Model Averaging: As suggested in [Izmailov et al., 2018], to reduce the variance of our model performance, we take simple average of model weights stored at multiple points. All checkpoints beyond a point when the learning curve flattened are selected for computing the average.

Training Time: Table C.2 enumerates the exact training cost, in terms of total training epochs, number of gradient updates, and clock time, for all three tasks and for both our models as well as the baseline NLM model. Note that while a multi-valued model may have fewer parameters, and results in a much smaller graph and inference time for a given problem, its training time could still be higher, especially in terms of total training epochs and number of gradient updates. However, because of its memory efficiency, we can keep a much larger batch size during training, and because of its speed efficiency, each update is much faster. As a result, the overall clock time comes out to be comparable to the binary model for the two of our tasks, i.e. Futoshiki and Sudoku, and it is within 2x for GCP, even though the number of epochs is much higher.

Appendix D

Appendix to Chapter 6

D.1 Mean and Std Err. over multiple runs

Random Constraints: See table D.1 for the standard error of the accuracy and training time over 10 random datasets for different number of ground truth constraints.
Knapsack from Sentence Descriptions: See table D.2 for standard error over 10 runs with different random seeds for varying knapsack sizes.
Keypoint Matching: See table D.3 for the standard error of point-wise accuracy and training times over 3 runs with different random seeds for varying number of keypoints.

Table D.1: Mean ± std err of the vector accuracy ($\mathbf{M}_\Theta(x) = \mathbf{y}^*$) and training time over the 10 random datasets in each setting of Random constraints. Number of learnable constraints is twice the number of ground truth constraints. CombOpt: CombOptNet

	Vector Accuracy (%)				Training Time (min)			
	1	2	4	8	1	2	4	8
				Binary				
CombOpt	97.6 ± 0.4	95.3 ± 0.5	84.3 ± 3.5	63.4 ± 4.0	8.2 ± 1.7	13.5 ± 1.6	26.5 ± 1.4	40.8 ± 4.0
ILP–Loss	**97.8 ± 0.4**	**96.0 ± 0.3**	**92.8 ± 0.6**	**87.8 ± 3.4**	**7.3 ± 1.7**	**11.6 ± 1.9**	**18.1 ± 2.4**	**27.5 ± 4.8**
				Dense				
CombOpt	89.3 ± 1.1	74.8 ± 1.9	34.3 ± 5.6	2.0 ± 0.6	9.9 ± 1.4	16.8 ± 1.3	24.7 ± 2.0	48.2 ± 2.3
ILP–Loss	**96.6 ± 0.3**	**86.3 ± 2.3**	**74.0 ± 5.4**	**41.5 ± 5.7**	**7.3 ± 1.1**	**15.6 ± 2.1**	**17.6 ± 2.6**	**20.6 ± 4.5**

Table D.2: Mean ± std err. of accuracy and train-time over 10 runs for various knapsack datasets with different number of items in each instance. For $N = 25, 30$, *TO* represents timeout of 12 hours and we evaluate using the latest snapshot of the model obtained within 12 hours of total training. CombOpt: CombOptNet

	Vector Accuracy (mean ± std. err. in %)					Training Time (mean ± std. err. in min)				
	10	15	20	25	30	10	15	20	25	30
CombOpt	63.2±0.6	48.2±0.4	30.1±1.0	2.6±0.3	0.0±0.0	**41.0±4.1**	61.4±4.6	153.0±8.8	*TO*	*TO*
ILP–Loss	**71.4±0.4**	**58.5±0.3**	**48.7±0.7**	**41.0±0.5**	**28.4±0.7**	44.0±5.7	**51.0±7.9**	**82.2±9.8**	**106.1±7.7**	**111.6±11.0**

Table D.3: Mean ± std error of point-wise accuracy and training times over 3 runs with random seeds for varying number of keypoints. Neural+CI denotes ILP inference with known constraints over the cost learnt by neural model.

