Inclusion of Symbolic Domain-Knowledge into Deep Neural Networks

THESIS

Submitted in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

by

Tirtharaj Dash
(ID No. 2016PHXF0421G)

Under the Supervision of
Ashwin Srinivasan

and

Co-supervision of
Sukanta Mondal

COMPUTER SCIENCE AND INFORMATION SYSTEMS
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI – 333 031 (INDIA)

July 2022

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI – 330 031 (INDIA)

CERTIFICATE

This is to certify that the thesis entitled "**Inclusion of Symbolic Domain-Knowledge into Deep Neural Networks**" submitted by **Tirtharaj Dash**, bearing student ID No. **2016PHXF0421G** for the award of Ph.D. degree of the institute, embodies original work done by him under our supervision.

Signature of the Supervisor : _____

Name : **ASHWIN SRINIVASAN**

Designation : Senior Professor
Department of CS & IS and APPCAIR
BITS Pilani, K.K. Birla Goa Campus

Place : Goa

Date : _____

Signature of the Co-supervisor : _____

Name : **SUKANTA MONDAL**

Designation : Associate Professor
Department of Biological Sciences
BITS Pilani, K.K. Birla Goa Campus

Place : Goa

Date : _____

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI – 330 031 (INDIA)

DECLARATION

I, **Tirtharaj Dash**, declare that this thesis entitled "**Inclusion of Symbolic Domain-Knowledge into Deep Neural Networks**", submitted by me under the supervision of **Ashwin Srinivasan** and co-supervision of **Sukanta Mondal** is a bonafide research work. I also declare that the work carried out in this thesis has not been submitted previously in part or in full to this university or any other university or institute for award of any degree.

Signature of the Student : _____

Name : **TIRTHARAJ DASH**

ID No. : 2016PHXF0421G

Place : Goa

Date : _____

THIS THESIS IS DEDICATED TO MY GRANDPARENTS:

Shri Satyanarayan Dash (Grandfather)
Smt Sirisha Dash (Grandmother)

They were, therefore I am

Acknowledgements

There are a number of people and organisations to thank for their direct or indirect contributions to my PhD journey.

My Supervisor, Professor Ashwin Srinivasan. For our joint work on some of the most exciting problems in Machine Learning, and also for teaching me how to 'Think'. I am certainly improving at this. Or at least I think I am. His support and motivation throughout this journey of my PhD have been immense, and I cannot acknowledge this enough. Because of him, I got to meet and work with some of the best researchers in India and abroad.

My Co-supervisor, Professor Sukanta Mondal. For introducing me to research at his ABC Lab and teaching me how to stay positive in difficult times during research. I have had opportunities to collaborate with him on some interesting problems in Computational Biology.

My PhD Committee. For their timely support at various steps of my PhD: Professors Bharat M. Deshpande, Vinayak Naik, Biju K. Raveendran, Angshuman Sarkar and Dr. Aditya Challa.

My Co-authors, both in and outside BITS. For the exciting research, we conducted together. I learnt much by working with: Professor Ross King, Dr. Oghenejokpeme Orhobor, Dr. Lovekesh Vig, Dr. Gautam Shroff, Ramya Hebbalaguppe, Dr. Arijit Roy, Dr. Ramprasad Joshi and Dr. A. Baskar.

My Collaborators. For providing access to data and background knowledge, used in this dissertation. I am deeply thankful to Dr. Gustav Šourek (Czech Technical University, Prague) for providing the dataset information. I thank the researchers at the DTAI, University of Leuven, for suggestions on how to use the background knowledge within DMax Chemistry Assistant™. I also thank Dr. Oghenejokpeme Orhobor and Professor Ross King for providing me with the initial set of background knowledge definitions.

My Department. For providing an open and friendly atmosphere for research and teaching. I am particularly grateful for the assistance given by the department

staff: Mr. Shreenivas Naik and Mr. Yellumaharaj Akalwadi. I feel fortunate to be a member of this department.

My Students. For the effort, energy and enthusiasm they brought to many aspects of my stay at BITS, including teaching, research and projects. I learnt more from them than they did from me.

My Friends. For their unwavering support during this PhD. I thank the "Lunch-Box" group, especially Dr. Anuradha V., Dr. A. Baskar and Dr. Ramprasad Joshi, for the beautiful time spent together and for their insightful and wise discussions on various aspects of life. I express my special thanks to Dr. Gunja Sachdeva for her constant support and help during the writing of this dissertation. My appreciation also goes out to Dr. Rakesh Ranjan Swain and Ayush Deep for being good friends, and for their encouragement and support during my PhD. I also thank my cat "KitCat", who has been one of my great companions during my difficult times.

My Grandparents. For providing me every support to chase my dreams. My grandfather wanted me to excel in everything I did, and I am trying to fulfil that dream. Since my childhood, my grandmother took care of me like a "mother". I miss them.

My Parents and Sister. For their unconditional love, care and support.

Tirtharaj
July 2022

I thankfully acknowledge the partial support received from the sponsored research grant EMR/2016/002766, DST-SERB, Government of India, awarded to my supervisor. I also wish to thank my department for funding some of my conference registrations and travels. I gratefully acknowledge the following organisations, for recognising my contributions: (a) Machine Learning Journal (Springer) and the program committee of ILP-2018 for adjudging my paper for the "Best Student Paper Award"; (b) The European Association for Artificial Intelligence (EurAI) for the Travel Grant in 2018; (c) The Department of Science and Technology, Government of India, for the AWSAR Award; (d) ICML 2021 for the Computational Biology Fellowship; and (e) Google Research India for selecting me for the Graduate Research Symposium. Finally, I sincerely thank the people and organisations developing the software used during this research: Prolog (Yap), MATLAB, Unix Shell, Keras, PyTorch, LaTeX, MathCha Editor and Overleaf.

T.D.

Abstract

This dissertation is concerned with techniques for inclusion of domain-knowledge into Deep Neural Networks (DNNs). We are primarily concerned with real-world scientific problems with the following characteristics: (a) Data are naturally graph-structured (relational), (b) The amount of data available is typically small, and (c) There is significant domain-knowledge, usually expressed in some logical form (rules, taxonomies, constraints and the like). Broadly, there are 3 different ways in which the domain-knowledge can be incorporated into a DNN: by changing the input representation, by changing the loss function, or by changing the model (structure and parameters). We propose techniques for the inclusion of domain-knowledge into DNNs that change the input representation. In particular, our principal contributions are as follows: (1) We study the inclusion of complex domain-knowledge into Multilayer Perceptrons (MLPs) using relational features and propositionalisation [LDG91]. We propose a utility-based stochastic sampling technique for drawing features from a large but countable space of relational features; (2) We propose a simplified technique called 'vertex-enrichment' for incorporating symbolic domain knowledge into deep neural networks that deal with graph-structured data, known as graph neural networks (GNNs); (3) We propose a systematic technique to incorporate symbolic domain-knowledge into GNNs using the method of inverse entailment [Mug95] available in Inductive Logic Programming (ILP); and (4) We construct a sequence generation system using a modular combination of two deep generative models and a discriminator model based on (3), and use this system for a problem of early-stage lead discovery in drug design. Our implementations are techniques that combine neural networks and symbolic representations, resulting in new neuro-symbolic models, such as: Deep Relational Machines (DRMs), Vertex-Enriched Graph Neural Networks (VEGNNs), Bottom-Graph Neural Networks (BotGNNs), and a modular end-to-end neuro-symbolic system for the generation of novel molecules for drug design. Our primary hypothesis is that inclusion of domain-knowledge can significantly improve the performance of a deep neural network. We conduct large-scale empirical testing of our hypothesis, using nearly 75 datasets in the broad area of drug discovery that consist of over $200,000$ relational data instances and with domain-knowledge containing about 100 relations. In all cases, our empirical evidence supports the primary hypothesis and encourages the inclusion of domain-knowledge into deep neural networks for prediction and explanation.

Contents

List of Figures

List of Acronyms

Adam	Adaptive Moment Estimation (an optimisation algorithm)
AI	Artificial Intelligence
Aleph	A Learning Engine for Proposing Hypotheses (an ILP system)
BotGNN	Bottom-Graph Neural Network
BK	Background Knowledge
CNN	Convolutional Neural Network
CONV	Convolution (used for a block or a layer)
DL	Deep Learning
DNN	Deep Neural Network
DRM	Deep Relational Machine
GNN	Graph Neural Network
ILP	Inductive Logic Programming
MDIE	Mode-Directed Inverse Entailment
ML	Machine Learning
MLP	Multilayer Perceptron
NCI	National Cancer Institute
NN	Neural Network
POOL	Pooling (used for a block or a layer)
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SMILES	Simplified Molecular-Input Line-Entry System
VAE	Variational Autoencoder
VEGNN	Vertex-Enriched Graph Neural Network
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

The history of "machines that learn" is almost as old as the history of modern Computer Science. Of course, they figure prominently in Alan Turing's famous 1950 paper on Computing Machinery and Intelligence [Tur50], but proposals for connectionist-based learning appear even earlier with the development of theories emulating biological learning by McCulloch and Pitts [MP43], and parameter update by Donald Hebb [Heb49] and implementation of perceptron [Ros57], enabling the 'training' of a single-layered model. Later, the method of back-propagating errors [RHW86] in a multi-layered connectionist architecture led to its dramatic usage in recognising handwritten ZIP codes [LBD+89]. The reincarnated term, used for connectionist architectures, is *Neural Networks* or *Deep Neural Networks* or *DNNs*, in short. This field has witnessed several ground-breaking discoveries such as neural network models for learning from sequential data [HS97], deep generative model [HOT06], greedy layer-wise pre-training of deep neural networks [BLPL07]. In 2012, ImageNet classification [DDS+09] by training a deep neural network by efficiently using graphics processing units (GPUs) created a new wave in the field [KSH12], and the rest is history.

At the time of writing this dissertation, deep neural networks are undergoing an unprecedented resurgence in interest as the tools of choice in machine learning. Although many of the techniques are not new, there are at least 3 different threads that are driving recent activity "deep learning", a term that was first introduced to machine learning community by [Dec86], and to neural networks community by [Aiz99], however, the presently prevailing usage of this term is due to the work by [HOT06]: First, deep learning has become more useful as the amount of data has increased; Secondly, the significant improvements of software and hardware architectures have allowed more complex deep learning models to be trained in less time and with reasonable efficiency; Thirdly, deep learning has solved complicated applications from all domains of science and engineering with increasing predictive accuracy over time. These three threads have enabled machine-learning "in-the-large", allowing us to consider its application to problems about which we know little, but for which we are able to obtain large amounts of data (or at

least, we have large amounts of data for problems related to the one we want to solve).

This dissertation is, however, about the use of modern-day deep learning in a different setting involving knowledge-rich problems. An important example is the area of Artificial Intelligence (AI) for Science. This is concerned with the use of AI methods to accelerate our understanding of the natural world, and to assist the application of this understanding to the development of areas of engineering, medicine, healthcare, agriculture, environment and so on. An example of this is the development of Robot Scientists [KWJ+04]. In this, advanced AI machinery is coupled with robotic wet-lab hardware to execute the classic scientific "hypothesize-and-test" cycle characteristic of scientific experimentation. The Robot Scientist is able to achieve, in some measure, the following: (1) identifying the best explanation for a prediction based on what is known; (2) suggesting hidden variables or mechanisms which could improve the prediction; and (3) proposing experiments to test the hypotheses. While further ambitious plans exist for Robot Scientists for completely automating scientific discoveries [Kit16], the current use of machine learning for scientific discoveries remains as providing assistance to the scientist(s) in the loop. An example of such a collaborative system is in Figure 1.1, which intends to describe a setting where a scientist is attempting to understand some (natural) phenomena. They conduct experiments and obtain a set of observations, which are provided as data to a machine learning (ML) engine. Additionally, the scientist also provides the machine learning engine with domain-expertise that maybe relevant to understanding the phenomenon being studied.

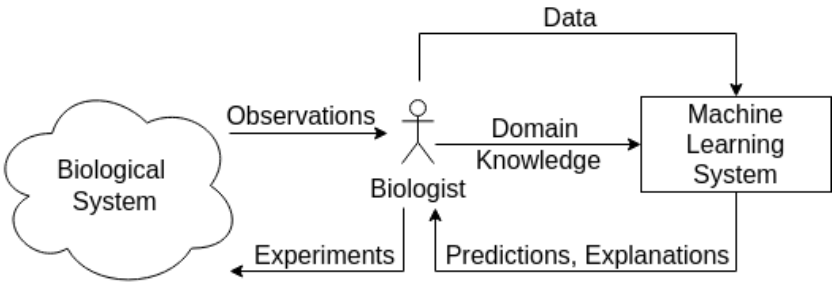


Figure 1.1: An example of using present day machine learning systems as assistance for scientific discoveries. The scientist-in-the-loop is a biologist. The biologist conducts experiments in a biological system, obtains experimental observations. The biologist then extracts data that can be used to construct machine learning model(s). Additionally, the machine learning system has access to domain-knowledge that can be obtained from the biologist. The machine learning system then conveys its predictions and explanations to the biologist.

In the diagram shown above, it is unclear how the domain information from an expert be encoded and provided while building a collaborative system. It is imperative, however, for such collaborative Human-and-AI systems to work effectively, we need at least the following: (1) We have to be able to tell the learner what we know, in a suitably precise

form; and (2) The machine has to be able to tell us what it has found, in a suitably understandable form. Point (1) concerns with some form of encoding of the available domain expertise, and point (2) concerns with some form of human-understandable explanation of the machine's prediction(s). While the remarkable recent successes of deep neural networks [Sch15] on a wide variety of tasks makes a substantial case for their use in model construction, it is not immediately obvious how either (1) or (2) should be done with deep neural networks. The research carried out in this dissertation is focused on providing some answers to (1), that is, incorporating domain information into deep neural networks. However, understanding models constructed by deep neural networks are a different area of intense research activity (see, for example, some recent surveys in this area: [Lip16, ADRDS+20]). This dissertation does not answer this aspect of research.

In the last few years, incorporating some form of domain-knowledge into learning has been emphasised strongly in the deep learning and reasoning community. Experts are stressing that incorporating domain-knowledge into deep neural networks could result in: (a) achieving highly general models and therefore could result in higher predictive performance than the models built only with available data [Mar18, BDR+20]; (b) resulting in robust reasoning systems [Mar20]. Some of these points have been investigated recently in a research report on the inclusion of domain-knowledge into deep learning [Stå21]. The report also highlights that providing domain information may not always be provided as input to a model directly but can be represented as some form of internal logic or external constraint. Furthermore, the inclusion of domain-knowledge in learning could solve some challenges in those scientific areas that require understanding data using human-machine collaboration, such as in medical diagnosis and drug discovery. All the above benefits could be achieved even if available data is scarce and the machine is provided with human-knowledge of the domain encoded in a sufficiently precise form. It is also unsurprising that a recent report on AI for Science [STN+20] identifies the incorporation of domain-knowledge as the first of the 3 Grand Challenges in developing AI systems:

> "ML and AI are generally domain-agnostic... Off-the-shelf practice treats [each of these] datasets in the same way and ignores domain knowledge that extends far beyond the raw data itself—such as physical laws, available forward simulations, and established invariances and symmetries—that is readily available... Improving our ability to systematically incorporate diverse forms of domain knowledge can impact every aspect of AI..."

Motivated by the above-discussed points, we aim to pursue this research direction, and propose new and effective methods that could help to construct robust and accurate scientific assistants. Specifically, our interest is in problems where both data and domain-knowledge are uniformly represented in first-order logic. In what follows, we first describe the importance of domain-knowledge and provide a brief overview of the approaches in

this area. We then outline some major difficulties in inclusion of domain-knowledge into deep neural networks. Next, we state the main contributions of this dissertation and provide detail on how the chapters in this dissertation are organised.

1.1 The Importance of Domain-Knowledge

Even though the idea of incorporating domain-knowledge into learning seems more pronounced recently; this is not entirely a new direction in machine learning. The earliest work in incorporating domain-knowledge into various AI methods, both symbolic and connectionist dates back to the late 1960s or early 1970s. Probably, the oldest research in this direction is by Plotkin in his doctoral thesis on "Automatic Methods of Inductive Inference" [Plo72] in which he defined generalisation from experience, relative to a body of knowledge. Michalski attempted to develop learning systems in which concepts were expressed in augmented predicate calculus. For example, Michaski's INDUCE program [Mic73, Mic80] augments the data rules input by the user with the inference rules in the domain-knowledge resulting in the construction of new rules. CONFUCIUS is a program that works with the principle that descriptions of inputs could be improved by learning domain-knowledge [CS82]. In the implementation of Marvin [SB86], a successor of CONFUCIUS, Sammut and Banerji discuss how existing (domain) concepts can be learned and re-used to learn new (domain) concepts. Here the domain concept is provided by a domain-expert (human) in an indirect fashion: the domain-expert shows the learner (Marvin) a positive example of the concept (the *target* concept) to be learned. The description of the example represents the concept which contains only one object (the shown example), and Marvin's task is to generalise the initial example in a manner that it describes all the positive examples of the target concept and none of the negative examples. Duce [Mug87] is a machine learning system that uses six transformation operators to construct high-level domain-features for a set of examples objects from their descriptions. Duce has its successor in CIGOL [MB88] where, given examples of a high-level predicate, CIGOL generates related sub-concepts to be named by a human-expert. In the process, CIGOL uses existing clauses in the database to be treated as domain-knowledge while constructing new sub-concepts. Such ideas of learning from examples and domain-knowledge comes under the umbrella term coined in 1991, called "Inductive Logic Programming (ILP)" [Mug91]. ILP provides a systematic learning method to induce hypotheses from data and background knowledge. Here data, background knowledge and hypotheses are uniformly represented in first-order logic. In the area of neural networks, knowledge-based artificial neural network (KBANN) [TSN90, TS94] is, to the best of our knowledge, the oldest method that incorporates domain-knowledge encoded as simple propositional rules.