	Pointwise Accuracy (mean ± std err in %)				Training Times (mean ± std. err. in min)			
	4	5	6	7	4	5	6	7
Neural	80.88±0.87	78.04±0.40	75.39±0.50	73.49±0.55	148±26	**37±12**	**30±9**	**40±13**
Neural + CI	82.42±0.55	79.99±0.16	77.64±0.25	75.88±0.43	148±26	**37±12**	**30±9**	**40±13**
CombOptNet	83.86±0.62	**81.43±0.49**	78.88±0.65	76.85±0.54	**41±15**	67±8	144±31	279±39
ILP–Loss	81.76±1.71	79.59±0.18	77.84±0.36	76.18±0.06	115±13	92±3	106±1	109±5
ILP–Loss + Sol.	**84.64±0.62**	81.27±1.12	**79.51±0.53**	**78.59±0.55**	43±12	73±10	99±9	174±25

Appendix E

Appendix to Chapter 7

E.1 Related Works

E.1.1 Comparison of TF based KBC Models with Path based Approaches

Methods like DeepPath, MINERVA, are inherently explainable as they work directly with paths, unlike TF based methods. However, such methods either do not scale or do not perform as well as TF based methods for large scale KBs such as FB15k. MINERVA is the latest of the two techniques and table 5 in [Das et al., 2018] shows it to be better than DeepPath and hence in the following we compare against only MINERVA. On FB15k-237, which is a smaller version of FB15k with 237 relations, MINERVA reports HITS@1 of 0.217 (table 4 in [Das et al., 2018]) whereas our underlying model Type-Distmult reports significantly higher HITS@1 of 0.293 (table 3 in [Jain et al., 2018]). [Das et al., 2018] doesn't report performance of MINERVA on full FB15K, though their code is publicly available. In our preliminary experiments, we trained their code on FB15K using the same hyperparameters as that for FB15K-237 provided by the authors and we achieved HITS@1 of only 0.25 and upon optimizing it further, we could reach only 0.29. On the other hand, [Jain et al., 2018] reports HITS@1 of 0.66 for the Type-Distmult TF model used in our experiments.

E.1.2 Pedagogical Approach

The approach as mentioned in [Gusmão et al., 2018] does not scale to the size of FB15K dataset. They perform their experiments on a toy FB13 dataset where they use only 13 relations of the entire FB15K dataset. We tried their methodology on FB15K but memory and storage requirements where intractable. According to our calculations, it would take

25 TB of storage and 20 days of compute, using the available code on an available system with 256 GB of RAM and 40 Intel(R) Xeon(R) CPU @ 2.80GHz cores.

E.2 Experiment Details

Hyperparameters: We use SGD optimizer, with a learning rate of 0.001, momentum of 0.9 and a batch size of 2048 for all our experiments. Table E.1 states the hyper parameters used for each setting.

Dataset	Setting	ρ_{neg}	ρ_{pos}	λ_2	λ_1
FB15K	Un-supervised	1	0.1	0	0
	Semi supervised	1	0.125	1	1
YAGO	Un-supervised	0.5	0.2	0	0
	Semi supervised	0.25	0.25	1	1