In the next chapter, we conduct a comprehensive survey of a wide range of studies

that incorporate some form of domain-knowledge into deep neural networks. In there, we have studied that any form of domain-knowledge constrains either the structure or the parameters of a deep network. Therefore, we refer to these as "constraints". We will restrict ourselves to domain-knowledge that can be represented either as logical or as numerical constraints. Under logical constraints, we consider domain-knowledge that is represented in propositional logic, predicate logic, including binary or more generally n-ary relations, canonical normal forms, and program primitives. The numerical constraints are represented by priors on the model structure and parameters, leading to the introduction of additional terms in the loss function of the model. So, we restrict the review here to research that involves these forms of background knowledge. We note that there is a class of hybrid systems combining neural and logical systems (see for example, [GBG12, RDMM20]) that attempts to emulate logical inference or represent logical concepts. Although this hybrid approach is relevant to our present research conducted in this dissertation, incorporation of logically encoded domain-knowledge into deep neural networks is more specific than just constructing a hybrid system. We also categorise the approaches of inclusion of domain-knowledge into a deep neural network as follows: (1) transforming the data representation; (2) transforming the loss function; and (3) transforming the model (either its structure or parameters). In a sense, this progression reflects a graded increase in the complexity of changes involved. More detailed discussions on the aspects discussed here are provided in Chapter 2.

1.2 Difficulties in Inclusion of Domain-Knowledge into Deep Neural Networks

There are many difficulties associated with the inclusion of domain-knowledge into deep neural networks. While these difficulties are elaborated in our next chapter, we provide a non-exhaustive list of some of the purely implementational aspects, as follows:

- There is no standard framework for translating logical constraints to neural networks.
- Any form of logical constraint is not differentiable.
- Logical constraints can introduce cyclic dependencies if they are directly used to construct a deep neural network structure.
- The process of introducing a loss term often results in a hard optimisation problem (sometimes constrained) to be solved. Furthermore, it may require additional mathematical tools for a solution that can be implemented practically.

- Deep neural network structures constrained via logical domain-knowledge may not always be scalable to large datasets.

- While numerical constraints are introduced into the loss function or as regularisation terms, it is not straightforward and sometimes not very intuitive.

- A vast majority of the studies on incorporating some form of domain-knowledge into a deep neural network, the data considered is often represented as a numeric feature vectors or, in general, tensors. Adopting the underlying technique of learning from data that has relational structures is not straightforward. One example of a problem with high scientific value where data is relational is drug discovery, where each data instance is a molecular compound [KMSS96].

None of the difficulties listed above is concerned with the broader conceptual question of how domain-knowledge is to be acquired and represented in a machine-friendly form in the first place. In this dissertation, we will not be addressing this conceptual difficulty. Instead, we will focus on resolving some of the implementational issues for problems for which domain-knowledge is already available in some machine-readable form. In particular, we will focus our study on problems where domain-knowledge is encoded as statements in a subset of first-order logic and is about data with some relational structures.

1.3 Contributions of this Dissertation

In this dissertation, we are primarily concerned with real-world scientific problems with the following characteristics: (a) Data are naturally graph-structured (relational), (b) The amount of data available is typically small, and (c) There is significant domain-knowledge, usually expressed in some logical form (rules, taxonomies, constraints and the like). Below we outline the principal contributions made by this dissertation.

Conceptual. An approach to the stochastic selection of "relational features" as a mechanism of inclusion of domain-knowledge into multilayer perceptrons; An approach for simplified inclusion of domain-knowledge into graph-based neural networks for domain-knowledge that is represented in the form of hyperedges in a graph; An approach for complete inclusion of domain-knowledge into graph-based neural networks, through the use of ideas from mode-directed inverse entailment;

Implementational. Techniques that combine deep neural networks and symbolic representations resulting in the implementation of neuro-symbolic learners such as: Deep Relational Machines (DRMs), Vertex-Enriched Graph Neural Networks (VEGNNs),

Bottom-Graph Neural Networks (BotGNNs); and a modular end-to-end neuro-symbolic system that uses a BotGNN as a system component for generation of novel molecules for drug-design;

Applications. Investigating the applications of implementations mentioned above on large-scale carcinogenicity problems and lead-discovery problems relevant to drug design.

A more detailed breakup of the contributions is as follows:

1. We construct multilayer perceptrons (MLPs) from relational data and background knowledge using propositionalisation [LDG91], a technique in ILP that transforms relations into a simpler format, typically a feature-vector or attribute-value representation. Here, we propose a utility-based stochastic sampling method to draw relational features from a large and countable discrete feature space.

2. We propose a simplified technique called 'vertex-enrichment' for incorporating symbolic domain-knowledge into a class of deep neural networks that deal with graph-structured data, known as graph neural networks. We also demonstrate how incorporating higher-order n-ary relations discovered by ILP can further improve the predictive performance of vertex-enriched graph neural networks.

3. We propose a systematic technique to incorporate symbolic domain-knowledge into graph neural networks using the method of inverse entailment available in ILP [Mug95].

4. We construct a conditional deep generative model via the inclusion of domain-knowledge by proposing a methodology that consists of a collaborative combination of two generators and a discriminator proposed in the contribution 3 above. We study the application of this collaborative model for the problem of early-stage lead discovery in drug design.

1.4 Organisation of the Dissertation

The remainder of this dissertation is organised as follows:

In Chapter 2, we provide a review of the categories and subcategories of domain-knowledge, and various methods of their inclusion into deep neural networks (DNNs). We further examine some challenges in adopting the underlying techniques to deal with relational data and symbolic domain-knowledge.

In Chapter 3, we construct DNNs by the standard method of propositionalisation of relational features, constructed using ILP using data and background knowledge. Each

relational feature encodes some logical description of the data instance. The propositionalisation technique transforms a set of relational features into a simple format such as numeric feature vectors that can be used as inputs for a deep neural network. We propose a guided sampling strategy to sample relational features from a large countable discrete feature space. The class of DNNs investigated in this chapter are called Multilayer Perceptrons (MLPs).

Chapter 4 proposes a simplification method called "vertex-enrichment" to incorporate symbolic domain-knowledge into deep neural networks suitable for graph-structured (relational) data, called graph neural networks (GNNs). In this work, each domain relation is treated as a hyperedge. The proposed method of vertex-enrichment allows enriching the feature associated with every vertex of the graph that appears in a hyperedge. Here we will show that the vertex-enrichment technique results in simplified inclusion of domain-knowledge into GNNs.

In Chapter 5, we propose a general technique for the inclusion of multi-relational domain-knowledge into GNNs, using the method of inverse entailment [Mug95] developed within the area of Inductive Logic Programming (ILP). ILP constructs a bottom-clause (the most specific explanation about a data instance) from data and background knowledge. Here, we construct an equivalent graph representation corresponding to a bottom-clause, and show how a GNN can operate on this graph. The technique proposed here provides a framework for the complete inclusion of relational information into a GNN.

In Chapter 6 we extend our study in Chapter 5 to construct a deep generative model for molecule generation. That is, we propose a modular system that uses two deep generative models and a GNN-based discriminator constructed in Chapter 5. The technique proposed here allows indirect inclusion of relational information in the domain-knowledge into the conditional generation of novel molecules.

In all the chapters, we would aim to investigate our primary hypothesis that the inclusion of domain-knowledge improves the performance of deep neural networks. To this end, our investigation would be in the broad of drug discovery. Specifically, we conduct large-scale empirical testing of the resulting DNN models in the Chapter 3 to Chapter 5, using nearly 75 datasets that consists of over 200,000 relational data instances and with domain-knowledge containing about 100 relations. In Chapter 6, we evaluate our constructed system for conditional generation of novel small molecules to act as inhibitors for a well-studied target protein. We outline the main findings of this dissertation in Chapter 7 and provide details of future research directions that could spin out from our present work.

Some additional details concerning the technical background, such as DNNs, ILP, are provided in Appendix A. In Appendix B we provide details of the additional experiments that are conducted to evaluate some secondary research questions in our chapters.

Chapter 2

Literature Review*

In this chapter, we present a survey of techniques in which existing domain-knowledge is included when constructing models with deep neural networks.

In this survey, we restrict the studies on the incorporation of domain-knowledge into neural networks with one or more hidden layers. If the domain-knowledge expressed in a symbolic form (for example, logical relations that are known to hold in the domain), then the broad area of hybrid neural-symbolic systems (see for example, [GBG12, RDMM20]) is clearly relevant to the material in this chapter. However, the motivation driving the development of hybrid systems is much broader than this chapter, being concerned with general-purpose neural-based architectures for logical representation and inference. Here our goals are more modest: we are looking at the inclusion of problem-specific information into machine-learning models of a kind that will be described shortly. We refer the reader to [BGB+17] for reviews of work in the broader area of neural-symbolic modelling. More directly related to the survey in this chapter is the work on "informed machine learning", reviewed in [vRMB+21]. We share with this work the interest in prior knowledge as an important source of information that can augment existing data. However, the goals of that paper are more ambitious than here. It aims to identify categories of prior knowledge, using as dimensions: the source of the knowledge, its representation, and its point of use in a machine-learning algorithm. In this survey, we are only concerned with some of these categories. Specifically, in terms of the categories in [vRMB+21], we are interested in implicit or explicit sources of domain-knowledge, represented either as logical or numeric constraints and used in the model-construction stage by DNNs. Informal examples of what we mean by logical and numerical constraints are shown in Figure 2.1. In general, we will assume logical constraints can, in principle, be represented as statements in propositional logic or predicate logic. Numerical constraints will be representable, in

*The content of this chapter is based on the following:
T. Dash, S. Chitlangia, A. Ahuja, A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks", *Nature Scientific Reports*, 2022; https://doi.org/10.1038/s41598-021-04590-0.

principle, as terms in an objective function being minimised (or maximised), or prior distributions on models. We believe this covers a wide range of potential applications, including those concerned with scientific discovery.

For inhibiting this protein:	The model should follow that:
The presence of a peroxide	$p(y = 1\|\mathbf{x}) \geq 0.9$
bridge is relevant.	$p(y = 0\|\mathbf{x}) \leq 0.1$
The target site is	Initial weights should be
at most 20Å.	$3n - 2.3$ [TSN90]
(a)	**(b)**

Figure 2.1: Informal descriptions of (a) logical; and (b) numerical constraints.

2.1 Focus of this Review

We adhere to the following informal specification for constructing a deep neural network: given some data D, a structure and parameters of a deep neural network (denoted by π and $\boldsymbol{\theta}$, respectively), a learner $\mathcal{L}$ attempts to construct a neural network model M that minimises some loss function L. Figure 2.2 shows a diagrammatic representation. Note that: (a) we do not describe how the learner $\mathcal{L}$ constructs a model M given the inputs. But, it would be normal for the learner to optimise the loss L by performing an iterative estimation of the parameters $\boldsymbol{\theta}$, given the structure π; and (b) we are not concerned with how the constructed deep model M will be used. However, it suffices to say that when used, the model M would be given one or more data-instances encoded in the same way as was provided for model-construction.

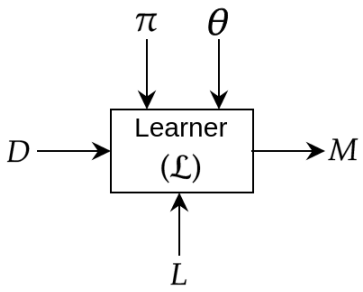


Figure 2.2: Construction of a deep neural network model M from data (D) using a learner ($\mathcal{L}$). We use π to denote the structure (organisation of various layers, their interconnections, etc.) and $\boldsymbol{\theta}$ to denote the parameters (synaptic weights) of the deep neural network. L denotes the loss function (for example, cross-entropy loss in case of classification).

In the literature, domain-knowledge–also called background knowledge–does not appear to have an accepted definition, other than that, it refers to information about the

problem. This information can be in the form of relevant features, concepts, taxonomies, rules-of-thumb, logical constraints, probability distributions, mathematical distributions, causal connections and so on. In this chapter, we use the term "domain-knowledge" to refer to problem-specific information that can directly be translated into alterations to the principal inputs of Figure 2.2. That is, by domain-knowledge we will mean problem-specific information that can change: (1) The input data to a deep neural network; (2) The loss-function used; and (3) The model (that is, the structure or parameters) of the deep neural network. In a sense, this progression reflects a graded increase in the complexity of changes involved. Figure 2.3 tabulates the principal implications of this position for commonly-used deep learning architectures.

The rest of the chapter is divided into multiple sections: We start by discussing some existing techniques for inclusion of domain-knowledge by augmenting or transforming inputs. Then we focus on some popular loss functions that are used to drive the inclusion of domain-knowledge during training of a deep neural network. We describe various biases on model parameters and changes to the structure of deep neural networks. In summary, we outline some major challenges related to the inclusion of domain-knowledge in the ways we describe and where the contributions of this lie in this area of research.

2.2 Transforming the Input Data

One of the prominent approaches to incorporate domain-knowledge into a deep neural network is by changing inputs to the network. Here, the domain-knowledge is primarily in symbolic form. The idea is simple: If a data instance could be described using a set of attributes that not only includes the raw feature-values but also includes more details from the domain, then a standard deep neural network could then be constructed from these new features. A simple block diagram in Figure 2.4 shows how domain-knowledge is introduced into the network via changes in inputs. In this survey, we discuss broadly two different ways of doing this: (a) using relational features, mostly constructed by a method called propositionalisation [LDG91] using another machine learning system (for example, Inductive Logic Programming) that deals with data and background knowledge; (b) without propositionalisation.

2.2.1 Propositionalisation

The pre-eminent form of symbolic machine learning based on the use of relations in first-order logic is Inductive Logic Programming (ILP) [Mug91], which has an explicit role for domain-knowledge being incorporated into learning. The simplest use of ILP [Mug91] to incorporate n-ary relations in domain-knowledge into a neural network relies on techniques that automatically "flatten" the domain-knowledge into a set of domain-specific

Arch.	Domain-Knowledge Effect		
	Transform Input Data	**Transform Loss Function**	**Transform Model**
MLP	Reformulate feature-representation	(For all architectures) Reformulate regularisation term; Addition of syntactic or semantic constraints with associated penalties; Differential costs of decisions	Changes in layers, hidden units
CNN	Reformulate spatial-representation		As with MLPs, plus changes to connections between units; changes convolution filters
RNN, Transformer	Reformulate sequence-representation		As with MLPs, plus possible changes to attention-mechanism
GNN	Reformulate graph-representation		As with MLPs, plus changes to graph-convolution

Figure 2.3: Some implications of using domain-knowledge for commonly-used deep neural network architectures. Here MLP stands for Multilayer Perceptron, CNN stands for Convolutional Neural Network, RNN stands for Recurrenural Network and GNN stands for Graph Neural Network. MLPs, CNNs and RNNs are now commonplace architectures for deep neural networks and detailed descriptions can be found in any standard textbook (for example, [BGC17, ZLLS21]). GNNs are increasingly the DNN model of choice for dealing with graph-based data, and a good description can be found in [Ham20]. In this dissertation, we will be mainly concerned with MLPs and GNNs: the details required are in Appendix A.

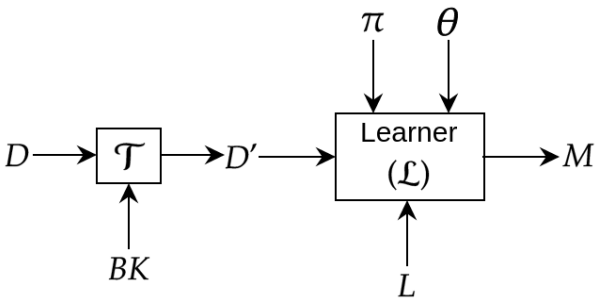


Figure 2.4: Introducing background knowledge into deep neural network by transforming data. $\mathcal{T}$ is a transformation block that takes input data D, background knowledge (BK) and outputs transformed data D' that is then used to construct a deep model using a learner $\mathcal{L}$.

relational features. Although not all DNNs require data to be a set of feature-vectors, this form of data representation is long-standing and still sufficiently prevalent. In logical terms, we categorise feature-based representations as being encodings in propositional logic. The reader would point out, correctly, that feature-values may not be Boolean. This is correct, but we can represent non-Boolean features by Boolean-valued propositions (for example, a real-valued feature f with value 4.2 would be represented by a corresponding Boolean feature f' that has the value 1 if $f = 4.2$ and 0 otherwise). With the caveat of this rephrasing, it has, of course, been possible to provide domain-knowledge to neural networks by employing domain-specific features devised by an expert. However, we focus here on ways in which domain-knowledge encoded as rules in propositional logic and first-order logic has been used to construct the input features for a deep neural network. Techniques for automatic construction of Boolean-valued features from relational domain-knowledge have a long history in the field of ILP [Md94, MDRP+12, CDM20], originating from the LINUS [LDG91]. Often called *propositionalisation*, the approach involves the construction of features that identify the conditions under which they take on the value 1 (or 0). For example, given (amongst other things) the definition of benzene rings and of fused rings, an ILP-based propositionalisation may construct the Boolean-valued feature that has the value 1 if a molecule has 3 fused benzene rings, and 0 otherwise. The values of such Boolean-valued features allow us to represent a data instance (like a molecule) as a Boolean-valued feature-vector, which can then be provided to a neural network. There is a long history of propositionalisation: see [KLF01] for a review of some of early use of this technique, and [LSR20, VSBV17] who examine the links between propositionalisation and modern-day use of embeddings in deep neural networks. More clearly, the authors examine that both propositionalisation and embedding approaches aim at transforming data into tabular data format while they are being used in different problem settings and contexts. One recent example of embedding is demonstrated in [GRS+21] where the authors use different text-embedding approaches such as sentence encoder [CYK+18] and GPT2 [RWC+19] to transform textual domain-knowledge into embedding vectors.