Table E.1: Hyper-parameters used for various settings

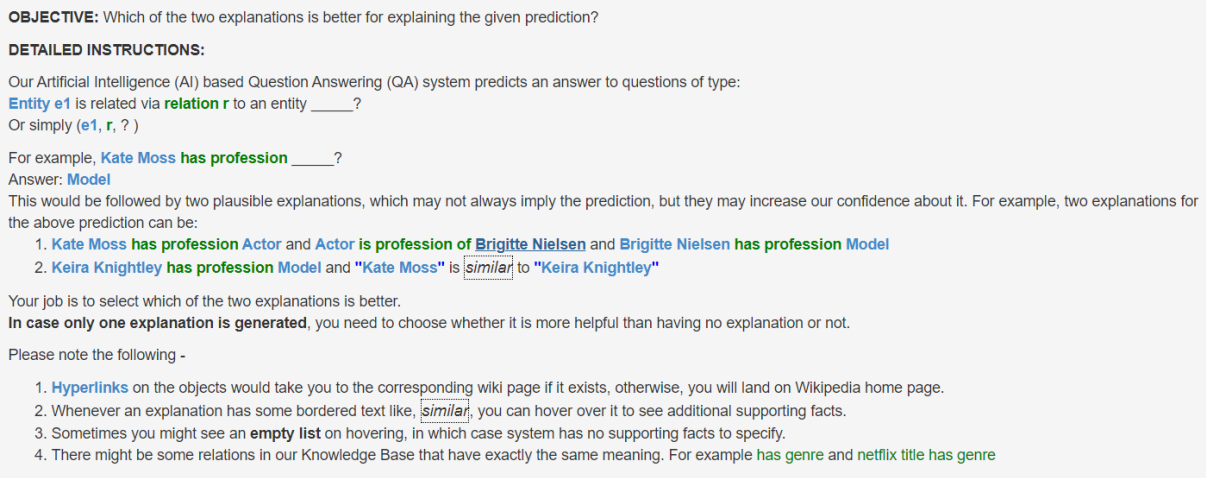
Computing Infrastructure and Time Analysis: For generating feature vectors corresponding to each template, we used a machine with 16 Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz cores and 128 GB of RAM. No GPU memory was required. It required about 5 hours to preprocess and generate the feature vectors for the train and dev sets. For training the two layer MLP in SM, we used a GPU with 16 GB memory, though the GPU memory required is less than 1 GB (for a batch size of 2048). The module gets trained in about 20 minutes.

At test time, to explain a prediction in OxKBC most of the time is spent in extracting features for different templates. For FB15K, it takes on average around 500 milliseconds per test sample to extract template features using a single core of an Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz. Once features are extracted, it takes only 0.38 milliseconds to select the best template using the selection module. On the other hand, in rule mining, once the rules are mined, it takes only 2.04 milliseconds to select the best rule for a given test sample. Even though at test time, OxKBC is much slower than rule mining, we believe it is still fast enough to be usable, especially for a human interacting with an explanation interface. We also note that this extra inference time comes with the advantage of improved quality of explanations.

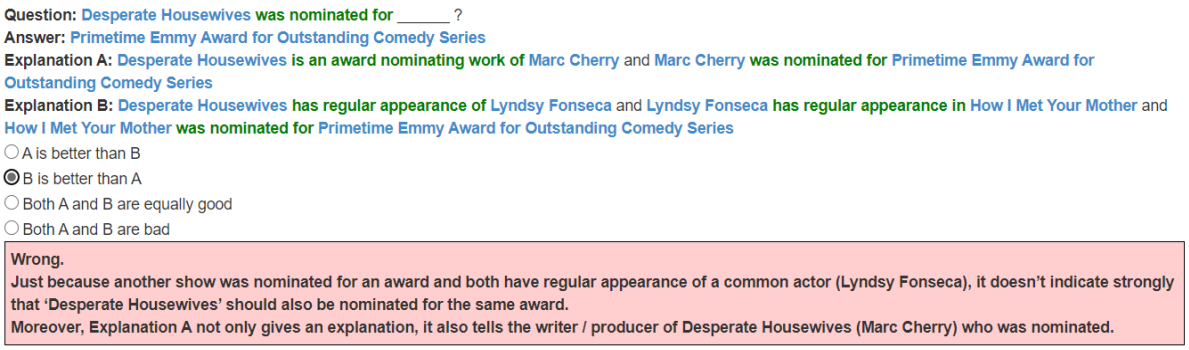
E.3 Mechanical Turk Experiment Details

Selecting Workers: To make sure that the workers really understand the task, we create an interactive tutorial of 6 questions in which users can select different options and get feedback about each of them. Figure E.1 shows a screenshot from the tutorial. To make our tutorial unbiased, we select questions which expose the good and bad sides of both the systems, and at no point are the workers told which explanation comes from which system. On completing the tutorial, they perform the task of comparing the two explanations on a small set of 10 gold questions. For each question, they are also asked to give a subjective reason for their choice. We analyze their answers quantitatively and their reasons qualitatively and manually cherry-pick the diligent and reasonable workers. All subsequent experiments are done with these quality workers.