A direct application of propositionalisation, demonstrating its utility for deep neural networks, has been its use in Deep Relational Machines (DRMs: [Lod13]). A DRM is a deep fully-connected neural network with Boolean-valued inputs obtained from propositionalisation by an ILP engine.

The idea of propositionalisation also forms the foundation for a method known as 'Bottom Clause Propositionalisation (BCP)' to propositionalise the literals of a most-specific clause, or "bottom-clause" in ILP. Given a data instance, the bottom-clause is the most-specific first-order clause that entails the data instance, given some domain-knowledge. Loosely speaking, the most-specific clause can be thought of "enriching" the data instance with all domain relations that are true, given the data instance. The construction of such most-specific clauses and their subsequent use in ILP was introduced in [Mug95]. CILP++ [FZG14] uses bottom-clauses for data instances to construct feature-vectors for neural networks. This is an extension to CIL^{2}P in which the neural network uses recurrent connections to enforce the background knowledge during the training [GZ99].

Propositionalisation has conceptual and practical limitations. Conceptually, there is no variable-sharing between two or more first-order logic features. That is, all useful compositions have to be pre-specified. Practically, this makes the space of possible features extremely large: this has meant that the feature-selection has usually been done separately from the construction of the neural network. In this context, another work that does not employ either propositionalisation or network augmentation considers a combination of symbolic knowledge represented in first-order logic with matrix factorization techniques [RSR15]. This exploits dependencies between textual patterns to generalise to new relations.

Recent work on neural-guided program synthesis also explicitly includes domain-specific relations. Here programs attempt to construct automatically compositions of functional primitives. The primitives are represented as fragments of functional programs that are expected to be relevant. An example of neural-guided program synthesis that uses such domain-primitives is DreamCoder [EMM$^+$18, EWN$^+$20]. DreamCoder receives as inputs the partial specification of a function in the form of some inputs–output pairs, and a set of low-level primitives represented in a declarative language. Higher-level domain-concepts are then abduced as compositions of these primitives via a neurally-guided search procedure based on a version of the Bayesian "wake-sleep" algorithm [HDFN95]. The deep neural networks use a (multi-hot) Boolean-vector encoding to represent functional compositions (a binary digit is associated with each primitive function, and takes the value 1 if and only if the primitive is used in the composite function).

There are some methods that do not use an explicit propositionalisation step, nevertheless, amount to re-formulating the input feature representation. In the area of "domain-adaptation" [BDBC$^+$10], "source" problems act as a proxy for domain-knowledge

for a related "target" problem.[1] There is a form of domain-adaptation in which the target's input representation is changed based on the source model. In [DQW+21], for example, a feature-encoder ensures that the feature representation for the target domain that is the same as the one used for the source.

2.2.2 Binary and n-ary Relations

An influential form of representing relational domain-knowledge takes the form *knowledge graph*, which are labelled graphs, with vertices representing entities and edges representing binary relations between entities. A knowledge graph provides a structured representation for knowledge that is accessible to both humans and machines [PSS20]. Knowledge graphs have been used successfully in variety of problems arising in information processing domains such as search, recommendation, summarisation [SPG19]. Sometimes the formal semantics of knowledge graphs such as domain ontologies are used as sources for external domain-knowledge [YLD+21]. We refer the reader to [HBC+20] to a comprehensive survey of this form of representation for domain-knowledge.

Incorporation of the information in a knowledge-graph into deep neural models–termed "knowledge-infused learning"–is described in [KGS19, SGKW19]. This aims to incorporate binary relations contained in application-independent sources (like DBPedia, Yago, WikiData) and application-specific sources (like SNOMED-CT, DataMed). The work examines techniques for incorporating relations at various layers of deep neural networks (the authors categorise these as "shallow", "semi-deep" and "deep" infusion). In the case of shallow infusion, both the external knowledge and the method of knowledge infusion are shallow, utilising syntactic and lexical knowledge in word embedding models. In semi-deep infusion, external knowledge is involved through attention mechanisms or learnable knowledge constraints acting as a sentinel to guide model learning. Deep infusion employs a stratified representation of knowledge representing different levels of abstractions in different layers of a deep learning model to transfer the knowledge that aligns with the corresponding layer in the learning process. Fusing the information in a knowledge-graph in this way into various level of hidden representations in a deep neural network could also allow quantitative and qualitative assessment of its functioning, leading to knowledge-infused interpretability [GFS21].

There have been some recent advances in introducing external domain-knowledge into deep sequence models. For instance, in [YLD+21], the authors incorporate domain-specific knowledge into the popular deep learning framework, BERT [DCLT19] via a

[1] Superficially, this is also the setting underlying *transfer* learning. However, the principal difference is that source and target problems are closely related to domain-adaptation, but this need not be the case with transfer learning. Transfer-learning also usually involves changes in both model-parameters and model-structure. Domain-adaptation does not change the model-structure: we consider these points in a later section.

declarative knowledge source like drug-abuse ontology. The model constructed here, called Gated-K-BERT, is used jointly for extracting entities and their relationship from tweets by introducing the domain-knowledge using an entity position-aware module into the primary BERT architecture. The experimental results demonstrate that incorporating domain-knowledge in this manner leads to better relation extraction as compared to the state-of-the-art. This work could fall within the category of semi-deep infusion as described in [KGS19]. [YZQ+19], in their study on learning from electronic health records show that the adjacency information in a medical knowledge graph can be used to model the attention mechanism in an LSTM-based RNN with attention. Whenever the RNN gets an entity (a medical event) as an input, the corresponding sub-graph in the medical knowledge graph (consisting of relations such as *causes* and *is-caused-by*) is then used to compute an attention score. This method of incorporating the medical relations into the RNN falls under the category of semi-deep knowledge infusion. While the above methods use the relational knowledge from a knowledge-graph by altering or adding an attention module within the deep sequence model, a recent method called KRISP [MCP+21] introduces such external knowledge at the output (prediction) layer of BERT. This work could be considered under the category of shallow infusion of domain-knowledge as characterised by [KGS19].

Knowledge graphs can be used directly by specialised deep neural network models that can handle graph-based data as input (graph neural networks, or GNNs). The computational machinery available in GNN then aggregates and combines the information available in the knowledge graph (an example of this kind of aggregation and pooling of relational information is in [SKB+18]). The final collected information from this computation could be used for further predictions. Some recent works are in [PKD+19, WZX+19], where a GNN is used for estimation of node importance in a knowledge-graph. The intuition is that the nodes (in a problem involving recommender systems, as in [WZX+19], a node represents an entity) in the knowledge-graph can be represented with an aggregated neighbourhood information with bias while adopting the central idea of aggregate-and-combine in GNNs. The idea of encoding a knowledge graph directly for a GNN is also used in Knowledge-Based Recommender Dialog (KBRD) framework developed for recommender systems [CLZ+19]. In this work, the authors treat an external knowledge graph, such as DBpedia [LIJ+15], as a source of domain-knowledge allowing entities to be enriched with this knowledge. The authors found that the introduction of such knowledge in the form of a knowledge-graph can strengthen the recommendation performance significantly and can enhance the consistency and diversity of the generated dialogues. In KRISP [MCP+21], a knowledge-graph is treated as an input for a GNN where each node of the graph corresponds to one specific domain-concept in the knowledge graph. This idea is a consequence of how a GNN operates: it can form more complex domain-concepts by propagating information of the basic domain-concepts along the edges in

the knowledge-graph. Further, the authors allow the network parameters to be shared across the domain-concepts with a hope to achieve better generalisation. We note that while knowledge-graphs provide an explicit representation of domain-knowledge in the data, some problems contain domain-knowledge implicitly through an inherent topological structure (like a communication network). Clearly, GNNs can accommodate such topological structure just in the same manner as any other form of graph-based relations (see for example: [ZWQ+19]).

2.3 Transforming the Loss Function

One standard way of incorporating domain-knowledge into a deep neural network is by introducing "penalty" terms into the loss (or utility) function that reflect constraints imposed by domain-knowledge. The optimiser used for model construction then minimises the overall loss that includes the penalty terms. Figure 2.5 shows a simple block diagram where a new loss term is introduced based on the background knowledge. We distinguish two kinds of domain constraints–syntactic and semantic–and describe how these have been used to introduce penalty terms into the loss function.

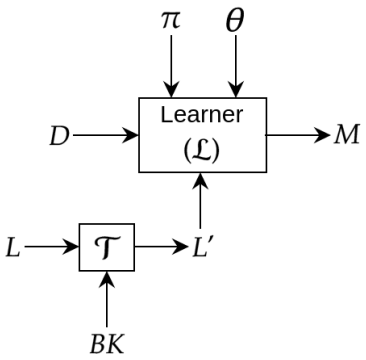


Figure 2.5: Introducing background knowledge into deep neural network by transforming the loss function L. $\mathcal{T}$ block takes an input loss L and outputs a new loss function L' by transforming (augmenting or modifying) L based on background knowledge (BK). The learner $\mathcal{L}$ then constructs a deep model using the original data D and the new loss function L'.

2.3.1 Syntactic Loss

The usual mechanism for introducing syntactic constraints is to introduce one or more *regularisation* terms into the loss function. The most common form of regularisation introduces penalties based on model complexity (number of hidden layers or number of parameters and so on: see, for example, [KGC17]).

A more elaborate form of syntactic constraints involves the concept of *embeddings*. Embeddings refer to the relatively low-dimensional learned continuous vector represen-

tations of discrete variables. Penalty terms based on "regularising embeddings" are used to encode syntactic constraints on the complexity of embeddings. [Fu95] was an early work in this direction, in which the authors proposed a strategy to establish constraints by designating each node in a Hopfield Net to represent a concept and edges to represent their relationships and learn these nets by finding the solution which maximises the greatest number of these constraints. [RBSR14] was perhaps the first method of regularising embeddings from declarative knowledge encoded in first-order logic. The proposal here is for mapping between logical statements and their embeddings, and logical inferences and matrix operations. That is, the model behaves as if it is following a complex first-order reasoning process, but operates at the level of simple vectors and matrix representations. [RSR15] extended this to regularisation by addition of differentiable loss terms to the objective-based on propositionalisation of each first-order predicate. Guo *et al.* [GWW+16] proposed a joint model, called KALE, which embeds facts from knowledge-graphs and logical rules, simultaneously. Here, the facts are represented as ground atoms with a calculated truth value in $[0, 1]$ suggesting how likely the fact holds. Logical rules (in grounded form) are then interpreted as complex formulae constructed by combining ground atoms with logical connectives, which are then modelled by fuzzy t-norm operators [Háj13]. The truth value that results from this operation is nothing but a composition of the constituent ground atoms, allowing the facts from the knowledge graph to be incorporated into the model.

[LS20] develop a method to constraint individual neural layers using soft logic based on massively available declarative rules in ConceptNet. [HBZ+18] incorporates first-order logic into low-dimensional spaces by embedding graphs nodes and represents logical operators as learned geometric relations in the space. [DRR16] proposed ordering of embedding space based on rules mined from WordNet and found it to better prior knowledge and generalisation capabilities using these relational embeddings. [LFJ+18] show that domain-based regularisation in loss function can also help in constructing deep neural networks with less amount of data in prediction problems concerned with cloud computing. In [TA18], a knowledge-based distant regularisation framework was proposed that utilises the distance information encoded in a knowledge-graph. It defines prior distributions of model parameters using knowledge-graph embeddings. They show that this results in an optimisation problem for a regularised factor analysis method.

2.3.2 Semantic Loss

Penalty terms can also be introduced on the extent to which the model's prediction satisfies semantic domain constraints. For example, the domain may impose specific restrictions on the prediction ("output prediction must be in the range $[3, 6]$"). One way in which such information is provided is in the form of domain-constraints. Penalty terms

are then introduced based on the number and importance of such constraints that are violated.

A recent work that is based on the loss function is in [XZF+18]. Here the authors propose a semantic loss that signifies how well the outputs of the deep neural network matches some given constraints encoded as propositional rules. The general intuition behind this idea is that the semantic loss is proportional to a negative logarithm of the probability of generating a state that satisfies the constraint when sampling values according to some probability distribution. This kind of loss function is particularly useful for semi-supervised learning as these losses behave like self-information and are not constructed using explicit labels and can thus utilize unlabelled data. In [THM21], a compositional framework called NeuroLog is proposed that can allow plug-and-play of a neural module and a symbolic module. The domain-constraints are provided to the neural module in an abductive manner. That is, the predictions (or the outputs) from the neural module are used as input for the symbolic module that encodes some logical constraints in first-order logic. If the predictions from the neural module were all correct, then the deduction from the symbolic module would match some desired output; otherwise, the symbolic module will provide an abductive feedback to the neural module via a semantic loss. The loss function dictates how close–semantically–are the outputs of the neural net to the formula found via abduction. The neural module is then trained to reduce this semantic loss.

[HML+16] proposed a framework to incorporate first-order logic rules with the help of an iterative distillation procedure that transfers the structured information of logic rules into the weights of neural networks. This is done via a modification to the knowledge-distillation loss proposed by Hinton et al. [HVD15]. The authors show that taking this loss-based route of integrating rule-based domain-knowledge allows the flexibility of choosing a deep neural network architecture suitable for the intended task.

In [FBDC+19], authors construct a system for training a neural network with domain-knowledge encoded as logical constraints. Here the available constraints are transferred to a loss function. Specifically, each individual logic operation (such as negation, and, or, equality etc.) is translated to a loss term. The final formulation results in an optimisation problem. The authors extract constraints on inputs that capture certain kinds of convex sets and use them as optimisation constraints to make the optimisation tractable. In the developed system, it is also possible to pose queries on the model to find inputs that satisfy a set of constraints. In a similar line, [MIM+19] proposed domain-adapted neural network (DANN) that works with a balanced loss function at the intersection of models based on purely domain-based loss or purely inductive loss. Specifically, they introduce a domain-loss term that requires a functional form of approximation and monotonicity constraints. Without detailing much on the underlying equations, it suffices to say that formulating the domain loss using these constraints enforces the model to learn not only

from training data but also in accordance with certain accepted domain rules.

Another popular approach that treats domain knowledge as 'domain constraints' is semantic based regularisation [DGS17, DRG17]. It builds standard multilayered neural networks (e.g. MLP) with kernel machines at the input layer that deal with continuous-valued features. The outputs of the kernel machines are then input to the higher layers implementing a fuzzy generalisation of the domain constraints that are represented in first-order logic. The regularisation term consisting of a sum of fuzzy generalisation of constraints using t-norm operations in the cumulative loss then penalises each violation of the constraints during the training of the deep neural network. [SLM20] inject domain-knowledge at training time via an approach that combines semantic based regularisation and constraint programming [RVBW06]. This approach uses the concept of 'propagators', which is inherent in constraint programming to identify infeasible assignments of variables to values in the domain of the variables. The role of semantic-based regularisation is to then penalise these infeasible assignments weighted by a penalty parameter. This is an example of constraints on inputs. In a similar line, [LZZ21] introduce domain-knowledge into a deep LSTM-based RNN at three different levels: constraining the inputs by designing a filtering module based on the domain-specific rules, constraining the output by enforcing an output range, and also by introducing a penalty term in the loss function.

A library for integrating symbolic domain-knowledge with deep neural networks was introduced recently in [FGU+21]. It provides some effective ways of specifying domain-knowledge, albeit restricted to (binary) hierarchical concepts only, for problems arising in the domain of natural language processing and some subset of computer vision. The principle of integration involves constraint satisfaction using a primal-dual formulation of the optimisation problem. That is: the goal is to satisfy the maximum number of domain constraints while also minimising the training loss, an approach similar to the idea proposed in [FBDC+19, MIM+19, SLM20].