Human Intelligence Task: Figure E.1 shows a screenshot of the task given to workers. We ask them to chose which out of the two explanations are better. They can select or reject both the explanations as well. In case only one system gives an explanation, workers choose if it is better than no explanation or not. In order to explain the task better, we create an interactive tutorial where workers can see why some explanation is inferior compared to its counterpart. For explaining entity similarity or high frequency of entity, we also generate a context of why two entities are similar or why is some entity frequent and the user can hover over bordered boxes to see the additional facts.



(a) Instructions provided to workers



(b) The tutorial has 6 of such questions which we use to tell workers why some explanations are better than others.

(c) Example query with frequent entity context

Figure E.1: Human Intelligence Task on Amazon Mechanical Turk

List of Publications

This thesis is based on the following publications:

1. Nandwani, Yatin, Abhishek Pathak, Mausam and Parag Singla. "A Primal Dual Formulation For Deep Learning With Constraints." NeurIPS (2019).

2. Nandwani, Yatin, Ankesh Gupta, Aman Agrawal, Mayank Singh Chauhan and Parag Singla. "OxKBC: Outcome Explanation for Factorization Based Knowledge Base Completion." AKBC (2020).

3. Nandwani, Yatin, Deepanshu Jindal, Mausam and Parag Singla. "Neural Learning of One-of-Many Solutions for Combinatorial Problems in Structured Output Spaces." ICLR (2021).

4. Nandwani, Yatin, Vidit Jain, Mausam and Parag Singla. "Neural Models for Output-Space Invariance in Combinatorial Problems." ICLR (2022).

5. Nandwani, Yatin, Rishabh Ranjan, Mausam and Parag Singla. "A Solver-free Framework for Scalable Learning in Neural ILP Architectures." NeurIPS (2022).

List of other publications during graduation:

1. Nandwani, Yatin, Ankit Anand and Parag Singla. "Lifted Inference for Faster Training (LIFT) in End-to-End Neural-CRF Models." Relational Representation Learning Workshop, NeurIPS (2018).

2. Leyton-Brown, Kevin, Mausam, Yatin Nandwani, Hedayat Zarkoob, Chris Cameron, N. Newman and Dinesh Raghu. "Matching Papers and Reviewers at Large Conferences." ArXiv abs/2202.12273 (2022).

Biography

Yatin Nandwani is a research scientist at IBM Research, India. He is part Conversational AI team that focuses on the business problem of using LLMs as enterprise chatbots. He completed his PhD in 2024 in the area of Machine Learning and Artificial Intelligence, guided by Mausam and Parag Singla at the Computer Science and Engineering Department, Indian Institute of Technology Delhi. Prior to joining the PhD program, he worked in the quantitative finance industry for a decade. He completed his five–year integrated M.Tech in Mathematics and Computing from the Indian Institute of Technology Delhi in 2007.

Currently, his research interest lies in addressing key challenges faced when using LLMs as enterprise chatbots. In particular, he focuses on problems such as mitigating hallucinations in text generated by LLMs, crafting domain-specific models utilizing domain-specific corpora, injecting domain knowledge in LLMs, preventing catastrophic forgetting during fine-tuning processes, and exploring methods to align LLMs with reward models and human preferences.

During his PhD, he worked in the broad area of neuro-symbolic AI, focusing on deep learning with implicit and explicit constraints. Specifically, he looked at how to train neural models for solving symbolic tasks. What are the different ways of learning better neural models by augmenting them with domain knowledge? How can we efficiently integrate differentiable symbolic solvers with neural networks for end-to-end training of neuro-symbolic architectures?