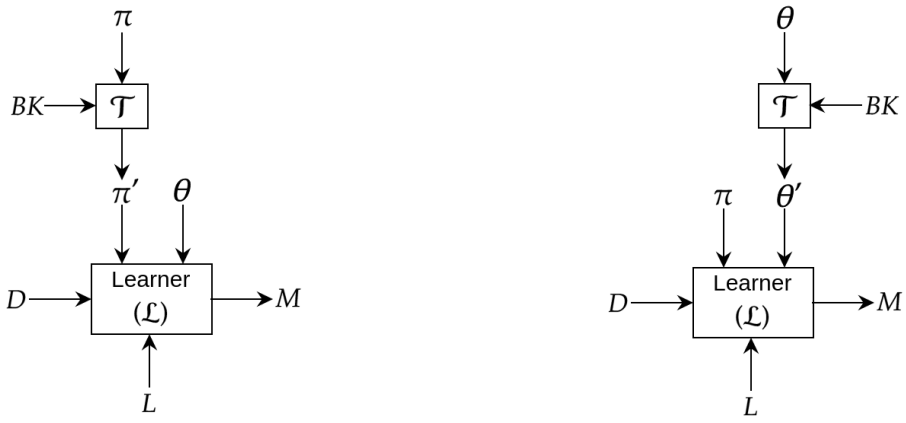
While adding a domain-based constraint term to the loss function may seem appealing, there are a few challenges. One challenge that is pointed out in a recent study [HKBG21] is that incorporating domain-knowledge in this manner (that is: adding a domain-based loss to the standard problem-specific loss) may not always be suitable while dealing with safety-critical domains where 100% constraint satisfaction is desirable. One way to guarantee 100% domain-constraint satisfaction is by directly augmenting the output layer with some transformations and then deriving a new loss function due to these transformations. These transformations are such that they guarantee the output of the network to satisfy the domain constraints. In this study, called MultiplexNet [HKBG21], the domain-knowledge is represented as a logical formula in disjunctive normal form (DNF) Here the output (or prediction) layer of a deep neural network is viewed as a multiplexor in a logical circuit that permits branching in logic. That is, the output of the network always satisfies one of the constraints specified in the domain-knowledge

(disjunctive formula).

The other form of semantic loss could be one that involves a human for post-hoc evaluation of a deep model constructed from a set of first-order rules. In this line, [HH21] proposed an analogical reasoning system intended for discovering rules by training a sequence-to-sequence model using a training set of rules represented in first-order logic. Here the role of domain-knowledge comes post training of the deep sequence model; that is, an evaluator (a human expert) tests each discovered rule from the model by unifying them against the (domain) knowledge base. The domain-knowledge here serves as some kind of a validation set where if the ratio of successful rule unification crosses a certain threshold, then the set of discovered rules are accepted.

2.4 Transforming the Model

Over the years, many studies have shown that domain-knowledge can be incorporated into a deep neural network by introducing constraints on the model parameters (weights) or by making a design choice of its structure. Figure 2.6 shows a simple block diagram showing domain-knowledge incorporation at the design stage of the deep neural network.



(a) Transforming structure

(b) Transforming parameters

Figure 2.6: Introducing background knowledge into deep neural network by transforming the model (structure and parameter). In (a), the transformation block $\mathcal{T}$ takes a input structure of a model π and outputs a transformed structure π' based on background knowledge (BK). In (b), the transformation block $\mathcal{T}$ takes a set of parameters $\boldsymbol{\theta}$ of a model and outputs a transformed set of parameters π' based on background knowledge (BK).

2.4.1 Constraints on Parameters

In a Bayesian formulation, there is an explicit mechanism for the inclusion of domain-knowledge through the use of priors. The regularisation terms in loss-functions, for

example, can be seen as an encoding of such prior information, usually on the network's structure. Priors can also be introduced on the parameters (weights) of a network. Explicitly, these would take the form of a prior distribution over the values of the weights in the network. The priors on networks and network weights represent our expectations about networks before receiving any data, and correspond to penalty terms or regularisers. Buntine and Weigend [BW91] extensively study how Bayesian theory can be highly relevant to the problem of training feedforward neural networks. This work is explicitly concerned with choosing an appropriate network structure and size based on prior domain-knowledge and with selecting a prior for the weights.

The work by [Nea95] on Bayesian learning for neural networks also showed how domain-knowledge could help build a prior probability distribution over neural network parameters. In this, the shown priors allow networks to be "self-regularised" to not overfit even when the complexity of the neural network is increased. In a similar spirit, [KT07] showed how prior domain knowledge could be used to define 'meta-features' that can aid in defining the prior distribution of weights. These meta-features are additional information about each of the features in the available data. For instance, for an image recognition task, the meta-feature could be the relative position of a pixel (x, y) in the image. This meta information can be used to construct a prior over the weights for the original features.

There have been some studies that involve representing domain-knowledge as qualitative statements, called monotonicity constraints [ARD05]. These studies have focused primarily on estimating parameters for Bayesian networks [YN13]. Since monotonic constraints are intuitive to the domain-experts in many real-world scenarios, there have been decent attempts in constructing monotonic deep neural networks. The main idea is again the same as discussed before: Constraining the parameters of the deep neural network to be non-negative by imposing a non-negative activation function at the internal neurons [MZB+21, RS22].

Transfer Learning

Transfer Learning is a mechanism to introduce priors on weights when data is scarce for a problem (usually called the "target" domain). Transfer learning relies on data availability for a problem similar to the target domain (usually called the "source" domain). From the position taken in this chapter, domain-knowledge for transfer learning is used to change the structure or the parameter values (or both) for a model for the target problem. The nature of this domain-knowledge can be seen prior distributions on the structure and/or parameter-values (weights) of models for the target problem. The prior distributions for the target model are obtained from the models constructed for the source problem.

In practice, transfer learning from a source domain to a target domain usually in-

volves a transfer of weights from models constructed for the source domain to the network in the target domain. This has been shown to boost performance significantly. From the Bayesian perspective, transfer learning allows the construction of the prior over the weights of a neural network for the target domain based on the posterior constructed in the source domain. Transfer learning is not limited by the kind of task (such as classification, regression, etc.) but rather by the availability of related problems. Language models are some of the very successful examples of the use of transfer learning, where the models are initially learnt on a huge corpus of data and fine-tuned for other more specialised tasks. [ZQD+20] provides an in-depth review of some of the mechanisms and the strategies of transfer learning. Transfer learning need not be restricted to deep neural networks only: in a recent study, [LWM18] proposes a model that transfers knowledge from a deep neural network to a decision tree using knowledge distillation framework. The symbolic knowledge encoded in the decision tree could further be utilised for a variety of tasks. A recently available learning technique that is closely related to transfer learning is Transformational Machine Learning (TML [OOD+21]), albeit with a major difference: While in transfer learning the parameters of a trained model built for one problem serve as priors for a (related) target problem, in TML, the predictions of a (or a set of) model(s) serve as input features for a related target problem. This technique allows indirect inclusion of knowledge from one or multiple problems into a machine learning (including deep neural network) model.

A subcategory of transfer learning is one in which the problem (or task) remains the same, but there is a change in the distribution over the input data from the source and the target. This form of learning is viewed as an instance of domain-adaptation [BDBC+10]. Similar to transfer learning, the knowledge is transferred from a source domain to a target domain in the form of a prior distribution over the model parameters. This form of domain-adaptation uses the same model structure as the source, along with an initial set of parameter values obtained from the source model. The parameter values are then fine-tuned using labelled and unlabelled data from the target data [THDS15]. An example of this kind of learning is in [DSW+20] where a BERT model is fine-tuned with data from multiple domains. There are some recent surveys along these lines: [WD18, RP20].

2.4.2 Specialised Structures

DNN-based methods arguably work best if the domain-knowledge is used to inspire their architecture choices [BGKP21]. There are reports on incorporating first-order logic constructs into the structure of the network. This allows neural networks to operate directly on the logical sentences comprising domain-knowledge.

Domain-knowledge encoded as a set of propositional rules is used to constrain the structure of the neural network. Parameter-learning (updating of the network weights)

then proceeds as normal, using the structure. The result could be thought of as learning weighted forms of the antecedents present in the rules. The most popular and oldest work along this line is Knowledge-Based Artificial Neural Network (KBANN) [TSN90] that incorporates knowledge into neural networks. In KBANN, the domain-knowledge is represented as a set of hierarchically structured propositional rules that directly determines a fixed topological structure of a neural network [TS94]. KBANN was successful in many real-world applications; but, its representational power was bounded by pre-existing set of rules which restricted it to refine these existing rules rather than discovering new rules. A similar study is KBCNN [Fu93], which first identifies and links domain attributes and concepts consistent with initial domain-knowledge. Further, KBCNN introduces additional hidden units into the network, and most importantly, it allowed decoding of the learned rules from the network in symbolic form. However, both KBANN and KBCNN were not appropriate for learning new rules because of the way the initial structure was constructed using the initial domain-knowledge base.

Some of the limitations described above could be overcome with the proposal of a hybrid system by Fletcher and Obradovic [FO93]. The system was able to learn a neural network structure that could construct new rules from an initial set of rules. Here, the domain-knowledge is transformed into an initial network through an extended version of KBANN's symbolic knowledge encoding. It performed incremental hidden unit generation, thereby allowing construction or extension of the initial rule-base. In a similar manner, there was a proposal for using Cascade ARTMAP [Tan97] which could not only construct a neural network structure from rules but also perform explicit cascading of rules and multistep inferencing. It was found that the rules extracted from Cascade ARTMAP are more accurate and much cleaner than the rules extracted from KBANN [TS93].

In the late 1990s, Garcez and Zaverucha proposed a massively parallel computational model called CIL^{2}P based on feedforward neural network that integrates inductive learning from examples and domain-knowledge, expressed as a propositional logic program [GZ99]. A translation algorithm generates a neural network. Unlike KBANN, the approach uses the notion of "bipolar semi-linear" neurons. This allows the proof of a form of correctness, showing the existence of a neural-network structure that can compute the logical consequences of the domain-knowledge. The output of such a network, when combined into subsequent processing naturally incorporates the intended interpretation of the domain predicates. The authors extend this to the use of first-order logic programs: we have already considered this in section 2.2.

A recent proposal called LENSR [XXK+19] focuses on embedding symbolic knowledge expressed as logical rules. It considers two languages of representation: Conjunctive Normal Form (CNF) and decision-Deterministic Decomposable Negation Normal form (d-DNNF), which can naturally be represented as graph structures. The graph structures can be provided to a graph neural network (GNN) to learn an embedding suitable for

further task-specific implementations.

Somewhat in a similar vein to the work by [GZ99], the work reported in [XZF+18] considers as a set of propositional statements representing domain constraints. A deep neural network is then trained to find satisfying assignments for the constraints. Again, once such a network is constructed, it can clearly be used in subsequent processing, capturing the effect of the domain constraints. The network is trained using a semantic loss that we have described in subsection 2.3.2.

In [LS20], it is proposed to augment a language model that uses a deep net architecture with additional statements in first-order logic. Thus, given domain-knowledge encoded as first-order relations, connections are introduced into the network based on the logical constraints enforced by the domain-relations. The approach is related somewhat to the work in [SAZ+18] that does not explicitly consider the incorporation of domain-knowledge but does constrain a deep neural network's structure by first grounding a set of weighted first-order definite clauses and then turning them into propositional programs.

We note that newer areas are emerging that use representations for domain-knowledge that go beyond first-order logic relations. This includes probabilistic first-order logic as a way of including uncertain domain-knowledge [MDK+18]. One interesting way this is being used is to constrain the training of "neural predicates", which represent probabilistic relations that are implemented by neural networks, and the framework can be trained in an end-to-end fashion [MDK+18, WMMR21]. In DeepProbLog [MDK+18], for example, high-level logical reasoning can be combined with the sub-symbolic discriminative power of deep networks. For instance, a logic program for adding two digits and producing the output sum is straightforward. However, what if the inputs are images of the corresponding digits? Here, a deep neural network is used to map an image to a digit, while a (weighted) logic program, written in ProbLog [DRKT07], for the addition operation, is treated as the symbolic domain-knowledge. The ProbLog program is extended with a set of ground neural predicates for which the weights correspond to the probability distribution of classes of digits $(0, \ldots, 9)$. The parameters (weights of predicates and weights of neural network) are learned in an end-to-end fashion. A recent approach called DeepStochLog [WMMR21] is a framework that extends the idea of neural predicates in DeepProbLog to definite clause grammars [PW80]. The reader may note that although DeepProbLog and DeepStochLog do not really transform the structure of the deep neural network, we are still considering these methods under the heading of specialised structures. This is because of the fact that the hybrid architecture is a tightly coupled approach combining probabilistic logic and deep neural networks.

One of the approaches involves transformation of a probabilistic logic program to graph-structured representation. For instance, in kLog [FCDRDG14] the transformed representation is an undirected bipartite graph in the form of 'Probabilistic Entity-Relationship model' [HMK07] which allows the use of a graph-kernel [VSKB10] for data

classification purpose, where each data instance is represented as a logic program constructed from data and background-knowledge. Another approach uses weighted logic programs or *templates* with GNNs [ŠŽK21] demonstrating how simple relational logic programs can capture advanced graph convolution operations in a tightly integrated manner. However, it requires the use of a language of Lifted Relational Neural Networks (LRNNs) [SAZ+18]. Template-based construction of deep neural network structure can be also seen in Logical Neural Networks (LNNs: [RGL+20]). LNNs resemble a tree structure where the leaf nodes represent the facts in the data and background knowledge, the internal nodes implement logical connectives (and, or, implication, etc.) using t-norm operators derived from real-valued logic, and the outputs represent the rules [SdCRG21]. The inputs to LNNs are instantiated facts (Boolean), and the network is trained by minimising a constrained loss function which is a consequence of such a specialised structure.

An interesting proposal is to transform facts and rules, all represented in (weighted) first-order logic into matrix (or tensor) representations. Learning and inference can then be conducted on these matrices (or tensors) [SG16, CYM20] allowing faster computation. NeuralLog [GC21], for example, extends this idea and constructs a multilayered neural network, to some extent, similar to the ones in LRNN consisting of fact layer, rule layer and literal layer etc. The learning here refers to the updates of the weights of the rules. Another work that translates domain-knowledge in first-order logic into a deep neural network architecture consisting of the input layer (grounded atoms), propositional layer, quantifier layer and output layer is [DGS17]. Similar to LRNN, it uses the fuzzy t-norm operators for translating logical OR and AND operations.

Further emerging areas look forward to providing domain-knowledge as higher-order logic templates (or "meta-rules": see [CDM20] for pointers to this area). To the best of our knowledge, there are, as yet, no reports in the literature on how such higher-order statements can be incorporated into deep neural networks.

2.5 Summary of the Review

Our primary discussion in this review is about the techniques for inclusion of some form of domain-knowledge into deep neural networks. We classify these techniques of into 3 broad categories where the domain-knowledge (1) changes the input data representation, (2) changes the loss function used for training the deep neural network, and (3) changes the parameters or structure of the deep neural network. Under each of these categories, we studied several techniques of inclusion of domain-knowledge. We summarise these techniques in Figure 2.7. We discuss some practical challenges associated with these techniques next.

We first outline some practical challenges in incorporating domain-knowledge encoded as logical or numerical constraints into a deep neural network. Below, we outline some

Principal Approach	Work (Reference)	Type of Learner
Transforming Data	DRM [Lod13]	MLP
	CILP++ [FZG14]	MLP
	R-GCN [SKB+18]	GNN
	KGCN [WZX+19]	GNN
	KBRD [CLZ+19]	GNN
	DG-RNN [YZQ+19]	RNN
	DreamCoder [EWN+20]	DNN*
	Gated-K-BERT [YLD+21]	Transformer
	KRISP [MCP+21]	GNN, Transformer
Transforming Loss	IPKFL [KT07]	CNN
	ILBKRME [RSR15]	MLP
	HDNNLR [HML+16]	CNN, RNN
	SBR [DGS17]	MLP
	SBR [DRG17]	CNN
	DL2 [FBDC+19]	CNN
	Semantic Loss [XZF+18]	CNN
	LENSR [XXK+19]	GNN
	DANN [MIM+19]	MLP
	PC-LSTM [LZZ21]	RNN
	DomiKnowS [FGU+21]	DNN*
	MultiplexNet [HKBG21]	MLP, CNN
	Analogy Model [HH21]	RNN
Transforming Model	KBANN [TS94]	MLP
	Cascade-ARTMAP [Tan97]	ARTMAP
	CIL^{2}P [GZ99]	RNN
	DeepProbLog [MDK+18]	CNN
	LRNN [SAZ+18]	MLP
	TensorLog [CYM20]	MLP
	Domain-Aware BERT [DSW+20]	Transformer
	NeuralLog [GC21]	MLP
	DeepStochLog [WMMR21]	DNN*
	LNN [SdCRG21]	MLP

Figure 2.7: Some selected works, in no particular order, showing the principal approach of domain-knowledge inclusion into deep neural networks. DNN* refers to a DNN structure dependent on intended task. We use 'MLP' here to represent any neural network, that conforms to a layered-structure that may or may not be fully-connected. RNN also refers to sequence models constructed using Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells.

immediate practical challenges concerning the logical constraints:

- There is no standard framework for translating logical constraints to neural networks. While there are simplification methods which first, construct a representation of the logical constraint that a standard deep neural network can consume,

this process has its limitations as described in the relevant section above.

- Logic is not differentiable. This does not allow using standard training of deep neural network using gradient-based methods in an end-to-end fashion. Propagating gradients via logic has now been looked at in [EG18], but the solution is intractable and does not allow day-to-day use.

- Many neural network structures are directed acyclic graphs (DAGs). However, transforming logical formulae directly into neural network structures in the manner described in some of the discussed works can introduce cyclic dependencies, which may need a separate form of translations.

- A deep neural network, with its structure constrained via logical domain-knowledge may not always be scalable to large datasets.

There are also the following practical challenges concerning the numerical constraints:

- We have seen that the numerical constraints are often provided with the help of modification to a loss function. Given some domain-knowledge in a logical representation, constructing a term in loss function is not straightforward.

- Incorporating domain-knowledge via domain-based loss may not be suitable for some safety-critical applications.

- The process of introducing a loss term often results in a difficult optimisation problem (sometimes constrained) to be solved. This may require additional mathematical tools for a solution that can be implemented practically.

Furthermore, in this review, we have not discussed how the domain-knowledge is to be acquired from domain experts.

Chapter 3

Inclusion of Domain-Knowledge using Propositionalisation*

The simplest kind of neural network architecture to realise the power of deep learning is a multilayer perceptron (MLP), also called a feed-forward fully-connected neural network. An MLP consists of multiple hidden layers where each layer (linearly or non-linearly) transforms the feature it receives at its input. Mathematically speaking, an MLP is a composition of multiple (linear or non-linear) functions, trying to learn a hierarchy of intermediate feature representations that most effectively aid the global learning task. However, despite some spectacular successes, deep learning is thought unlikely to be sufficient for many kinds of data analysis problems. Setting aside any infrastructural difficulties, we still need to be able to address representational issues arising in problems that require us to capture complex relationships amongst objects in the domain; and also deal with the lack of large amounts of data.

Some of this difficulty may be alleviated if knowledge already available in the area of interest can be taken into account, both to identify relationships in the data and to play the role of priors (used here in the probabilistic sense, where prior distributions are one way of dealing with small amounts of data). Consider, for example, a problem in the area of drug design. Much may be known already about the target of interest, small molecules that have proved to be effective, what can and cannot be synthesized cheaply and so on. If these concepts are relevant to constructing a model for predicting good drugs, it seems both unnecessary and inefficient to require a deep neural network to re-discover them (the problem is actually worse: it may not even be possible to dis-

*The content of this chapter is based on the following:
T. Dash, A. Srinivasan, R.S. Joshi, A. Baskar, "Discrete stochastic search and its application to feature-selection for deep relational machines", *International Conference on Artificial Neural Networks*, 2019; https://doi.org/10.1007/978-3-030-30484-3_3. and
T. Dash, A. Srinivasan, L. Vig, O.I. Orhobor, R.D. King, "Large-scale assessment of deep relational machines", *International Conference on Inductive Logic Programming*, 2018; https://doi.org/10.1007/978-3-319-99960-9_2. (*Winner of the Best Student Paper Award)

cover the concepts from first-principles, using the data available). In Chapter 2 we have described several ways in which logical encodings of complex background knowledge can be provided to a deep neural network. In this chapter, we explore the simplest of these approaches (propositionalisation [LDG91]) for use in the simplest form of deep neural network (a multilayer perceptron, or MLP). In subsequent chapters, we will investigate more sophisticated ways in which logical encodings of domain-knowledge can be included into a more sophisticated form of deep neural network.

In this chapter, we describe Deep Relational Machines (or DRMs), which offer a simple way of incorporating complex domain-knowledge into MLPs. In a DRM, domain-knowledge is introduced through "relational features": in the original formulation of [Lod13] the features are selected by an Inductive Logic Programming (ILP) engine using domain-knowledge encoded as logic programs. Each input feature to a DRM is a first-order Boolean function. For example, "function f_1 is true if the instance x is a molecule containing a 7-membered ring connected to a lactone ring"—definitions of relations like 7-membered and lactone rings are expected to be present in the background knowledge. In the original DRM formulation [Lod13], these functions are learned by an ILP engine. This follows a long line of research sometimes called *propositionalisation* in which features constructed by ILP have been used by other modelling methods like regression, decision trees, SVMs, topic models, and multiplicative-weight linear threshold models [FSK12, JRS08, RJBS07, SSR12, SSRN06, SSJ+09, SK99], inspired by the original work in [LDG91]. In some recent works such as [Lod13, FZG15], the final model is constructed in two steps: first, a set of features are obtained, and then, the final model is constructed using these features, possibly in conjunction with other features already available. Usually the models show significant improvements in predictive performance when an existing feature set is enriched in this manner. In [Lod13], the deep neural network with features obtained using an ILP engine is shown to perform well, although the empirical evidence is limited. Reports in the literature on the use of DRMs have been deficient on the following counts: (1) They have been tested on very small amounts of data (the maximum appears to be 7 datasets–not all independent–with few 1000s of instances); (2) The background knowledge involved has been modest, involving few 10s of predicates; and (3) No guided strategy for selecting from the large space of possible relational features appears to have been developed (simple random sampling appears to be the most that has been employed: see for example [VSBV17]). In this chapter, we address each of these shortcomings as follows: (1) We conduct experiments on over 70 real-world datasets obtained from biochemical domain involving 100s of 1000s of relational data instances (These datasets will form a running thread for experimental results in the dissertation); (2) We use industrial-strength background knowledge involving multiple hierarchies of complex definitions of chemical structures to test the performance of DRMs at large-scale. Overall, the domain-knowledge consists of definitions for about 100

predicates; and (3) We propose a utility-based strategy called "hide-and-seek" sampling, for drawing relational features from a large but countable space of possible features.

3.1 Some Logic Programming Concepts

We first outline some standard logic programming concepts required for understanding some upcoming portions of this chapter. For additional background and further terminology see [Nil91, CL14, Llo12, Md94].

A language of first order logic programs has a vocabulary of constants, variables, function symbols, predicate symbols, logical implication '$\leftarrow$', and punctuation symbols. A function or predicate can have a number of arguments known as *terms*. Terms are defined recursively. A constant symbol (or simply "constant") is a term. A variable symbol (or simply "variable") is a term. If f is an m-ary function symbol, and $t_1, \ldots, t_m$ are terms, then the function $f(t_1, \ldots, t_m)$ is a term. A term is said to be *ground* if it contains no variables.

We will use the convention used in logic programming when writing clauses. Thus, predicate, function and constant symbols are written as a lower-case letter followed by a string of lower- or upper-case letters, digits or underscores ('_'). Variables are written similarly, except that the first letter must be upper-case. Usually, predicate symbols will be denoted by symbols like p, q, r *etc.* and symbols like X, Y, Z to denote variables. If p is an n-ary predicate symbol, and $t_1, \ldots, t_n$ are terms, then the predicate $p(t_1, \ldots, t_n)$ is an atom. Predicates with the same predicate symbol but different arities are distinguished by the notation p/n where p is a predicate of arity n.

A literal is either an atom or the negation of an atom. If a literal is an atom it is referred to as a positive literal, otherwise it is a negative literal. A clause is represented as an implication (or "rule"): $l_1, \ldots, l_i \leftarrow l_{i+1}, \ldots, l_k$, where each l_j is an atom. Alternatively, such a clause can also be represented as a disjunction of literals: $\{l_1 \vee \cdots \vee l_i \vee \neg l_{i+1} \vee \cdots \vee \neg l_k\}$ or as a set of literals: $\{l_1, \ldots, l_i, \neg l_{i+1}, \ldots, \neg l_k\}$. A definite clause $l_1 \leftarrow l_2, \ldots, l_k$ has exactly one positive literal, called the *head* of the clause, with the literals $l_2, \ldots, l_k$ known as the *body* of the clause. A definite clause with a single literal is called a *unit* clause, and a clause with at most one positive literal is called a Horn clause. A set of Horn clauses is referred to as a logic program.

Example 3.1. *Some examples of clauses:*

(1) $p(X) \leftarrow$

(2) $p(X) \leftarrow q(X, 2)$

(3) $p(X) \leftarrow q(X, Y), r(Y)$

A *substitution* θ is a finite set $\{V_1/t_1, \ldots, V_n/t_n\}$ mapping a set of n distinct variables V_i, $1 \leq i \leq n$, to terms t_j, $1 \leq j \leq n$ such that no term is identical to any of the variables. A substitution containing only ground terms is a *ground* substitution. For substitution θ and clause C the expression $C\theta$ denotes the clause where every occurrence of a variable in C is replaced by the corresponding term from θ. If θ is a ground substitution then $C\theta$ is called a ground clause.

3.2 Relational Data and Relational Features

This dissertation is about using machine learning to construct models from what we will be calling "relational data". It is common in machine learning to learn from data represented by feature-vectors, in which the j^{th} feature for the i^{th} data instance x_i is of the form $f_j(x_i) = v_{ij}$. But this is just a special case of representing data using a single relation (the equality relation, $=$). More generally, we can imagine that properties of x_i could be described by using many other relations. Consider, for example, allowing the use of the inequality relation $\leq$ to be included in the description of x_i. Then, perhaps we could additionally say $(f_j(x_i) = v_{ij}) \wedge (v_{ij} \leq 2.5)$. In this dissertation, we will use the term *relational data* to refer to data that can be described at least in the form of multiple tables in a relational database. Without getting into the details of exactly what relations are present, we will simply denote the set of relational data by $\mathcal{X}$. In addition, each relational data instance in $\mathcal{X}$ is associated with a class label, the class to which the instance belongs. We denote a finite set of class labels by $\mathcal{Y}$. A set of labelled relational data instances, denoted by $\mathcal{E}$, can be defined as a binary relation *class* which is a subset of the Cartesian product $\mathcal{X} \times \mathcal{Y}$. Any element in $\mathcal{E}$ is denoted by $class(x, c)$ where $x \in \mathcal{X}$ and $c \in \mathcal{Y}$.

In this chapter, we intend to learn deep neural networks using relational features describing the relational data instances. Informally, a "relational feature" is intended to capture the relations that exist in relational data instances. We represent a relational feature as a single definite clause:

$$C_i : \forall X \ (p(X) \leftarrow Cp_i(X))$$

or, simply written without the quantifier as

$$C_i : p(X) \leftarrow Cp_i(X)$$

where X is a variable takes values of relational data instances in $\mathcal{X}$, $Cp_i(X)$ is a conjunction of predicates. Each predicate in $Cp_i(X)$ is defined as part of some background knowledge B and it can have variables which are existentially quantified. For us, re-

lational features will therefore be a definite-clause in a restricted form of Datalog (we will not allow recursion). The distinguished literal $p(\cdot)$ is called the "head" or the "head literal" of C_i, which we assume is unique. The conjunction of predicates $Cp_i(\cdot)$ is called the "body" of clause C_i. In addition, as we mentioned, C_i is not self-recursive, that is, the body of C_i does not contain literal of the form $p(\cdot)$.

Example 3.2 (The trains problem)**.** *Let's take an example of a problem introduced by Michalski on discriminating between eastbound and westbound trains [Mic80]. The problem has the following setting:*

- *Consider there are two sets of trains: trains going east (Eastbound) and trains going west (Westbound).*

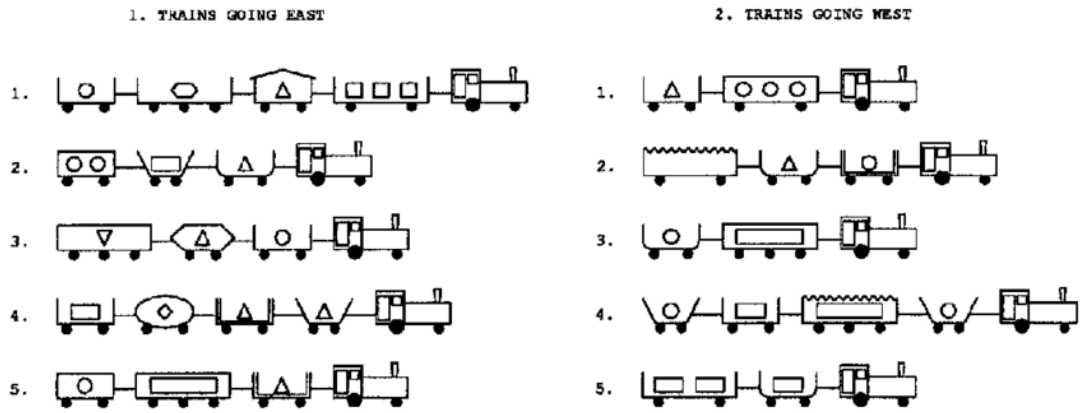


Figure 3.1: Michalski's "trains" problem; adapted from [Mic80, MMPS94].

- *Each train comprises of a set of locomotive pulling wagons; whether a particular train is travelling towards the east or towards the west is determined by some properties of that train.*

- *The learning task here is to determine what governs which kinds of trains are eastbound and which kinds are westbound.*

- *The following background knowledge about each wagon (or car) in the train are available: which train it is part of, its shape, how many wheels it has, whether it is open (i.e. has no roof) or closed, whether it is long or short, the shape of the things the car is loaded with. In addition, for each pair of connected wagons, knowledge of which one is in front of the other can be extracted. In a relational database, these information are represented as tables. Information about any relational data instance can then be obtained by natural joins among these tables.*

- *Let the background knowledge consists of the following tables:* $has\_car(Train, Car)$, $short(Car)$, $closed(Car)$, $shape(Car, Geometry)$, $wheels(Car, Count)$.

- *Given some relational data instances representing trains, here are some relational features:*

$$
\begin{aligned}
&C_1 : (p(X) \leftarrow has\_car(X, Y)) \\
&C_2 : (p(X) \leftarrow has\_car(X, Y), short(Y)) \\
&C_3 : (p(X) \leftarrow has\_car(X, Y), closed(Y)) \\
&C_4 : (p(X) \leftarrow has\_car(X, Y), short(Y), closed(Y)) \\
&C_5 : (p(X) \leftarrow has\_car(X, Y_1), short(Y_1), has\_car(X, Y_2), closed(Y_2))
\end{aligned}
$$

  Here the variable X in the head literals of the above features is universally quantified, denoting any train, and the variables Y, Y_1 and Y_2 in the body literals are existentially quantified, denoting cars. It is evident that the relational features are able to describe a relational data instance with the help of the predicates in the background knowledge (here: $has\_car/2$, $short/1$ and $closed/1$).

3.3 Propositionalisation

Many deep neural networks, such as MLPs require a data instance to be represented as a single numeric feature vector or a tensor. But, the clausal representation of relational features as described in the previous section does not tell us how to obtain a valuation (in Boolean or in real) of the feature itself for any data instance $X = x$. Therefore, we need a mechanism to convert these relational features into numeric feature vectors. A popular technique within the area of relational learning is *propositionalisation* that provides a way to encode relational features as Boolean-valued feature vectors [Lav90, LDG91]. We will call this kind of encoding of relational features as "proposionalised encoding" and define it as follows.

Definition 3.1 (Propositionalised Encoding)**.** *Given a set of relational data instances $\mathcal{X}$, background knowledge B, a set of relational features $\mathcal{C} = \{C_1, \ldots, C_d\}$, where each $C_i \in \mathcal{C}$ is of the form $(p(X) \leftarrow Cp_i(X))$, the propositionalised encoding of a relational feature for a data instance $x \in \mathcal{X}$ is a mapping $f : \mathcal{C} \times \mathcal{X} \rightarrow \{0, 1\}$ defined as:*

$$
f(C_i, x) = \begin{cases} 1 & \text{if } B \cup C_i\{X/x\} \models p(x) \\ 0 & \text{otherwise.} \end{cases}
$$

Thus, the propositionalised representation of the data instance x with d-relational features in $\mathcal{C}$ is a vector: $(f(C_1, x), f(C_2, x), \ldots, f(C_d, x))$.

Example 3.3 (Propositionalisation of Trains)**.** *Let $\mathcal{C}$ consists of the relational features described in Example 3.2. The propositionalised representation of the trains dataset described above would look like the following, where x denotes an instance representing a train.*

Example	$f(C_1, x)$	$f(C_2, x)$	$f(C_3, x)$	$f(C_4, x)$	$f(C_5, x)$	*class*
x_1	1	1	1	1	0	*eastbound*
x_2	1	1	1	1	1	*eastbound*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_N	1	1	1	0	0	*westbound*

3.4 A Discrete Space of Relational Features

For use in a DRM, we need to identify a set of relational features and their values for the data. Here we arrive at an immediate difficulty: the set of relational features can be extremely large (in many cases, even infinitely large). How are we to select a suitable subset of these for a DRM? Two aspects will be helpful to us. First, the set does have some structure: the relation of θ-subsumption (see Plotkin [Plo72]) imposes a partial-ordering on the set. Secondly, the field of Inductive Logic Programming (ILP) has developed techniques for bounding the set. We describe some of these concepts in brief as follows.

Definition 3.2 (Plotkin's θ-Subsumption)**.** *A clause C_1 θ-subsumes a clause C_2 if and only if there exists a substitution θ such that $C_1\theta \subseteq C2$. We write $C_1 \preceq_\theta C_2$ to denote C_1 θ-subsumes C_2. Further, whenever $C_1 \preceq_\theta C_2$ we will call C_1 is more general than C_2 and C_2 is more specific than C_1. For a set of clauses S and the subsumption ordering $\preceq$, we have that for every pair of clauses $C_1, C_2 \in S$, there is a least upper bound and greatest lower bound, called, respectively, the least general generalisation (lgg) and most general unifier (mgu) of C_1 and C_2, which are unique up to variable renaming. The subsumption partial ordering on clauses enables the definition of a lattice, called the* subsumption lattice*.*

Example 3.4 (θ-subsumption)**.** *For the following two clauses C_1 and C_2, C_1 θ-subsumes C_2, where $\theta = \{X/a, Y/b\}$ is a ground substitution. C_1 is more general than C_2 and C_2 is more specific than C_1.*

$$C_1 : p(X, Y) \leftarrow q(X, Y), r(X) \qquad C_2 : p(a, b) \leftarrow q(a, b), r(a).$$

Similarly, for C_3 and C_4 below, C_3 θ-subsumes C_4, where $\theta = \{X/Z, Y/Z\}$. C_3 is more general than C_4 and C_4 is more specific than C_3.

$$C_3 : p(X, Y) \leftarrow q(X, Y), q(Y, X) \qquad C_4 : p(Z, Z) \leftarrow q(Z, Z).$$

A fragment of the subsumption lattice over the set of relational features for the trains problem is shown in Figure 3.2.

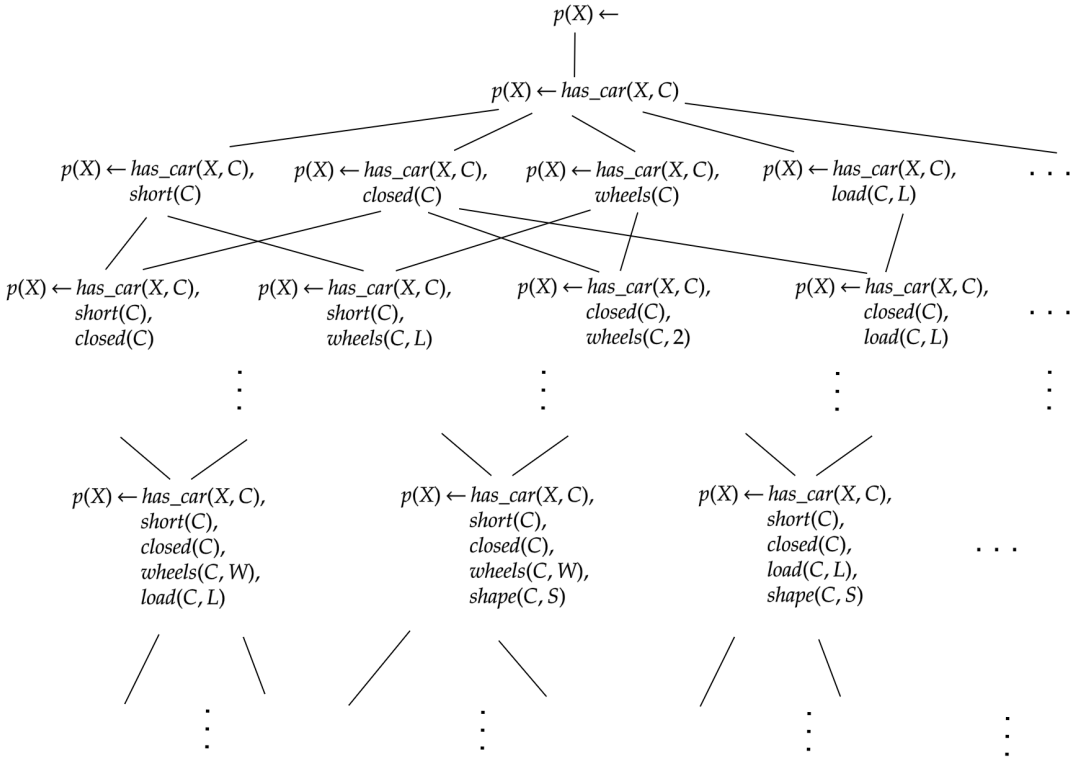


Figure 3.2: A fragment of the subsumption lattice of relational features for the trains problem.

3.4.1 Bounding the Lattice of Relational Features

We will resort to techniques developed in Inductive Logic Programming (ILP) to restrict the subsumption lattice in various ways. The techniques are mainly under the category of Mode-Directed Inverse Entailment, or MDIE, described in [Mug95]. We extensively use MDIE in a later chapter (Chapter 5), and postpone a formal description until then. For the present, we only present the relevant aspects in an informal manner.

MDIE allows us to bound the subsumption lattice by a *top* element. For us, this is the relational feature with an empty-body. Given a relational data instance e, background knowledge B, and a set of "mode declarations" M (again, we postpone a description of these to Chapter 5). MDIE identifies a *bottom* element for the subsumption lattice, called the *most-specific clause*, denoted by $\bot_{B,\mathrm{M}}(e)$. For us, this is the relational feature that contains all the information in B that is related to e. In practice, $\bot_{B,\mathrm{M}}(e)$ could be very large, sometimes infinitely-long. To address this, a further depth-bound d, and the resulting clause is the depth-limited most-specific clause in the mode-language, denoted by $\bot_{B,\mathrm{M},d}(e)$, which is finite.

Example 3.5 (Bounded Subsumption Lattice in ILP). *A bounded subsumption lattice of relational features for our Trains example is shown in Figure 3.3.*

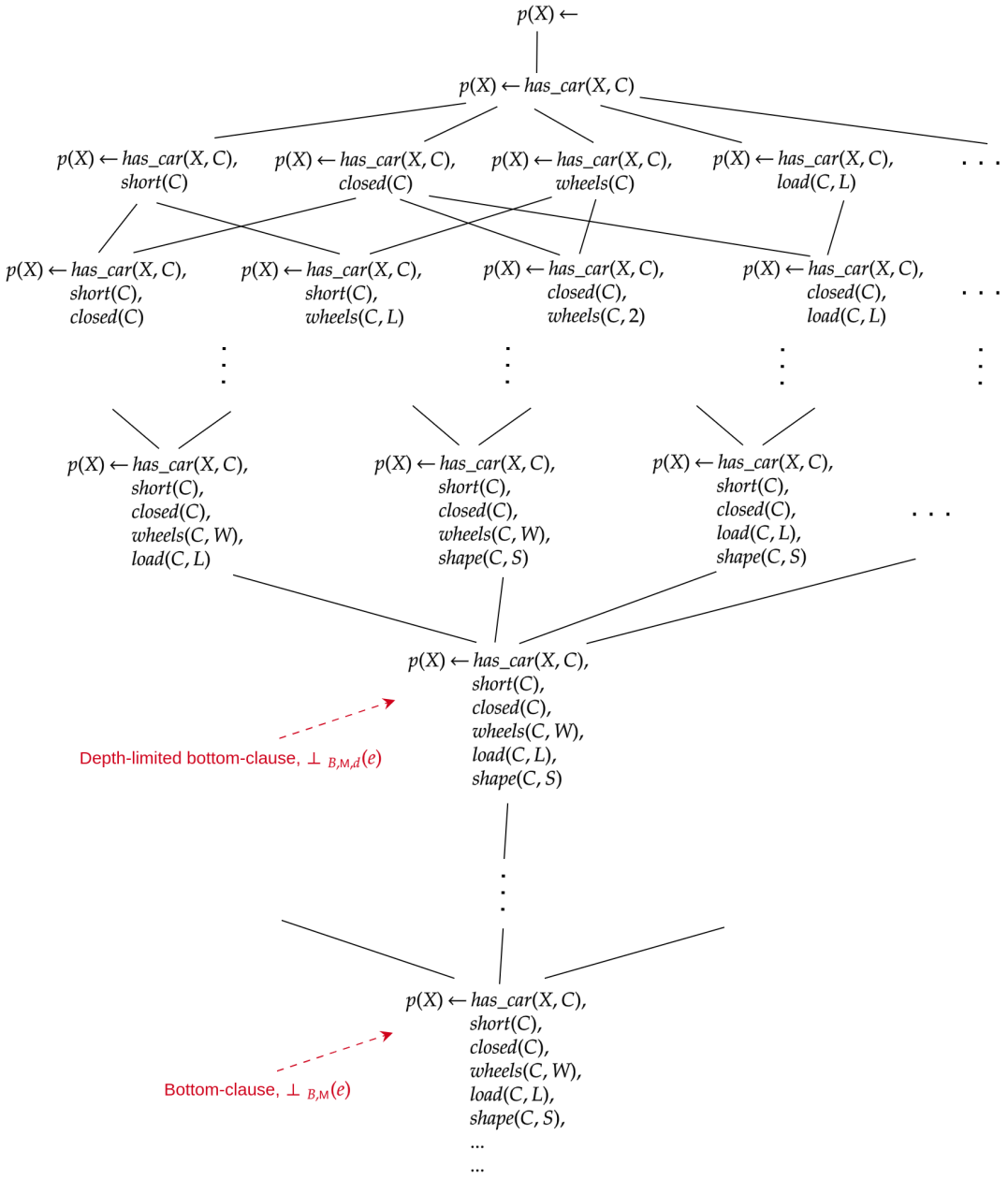


Figure 3.3: The subsumption lattice of relational features for the trains problem. The space is bounded by $p(X) \leftarrow TRUE$ at the top and by the bottom-clause ($\perp_{B,\mathrm{M}}(e)$) at the bottom. The size of the space is bounded by $\mathcal{O}(2^{|\perp_{B,\mathrm{M}}(e)|})$. The relational features are sampled from this space.

We will sample relational features from the bounded lattice constructed by MDIE. We investigate two kinds of sampling strategies: (1) simple random sampling; and (2) a utility-based sampling. The latter is a new sampling strategy that we propose in this dissertation and it is inspired from the idea of the popular "hide-and-seek" games.

For simple random sampling of relational features using MDIE, we use the procedure

from [SSR12]. For completeness, the procedure is reproduced in Procedure 1, using the terminology adopted in this dissertation. The steps are as follows: Given e, B, M, d, let the depth-limited bottom-clause be denoted by $\bot_{B,\mathrm{M},d}(e)$. The procedure needs a parameter called language constraints $\mathcal{L}$ that imposes additional restrictions on the relational features to be constructed by the procedure. One such restriction is the number of literals in the body of the relational feature. The procedure then constructs a clause C_i that subsumes $\bot_{B,\mathrm{M},d}(e)$ by randomly sampling a subset of literals from the body of $\bot_{B,\mathrm{M},d}(e)$. If C_i is not a redundant clause (that is: not drawn earlier) then a relational feature is constructed from it, as described formally in Procedure 1. This process is repeated until a preset maximum number of features ($MaxDraws$) is obtained. The construction of $\bot_{B,\mathrm{M}}(e)$ in Step 7 is as described in [Mug95], and subsumption refers to Plotkin's θ-subsumption as described earlier. The redundancy test used in Step 9 is subsumption-equivalence (which is weaker than logical equivalence).

Procedure 1 Simple random sampling of relational features from a bounded lattice.

1: **procedure** DRAWFEATURES($\mathcal{X}$, B, M, d, $\mathcal{L}$, $MaxDraws$)
2: Let $draws = 0$
3: Let $Drawn$ be $\langle\rangle$
4: Let $i = 1$
5: **while** $draws \leq MaxDraws$ **do**
6: Randomly draw with replacement an example, $e_i \in \mathcal{X}$
7: Let $\bot_{B,\mathrm{M},d}(e_i)$ subsumes $\bot_{B,\mathrm{M}}(e_i)$ ▷ Depth-limited bottom-clause.
8: Randomly draw a clause C_i s.t. $C_i \subseteq \bot_{B,\mathrm{M},d}(e_i)$
9: **if** C_i is not redundant given $Drawn$ **then**
10: Let $C_i = (p(X) \leftarrow Cp_i(X))$
11: Update sequence $Drawn$ with C_i
12: increment i
13: increment $draws$
14: **return** $Drawn$

3.5 Utility-based Sampling of Relational Features

We motivate our approach using a search-based view of feature selection. We can conceptually view this task as searching through subsets of all possible features that an ILP engine can construct. If the number of features that an ILP engine can construct, given data and background knowledge, denoted by $\mathcal{F}$, is small (below 10), then for each subset of features, we can construct and evaluate a DRM model and record its performance. Then the task is then to identify the feature subset that results in the best performance. However, in all practical situations, the set $\mathcal{F}$ will be very large (100s of 1000s or more). As a search problem, it is neither feasible to construct all the features nor practically possible to construct and evaluate all the models that can be constructed using these fea-

tures, making the search problem intractable. The figure shown in Figure 3.4 describes the problem pictorially. Even the search for a single good relational feature, in the large discrete space of features, can prove to be difficult. Figure 3.5 shows the feature space for the trains problem where each feature is associated with some utility score. The search problem described here, either for a feature subset or a single relational feature, can be mapped to the standard problem of "hide-and-seek" game, in which a hider (here the best subset or the best relational feature) hides in one of several locations (here, $2^{|\mathcal{F}|}$ or $O(2^{|\perp_{B,\mathrm{M}}(e)|})$) selected using some probability distribution (the 'hider distribution', denoted by H). The task of the seeker (here, the search procedure) is to find the hider by opening as few boxes as possible, guided by its own distribution (the 'seeker distribution', denoted by S). We formalise this optimisation problem by examining the relationship between the hider and seeker distributions. Intuitively, it would seem that an optimal search will result if the two distributions are the same. Our formalisation, however, shows that, surprisingly, this is not the case.

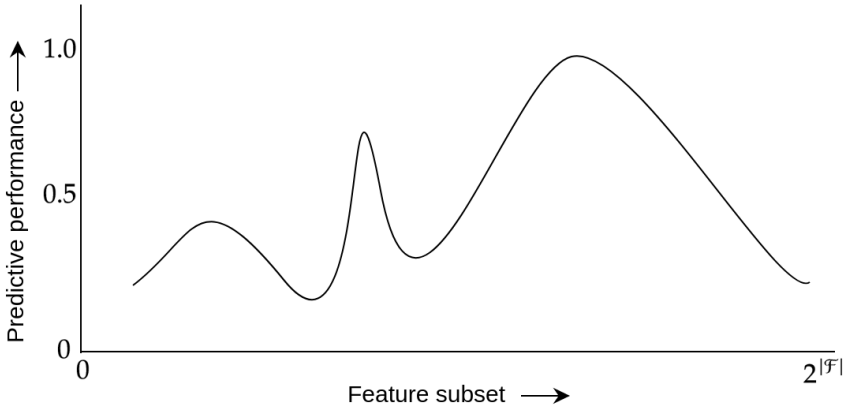


Figure 3.4: Redrawn and adapted from [JRS08]. Identifying the best subset of relational features for constructing a DRM. The X-axis enumerates the different subsets of relational features that can be constructed by an ILP engine ($\mathcal{F}$ denotes the set of all possible relational features that can be constructed by the engine). The Y-axis shows the probability that a data instance drawn randomly using some pre-specified distribution will be correctly classified by the constructed DRM, given the corresponding feature-subset in X-axis. We wish to identify the subset that yields the highest probability, without actually constructing all the features in $\mathcal{F}$.

Our interest is in machine learning algorithms searching potentially infinite discrete spaces [Blu92]. These algorithms can be applied for solving problems concerning natural phenomena that will involve sampling from known or unknown distribution. One example of this kind of problem is prediction of carcinogenicity of chemicals [KMSS96] which forms the application area in this thesis. In such cases, it is conceptually useful to think of targets being distributed according to some non-uniform distribution H. This machine learning setting of searching for targets in natural phenomena is different to the adversarial setting of a hide-and-seek game in which the purpose of the hider is to make

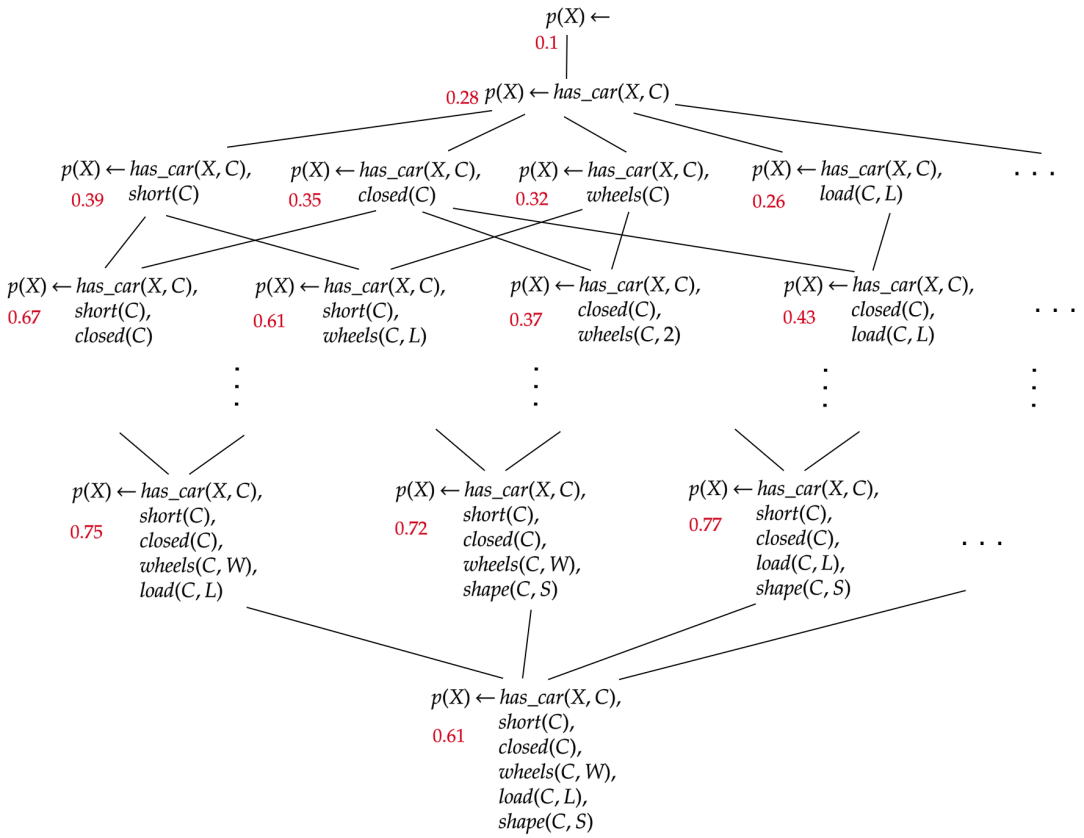


Figure 3.5: The subsumption lattice of relational features for the trains problem. Each feature is associated with a utility score (shown in red colour). Our proposed utility-based sampling strategy selects features from this space.

search as difficult as possible for the seeker. It will be seen below that this corresponds to the special case of H being a uniform distribution, which results in the maximal number of misses by the optimal S. Nevertheless, in this chapter, we will still refer to H as a "hider distribution", and to S as a "seeker distribution", with the caveat that this will not necessarily imply an adversarial setting. Furthermore, practical machine learning often deals with good, rather than optimal solutions. This means that there may be more than one possible target, and finding any one of them should suffice. We can characterise what this means in terms of H, and how will this affect the choice of S. In what follows, (1) We develop a relation between H and S. Further, we address theoretically and experimentally the cases arising from non-uniform H's and from many good solutions; (2) The results are used to develop a sampling procedure for large discrete search spaces; (3) The sampling procedure is then used to select features for Deep Relational Machines (DRMs) from a large space of relational features.

3.5.1 A Distributional Model of Discrete Search

We start with a distributional model that is consistent with the description of the discrete search hide-and-seek game in [Ruc91] and [Sto76]. We start with n boxes and one ball.

The ball is thrown onto the boxes (and it must fall into one of them) with some probability. For us, this gives rise to the "hider distribution" H on the boxes. A stochastic search procedure is a sampling strategy that draws boxes at random using some "seeker distribution" S, until it succeeds eventually after m trials to find the ball. A miss can be understood as an event of selecting (opening) a box and not finding the ball. Here we will assume that if the ball is in a box opened, then it will be detected. There is a cost associated with this search, monotonic in the number of misses m. It is natural to expect that the expected cost to find the ball will depend on whether or not H is known. We sharpen this intuition using the ideal setting when H is known completely to the stochastic search procedure.

Hider distribution known

We will assume that if the ball is in a box, then it will be found. As with a generalised form of the hide-and-seek game, this can be changed to allow finding the ball with some probability even if it is there. Let Z be a random variable for number for misses by the search before finding the ball. We want to find the expected number of misses $\mathrm{E}[Z]|_{H,S}$.

Lemma 3.1 (Expected number of misses with a single ball)**.** *Let $H = (h_1, h_2, \ldots, h_n)$ and $S = (s_1, s_2, \ldots, s_n)$ be discrete probability distributions. Assume that $h_i, s_i > 0$ for $i = 1$ to n. Let the ball be in one of the n boxes according to the hider distribution H. The search attempts to find the ball using the seeker distribution S. Then, the expected number of misses is*

$$\mathrm{E}[Z]|_{H,S} \;=\; \mathrm{E}[Z] = \sum_{k=1}^{n} \frac{h_k}{s_k} - 1. \tag{3.1}$$

Proof: The ideal case is $\mathrm{E}[Z] = 0$. That is, on average, the search opens the correct box k on its first attempt. Now $\mathrm{P}(Z = 0|$ the ball is in box $k) = h_k s_k = h_k(1 - s_k)^0 s_k$. Since the ball can be in any of the n boxes, $\mathrm{P}(Z = 0) = \sum_{k=1}^{n} h_k(1 - s_k)^0 s_k$. More generally, for $Z = j$, the search opens wrong boxes j times, and $\mathrm{P}(Z = j) = \sum_{k=1}^{n} h_k(1 - s_k)^j s_k$. The expected number of misses can now be computed:

$$\begin{aligned}
\mathrm{E}[Z]|_{H,S} &= \sum_{j=0}^{\infty} j\mathrm{P}(Z = j) \\
&= \sum_{j=0}^{\infty} j \sum_{k=1}^{n} h_k(1 - s_k)^j s_k.
\end{aligned}$$

Swapping the summations over j and k, and using the geometric series $\sum_{j=1}^{\infty} a^j = \frac{1}{1-a}$, whenever $|a| < 1$, we get

$$\begin{aligned}
\mathrm{E}[Z]|_{H,S} &= \sum_{k=1}^{n} h_k s_k \sum_{j=0}^{\infty} j(1-s_k)^j \\
&= \sum_{k=1}^{n} \left(h_k s_k \frac{1-s_k}{s_k^2} \right) \\
&= \sum_{k=1}^{n} \left(\frac{h_k s_k}{s_k^2} - \frac{h_k s_k^2}{s_k^2} \right) = \sum_{k=1}^{n} \frac{h_k}{s_k} - \sum_{k=1}^{n} h_k.
\end{aligned}$$

Since $\sum_k h_k = 1$, this simplifies to

$$\mathrm{E}[Z]|_{H,S} = \sum_{k=1}^{n} \frac{h_k}{s_k} - 1.$$

■

We note that $\mathrm{E}[Z]|_{H,S}$ is $n-1$ when $H = S$ or when S is a uniform distribution. Also, we state here that $\mathrm{E}[Z]|_{H,S} \geq (n-1)$ when H is uniform (this is shown later).

A generalisation of the expression of the expected misses above can be derived when there is more than one ball and each box can contain not more than one ball. The expected number of misses is lower if it is sufficient for the search to find any one of the balls.

Lemma 3.2 (Expected number of misses with K balls)**.** *Let H and S be discrete distributions as in Lemma 3.1. Let K balls be in K of the n boxes according to the hider distribution H. The search attempts to find at least one ball using the seeker distribution S. Then, the expected number of misses is*

$$\mathrm{E}[Z]|_{H,S} = \sum_{\substack{i \\ \sigma(i) \in \mathsf{P}(n,K)}} \left(\prod_{k=1}^{K} h_{\sigma(i)}^{(k)} \right) \left\{ \frac{1}{\left(\sum_{k=1}^{K} s_{\sigma(i)}^{(k)} \right)} - 1 \right\}, \tag{3.2}$$

where, $\sigma(i)$ is the i^{th} position in the permutation of $1, \ldots, n$.

Proof: This extends the Lemma 3.1 to a general case of multiple (K) stationary hiders. The number of ways the K hiders can choose to hide in n boxes is ${}^{n}P_K$. Let $\mathsf{P}(n, K)$ denote a set of all such permutations. For example, if there are 3 boxes (locations) and 2 hiders, they can hide in these boxes in 6 possible ways. That is, the first hider hides in box 1 and the second hider hides in box 2, denoted by $(1, 2)$, and so on, as follows:

$$\mathsf{P}(3, 2) = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}.$$

All the K hiders can hide in any one of the choices in $\mathsf{P}(n, K)$ with probability

$$\left(h_{\sigma(i)}^{(1)} h_{\sigma(i)}^{(2)} \dots h_{\sigma(i)}^{(K)}\right),$$

where $h_{\sigma(i)}^{(k)}$ denotes the probability of the hider in k^{th} place in the selected choice of $\sigma(i)$. Analogously, the seeker can find any one of these hiders with probability

$$\left(s_{\sigma(i)}^{(1)} + s_{\sigma(i)}^{(2)} + \dots + s_{\sigma(i)}^{(K)}\right),$$

and would not find the hider once is

$$1 - \left(s_{\sigma(i)}^{(1)} + s_{\sigma(i)}^{(2)} + \dots + s_{\sigma(i)}^{(K)}\right).$$

If the seeker makes the search j times, the probability that it will not find a hider is

$$\left(1 - \left(s_{\sigma(i)}^{(1)} + s_{\sigma(i)}^{(2)} + \dots s_{\sigma(i)}^{(K)}\right)\right)^j.$$

Now, the expected misses for this multiple hider formulation is given as

$$\mathrm{E}[Z] = \sum_{\substack{i \\ \sigma(i) \in \mathsf{P}(n,K)}} \prod_{k=1}^{K} h_{\sigma(i)}^{(k)} \sum_{j=0}^{\infty} j \left(1 - \sum_{k=1}^{K} s_{\sigma(i)}^{(k)}\right)^j \sum_{k=1}^{K} s_{\sigma(i)}^{(k)}.$$

This further simplifies to

$$\mathrm{E}[Z]|_{H,S} = \sum_{\substack{i \\ \sigma(i) \in \mathsf{P}(n,K)}} \prod_{k=1}^{K} h_{\sigma(i)}^{(k)} \left(\frac{1}{\sum_{k=1}^{K} s_{\sigma(i)}^{(k)}} - 1\right).$$

■

All the results above are derived assuming sampling with replacement. The first reason for this assumption is simply mathematical convenience. Sampling without replacement, when all boxes have equal probability is governed by the hypergeometric distribution, yielding $(n-1)/2$ misses on average (see below).

Lemma 3.3 (Expected misses for uniform S)**.** *Let H and S be discrete distribution as in Lemma 3.1 and S be uniform. Let the ball be in one of n boxes according to the hider distribution H. The search attempts to find the ball using the seeker distribution S without replacement. Then, the expected number of misses is*

$$\mathrm{E}[Z]|_{H,S} = \frac{n-1}{2}. \tag{3.3}$$

Proof: From Lemma 3.1, we have:

$$\mathrm{E}[Z]|_{H,S} = \sum_{j=0}^{\infty} j\mathrm{P}(Z=j) = \sum_{j=0}^{\infty} j \sum_{k=1}^{n} h_k (1-s_k)^j s_k.$$

Since the search is without replacement, it will incur a maximum of $n-1$ misses before the hider is found. Therefore, the summation over j will run till $n-1$ (unlike till ∞, in the search with replacement case). Further, a search without replacement would mean that once a box is opened and the hider was not found in the box, that box will be removed from the search space. This leads to a distribution of the probability mass to the rest of the boxes. This implies:

$$\begin{aligned}\mathrm{E}[Z]|_{H,S} &= \sum_{j=0}^{n-1} j \sum_{k=1}^{n} h_k \underbrace{\left(1-\frac{1}{n}\right)\ldots\left(1-\frac{1}{n-j+1}\right)}_{j \text{ terms}}\left(\frac{1}{n-j}\right) \\ &= \sum_{j=0}^{n-1} j \sum_{k=1}^{n} h_k \left(\frac{n-1}{n}\right)\ldots\left(\frac{n-j}{n-j+1}\right)\left(\frac{1}{n-j}\right).\end{aligned}$$

Simplifying the above and since H is a distribution, we get

$$\mathrm{E}[Z]|_{H,S} = \sum_{j=0}^{n-1} j \sum_{k=1}^{n} h_k \frac{1}{n} = \frac{n-1}{2}.$$

■

However, with non-uniform hider distributions, the appropriate seeker distribution is the more complex Wallenius non-central hypergeometric distribution, which is difficult to solve analytically (but numerical solutions are tractable in some cases: see [Fog08]). Secondly, it can be argued that for real problems involving multiple rounds of experimentation, sampling with replacement is in fact the correct model, since hypotheses discarded in one experiment, may nevertheless become viable options on later ones. Thirdly, it is a well-known practicality, that the differences do not matter if n is large. These caveats notwithstanding, the results in the next section can be seen as upper-bounds on those obtainable when sampling is done without replacement (even with a non-central hypergeometric distribution).

The following result is a consequence of the fact that H and S are distributions:

Theorem 3.1 (Expected misses is convex)**.** *Given a distribution H, and a positive distribution S,*

$$\mathrm{E}[Z]|_{H,S} = \sum_{i=1}^{n} \frac{h_i}{s_i} - 1$$

is convex.

Proof: The problem can be posed as a constrained optimisation problem in which the objective function that is to be minimized is

$$\mathrm{E}[Z]|_{H,S} \equiv f(S) = \sum_{i=1}^{n} \frac{h_i}{s_i} - 1.$$

For notational simplicity, we have denoted the function $\mathrm{E}[Z]|_{H,S}$ as $f(S)$. Our objective is to minimize the function f given any hider distribution H.

A practical test for convexity of a function is to check whether the function f has non-negative second derivative for all s_i in a given interval of f. A twice differentiable function, if convex, would curve-up without any inflection points in the given interval.

Note that f is a scalar function of multiple variables; that is, $S = (s_1, s_2, \ldots, s_n)$. Let ∇f denote the result of the partial derivative of f with respect to S. The result is clearly the vector

$$\begin{aligned}\nabla f &= \left(\frac{\partial f}{\partial s_1}, \frac{\partial f}{\partial s_2}, \ldots, \frac{\partial f}{\partial s_n}\right) \\ &= \left(-\frac{h_1}{s_1^2}, -\frac{h_2}{s_2^2}, \ldots, -\frac{h_n}{s_n^2}\right).\end{aligned}$$

Now, computing the double derivative of f with respect to S, denoted by $\nabla^2 f$, we get the following Hessian matrix:

$$\nabla^2 f = \nabla(\nabla f) = 2 \begin{bmatrix} \frac{h_1}{s_1^3} & 0 & \ldots & 0 \\ 0 & \frac{h_2}{s_2^3} & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & \frac{h_n}{s_n^3} \end{bmatrix}.$$

Since, $\forall i, h_i \geq 0, s_i > 0$, we can claim that $\nabla^2 f$ has all non-negative second derivative components which proves the convexity of f. ■

It follows that there is an optimal seeker distribution S^* such that Equation (3.1) is minimised.

Theorem 3.2 (Optimal S given H)**.** *Let H be a discrete distribution as in Lemma 3.1. The optimal seeker distribution $S^* = (s_1^*, s_2^*, \ldots, s_n^*)$ where $s_i^* = \frac{\sqrt{h_i}}{\sum_{j=1}^{n} \sqrt{h_j}}$, $1 \leq i \leq n$.*

Proof: We will write $\mathrm{E}[Z]|_{H,S}$ as a function of S i.e. $f(S)$. Our objective is to minimise $f(S) = \sum_{i=1}^{n} \frac{h_i}{s_i}$ subject to the constraint $\sum_{i=1}^{n} s_i = 1$. The corresponding dual form

(unconstrained) of this minimisation problem can be written as

$$g(S,\lambda) = \sum_{i=1}^{n} \frac{h_i}{s_i} + \lambda \left(1 - \sum_{i=1}^{n} s_i\right) \tag{3.4}$$

To obtain the optimal values of S and λ, we set $\frac{\partial g}{\partial s_i} = 0$ for $i = 1, \ldots, n$, and $\frac{\partial g}{\partial \lambda} = 0$. This gives $-\frac{h_i}{s_i^2} - \lambda = 0$ and $\sum_{i=1}^{n} s_i = 1$, which simplifies to $s_i = -\frac{\sqrt{h_i}}{\sqrt{\lambda}}$, $\forall i$. Substituting the value s_i in $\sum_{i=1}^{n} s_i = 1$ and the value of the parameter $\lambda = -\frac{h_i}{s_i^2}$, we get $-\frac{\sum_{i=1}^{n}\sqrt{h_i}}{-\frac{\sqrt{h_i}}{s_i}} = 1$. Simplifying the R.H.S. of Equation (3.4), we obtain the desired optimal seeker distribution S^*, $s_i^* = \frac{\sqrt{h_i}}{\sum_{j=1}^{n}\sqrt{h_j}}$, $1 \leq i \leq n$. ■

The following result follows for the special case of a uniform H:

Corollary 3.1 (S^* for Uniform H)**.** *If $H \sim Unif(1,n)$, then $S^* \sim Unif(1,n)$ and $E[Z]|_{H,S^*} = n - 1$.*

Proof: If S^* is non-uniform with $s_i^* > 0$ for all i, we have

$$\mathrm{E}[Z]|_{H,S^*} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{s_i^*} - 1 \geq \frac{n}{\sum_{i=1}^{n} s_i^*} - 1.$$

The denominator is 1 because S^* is a distribution. So, S^* must be a uniform distribution and in this case, the quantity

$$\mathrm{E}[Z]|_{H,S^*} = \sum_{i=1}^{n}\frac{1/n}{1/n} - 1 = \sum_{i=1}^{n} 1 - 1 = n - 1.$$

■

We note that this result is consistent with those presented in [Ruc91, Sto76] for the hide-and-search game, where the adversarial nature of the game requires the hider to select a uniform distribution to maximise the expected misses by a seeker.

If H distribution is non-uniform, then we note the following:

Corollary 3.2. *Given a non-uniform hider distribution H and a corresponding optimal seeker distribution S^*, we have*

$$E[Z]|_{H,S^*} = \sum_{i=1}^{n}\frac{h_i}{s_i^*} - 1 = \sum_{i=1}^{n}\left(\sqrt{h_i}\right)^2 - 1.$$

Proof: The proof follows from the series of equalities,

$$\begin{aligned}
\mathrm{E}[Z]|_{H,S^*} &= \sum_{i=1}^{n} \frac{h_i}{s_i^*} - 1 \\
&= \sum_{i=1}^{n} \frac{h_i}{\left(\frac{\sqrt{h_i}}{\sum_{j=1}^{j}\sqrt{h_j}}\right)} - 1 \\
&= \sum_{i=1}^{n} \sqrt{h_i} \sum_{j=1}^{j} \sqrt{h_j} - 1 = \sum_{i=1}^{n} \left(\sqrt{h_i}\right)^2 - 1.
\end{aligned}$$

■

With non-uniform H, the value of $\mathrm{E}[Z]$ can get substantially lower than $n-1$, which was obtained for uniform H. Thus, for non-uniform H, we find the following:

Theorem 3.3. *Let H and S^* be defined as in Theorem 3.2. Let $\mathrm{KLD}(U||V)$ denote the Kullback-Liebler divergence between distributions U and V. Then,*

$$E[Z]|_{H,S^*} = 2^{2KLD(H||S^*)+Entropy(H)} - 1.$$

Proof: The KL-divergence between the two distributions H and S^* is defined as

$$\begin{aligned}
\mathrm{KLD}(H\|S^*) &:= \sum_{i=1}^{n} h_i \log_2 \frac{h_i}{s_i^*} \\
&= \sum_{i=1}^{n} h_i \log_2 h_i - \sum_{i=1}^{n} h_i \log_2 \frac{\sqrt{h_i}}{\sum_{j=1}^{n}\sqrt{h_j}} \quad \text{(using Theorem 3.2)} \\
&= \frac{1}{2}\sum_{i=1}^{n} h_i \log_2 h_i + \log_2 \left(\sum_{j=1}^{n}\sqrt{h_j}\right)\left(\sum_{i=1}^{n} h_i\right) \\
&= -\frac{1}{2} Entropy(H) + \log_2 \left(\sum_{j=1}^{n}\sqrt{h_j}\right) \\
&= -\frac{1}{2} Entropy(H) + \log_2 \left(\mathrm{E}[Z]|_{H,S^*} + 1\right)^{\frac{1}{2}} \quad \text{(using Corollary 3.2)} \\
&= \frac{1}{2}\left[-Entropy(H) + \log_2 \left(\mathrm{E}[Z]|_{H,S^*} + 1\right)\right].
\end{aligned}$$

On simplifying, we get the required relation between $\mathrm{E}[Z]|_{H,S^*}$ and $\mathrm{KLD}(H\|S^*)$. ■

Based on the earlier result for uniform H, it is evident that if H is uniform, entropy of H will be a maximum, and the KL-divergence between H and S^* will be 0. As entropy of H decreases (H is non-uniform), although the KLD term increases, the overall expression has a minimum for $S = S^*$. We provide some further intuition about the optimal seeker S^* for the case of non-uniform H. In this case, some boxes will have higher than uniform probability of containing the target, and some will have less than uniform probability. In

order to minimise misses, the seeker also needs to look at unlikely boxes. Specifically, an unlikely box has to be selected with higher probability than that used by H to avoid many misses if the box contains the target. Therefore, in general, the optimal S distribution needs to have higher probabilities on unlikely boxes than the hider; and to compensate, lower probabilities on the likely boxes. This pushes the seeker closer to the uniform distribution, and therefore usually with higher entropy than H.

If, on the other hand, H is uniform then the seeker simply cannot use any higher entropy distribution and has no choice but to follow the hider's uniform distribution.

All these results require H to be known. In practice, the question is what can be done if H is not known. We consider this in the following section for the case of non-uniform H.

Hider Distribution Unknown

In almost all practical situations, we do not know H. What can be done in such cases? Based on the results of the previous section, we will begin by assuming, for efficient target identification, that H is non-uniform. We define a 2-partition of the locations based on H as follows: the U partition contains locations that have probability greater than uniform probability ($> \frac{1}{n}$) and the rest forms the V partition. Furthermore, we assume the following:

- Any target location has probability greater than $\frac{1}{n}$. All targets are to be found in the "target partition". W.l.o.g., we can take the target partition to be U; and
- The size of the target partition is known to be the proportion p ($0 < p < 1$).

The sample size (denoted[1] by s), which with high probability, will result in boxes from the target partition can be calculated.

Lemma 3.4 (Samples from the target partition)**.** *Let H be a distribution over a set X. Let U denote the set of boxes $\{x \in X : h(x) > 1/n\}$ and $L = X - U$. Without loss of generality, let the target(s) be in U, and let $p = |U|/|X|$ (> 0). Then a sample of size*

$$s \geq \frac{\log(1-\alpha)}{\log(1-p)}$$

will contain at least one element of U with probability $\geq \alpha$.

Proof: The probability that a randomly drawn box is not in the U partition is $(1-p)$. The probability that in a sample of s boxes, none are from the U partition is $(1-p)^s$, and

[1]This use of s should not be confused with the search-distribution probability s_i for a location i.

therefore the probability that there is at least 1 box amongst the s from the U partition is $1-(1-p)^s$. We want this probability to be at least α. That is,

$$1-(1-p)^s \geq \alpha.$$

With some simple arithmetic, it follows that

$$s \geq \frac{\log(1-\alpha)}{\log(1-p)}.$$

■

With the assumptions above, it is evident that the higher the number of targets, the greater the value of p, and the smaller the sample size s. That is, with many possible locations containing targets, it is easier to find at least one target location.[2] Of course, sampling only guarantees, with high probability, that there will be at least one box from the target partition. Thus, not all boxes in the sample will be from the target partition; and of those, not all may contain a target.

A procedure that uses the sample to search for the targets is in Procedure 2. The procedure takes as inputs: X, a set of boxes; p (> 0), the proportion of boxes in the target's partition; α, lower bound on probability of finding an element from the target's partition; t, an upper bound on the iterations of the sampler (This is same as $MaxDraws$ in Procedure 1); function $Hider : X \rightarrow \{TRUE, FALSE\}$ such that $Hider(x)$ is $TRUE$ for box x if a ball is in box x, and $FALSE$ otherwise; and returns a box with a target and the number of misses.

Procedure 2 The Sampling Procedure

```
1:  procedure SAMPLER(X, p, α, t, Hider)
2:      done ← FALSE
3:      m ← 0
4:      s ← ⌈log(1−α) / log(1−p)⌉
5:      while ¬done do
6:          Sample ← Draw(Unif, s, X)          ▷ Draw a sample of s boxes from X
7:          X′ ← {x : x ∈ Sample and Hider(x) = TRUE}
8:          if X′ ≠ ∅ or m > t then
9:              done ← TRUE
10:         m ← m + 1
11:     x ← Draw(Unif, 1, X′)
12:     return (x, m)
```

The following issues with this procedure are apparent immediately:

[2] We note that a similar argument is used in [HZJ07] to identify possibly good solutions in discrete event simulations; and is proposed for use in Inductive Logic Programming (ILP) in [Sri99b]. Both do not explicitly relate this to a distribution model, as is done here.

- Since sampling is done with replacement, in the worst case, the procedure can end up drawing many more than $n-1$ samples before finding the hider, unless the bound t stops the procedure before this happens;
- If $Hider(x) = FALSE$ for all $x \in B$ (that is, there is no ball), then the procedure will not terminate until the bound t is reached; and
- This procedure does not take into account the boxes which are already sampled (that is, the boxes are drawn independently of each other).

Obvious corrections for the first two issues are either to bound the maximum rounds of sampling allowed; or to use sampling without replacement (in experiments in this chapter, we adopt the former). One way in which t could be assigned is as follows: Let β denote a lower bound on the probability of obtaining a hider in t trials, each with sample s determined by p and α (that is, the probability of identifying a box with a ball is at least α). It is not hard to derive that if $t \geq \frac{\log(1-\beta)}{\log(1-\alpha)}$ then the probability of identifying a box with a ball in t trials will be at least β. The number of boxes after t rounds of sampling is clearly $s \times t$. To address the third issue, sampling can be made conditional on boxes already obtained (that is, adopting a Markov model): we do not pursue this further in this dissertation.

The procedure also assumes that there can be more than one box $x \in X$ with $Hider(x) = TRUE$, and that it is sufficient to find any one of these boxes. This assumption about multiple hiders often makes sense in practice, when we are happy with near-optimal solutions. We will demonstrate the applications of the above theoretical findings using simulations below.

Distributional Model of Discrete Search: Simulations

The results from simulations are as follows:

Known hider distribution Results of simulations with known H distributions are in Figure 3.6. These results confirm the following:

1. The seeker distribution obtained in Theorem 3.2 have higher entropy than the hider distribution (as expected).
2. For hider distributions other than the uniform, it is possible to obtain seeker distributions that make fewer expected misses than $n-1$ (which is the value obtained if the seeker knows the hider's distribution). Further, as predicted theoretically, this expected value is lower as n increases.
3. For low-entropy hiders, it is possible to obtain substantially low numbers of expected misses with the distribution obtained in Theorem 3.2.

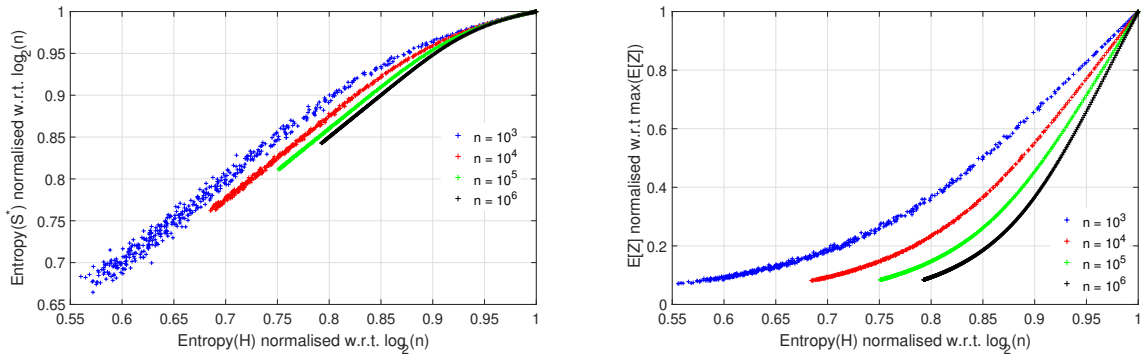


Figure 3.6: Known Hider Distribution: (Left) Entropy of the hider distribution vs. Entropy of the seeker distribution, (Right) Entropy of the hider distribution vs. Expected number of misses by the seeker.

Unknown hider distribution Figure 3.7 shows the expected number of misses observed using Procedure 2 for varying values of n ($|X|$) and p, the proportion of boxes in the H's target partition. In all cases, $\alpha = 0.95$ (that is, we want to be 95% sure of obtaining at least one box from the H's target partition on each iteration of sampling in Procedure 2). We note the following:

1. The number of misses increases as the number of hidden objects (balls) decreases (refer supplimentary material for the theory). This is as expected.
2. The expected number of misses: (a) is substantially less than the worst case of $n - 1$; (b) decreases as p increases; and (c) decreases as n increases. The last finding may seem surprising in the first instance. However, we note that the sample size in Procedure 2 does not change with n, but the actual number of balls for a given abscissa is much larger for larger values of n. A simple pigeonhole argument therefore suffices to explain the empirical result of finding balls quicker as n increases. This behaviour is also consistent with the theoretical case predicted when the hider is known (and observed empirically in Figure 3.6).

The discussions so far in this chapter forms various aspects of relational features and two ways of sampling relational features obtained using Inductive Logic Programming (ILP). Next we discuss how these relational features are used to construct standard fully-connected deep neural networks, called Deep Relational Machines (DRMs).

3.6 Application to Deep Relational Machines (DRMs)

A Deep Relational Machine (DRM [Lod13]) is a multilayer perceptrons (MLPs) constructed using relational features as inputs. That is, the inputs to the MLP network are Boolean-valued feature vectors obtained using propositionalisation of relational features.

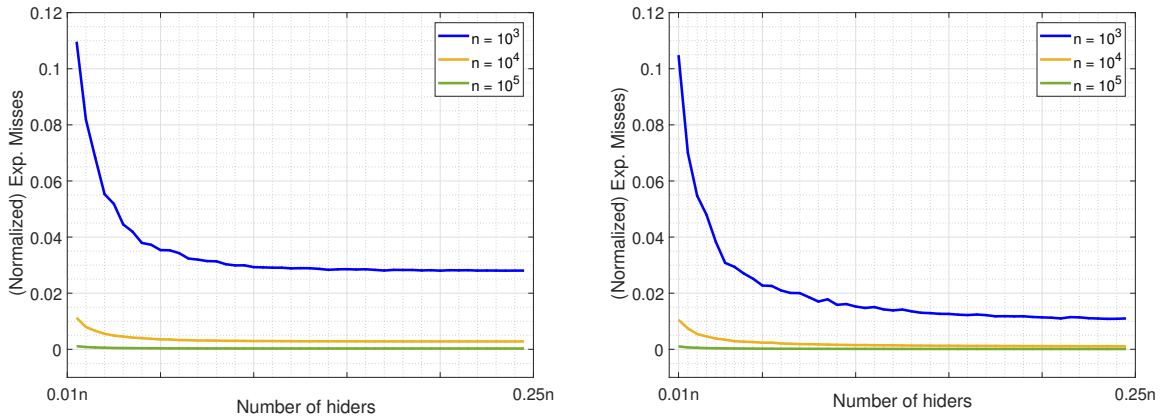


Figure 3.7: Unknown hider distribution, with more than 1 hider: (Left) $p = 0.1$, (Right) $p = 0.25$. That is, the proportion of boxes in the H's partition of the step-approximation is known to be 10% and 25% of n. The number of balls is varied from 1% of n to 25% of n (X-axis). The expected number of misses is on the Y-axis.

DRMs are a simple kind of neuro-symbolic architecture [BGB+17, dGL20] constructed from relational data and symbolic domain-knowledge. Figure 3.8 shows a diagrammatic representation of the process of constructing DRMs. There are two steps for the construction of a DRM: (a) selection of relational features and (b) construction of a multilayer perceptron using the features selected in (a). The input to an MLP is a Boolean-valued feature-vector representing a relational example using propositionalised encoding of relational features $f_1, \ldots, f_d$. This is diagrammatically shown in Figure 3.9. We now investigate the application of the utility-based sampling developed in the previous section.

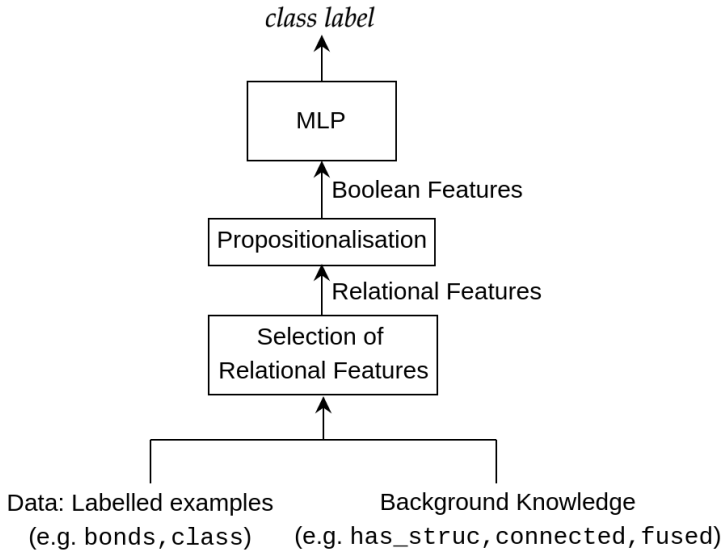


Figure 3.8: Diagrammatic Representation of a Deep Relational Machine (DRM). The examples shown at the bottom are the predicates in data and background knowledge. The selection of relational features includes the feature construction and sampling steps.

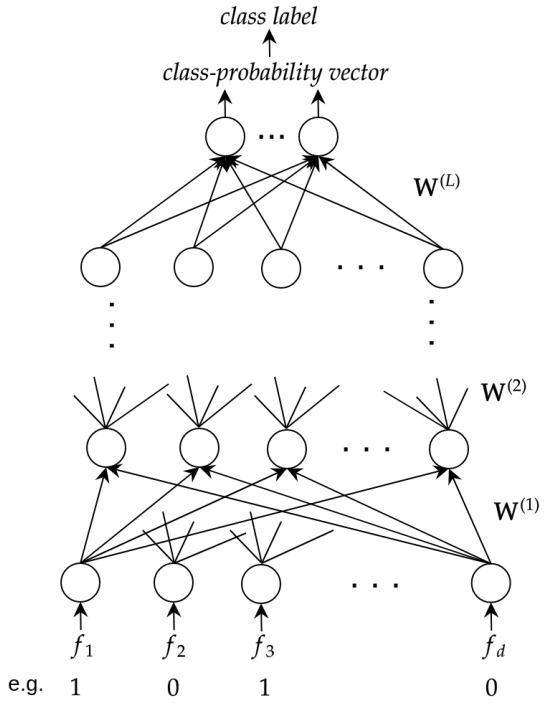


Figure 3.9: Diagrammatic Representation of Constructing a DRM using relational features and propositionalisation. The inputs to an MLP represent a Boolean-valued feature vector obtained by propositionalisation of the relational features $f_1, \ldots, f_d$. The parameters of the MLP are denoted as: $\mathbf{W}^{(\ell)}$, where ℓ denotes the layer index. In implementations, any $\mathbf{W}^{(\ell)}$ may contain an additional set of parameters called "bias weights" for which the inputs are always 1. The output of the MLP is an a class-label obtained from the class-probability vector of length k, where k is the number of classes.

3.7 Empirical Evaluation

Our running goal through out the thesis is to investigate the hypothesis that providing domain-knowledge can improve the predictive performance of deep neural networks. We now examine this in the context of DRMs that are provided with relational features constructed for large-scale datasets for which extensive domain-knowledge that is already available.

3.7.1 Aims

The aim of this empirical study is to evaluate the performance of DRMs that includes domain-knowledge through the use of relational features and propositionalisation. That is,

- We investigate whether the performance of a DRM constructed using relational features that includes domain-knowledge is better than the performance of a DRM constructed using relational features that does not include domain-knowledge.
- We investigate whether the utility-based sampling is better than uniform-random

sampling. Here we compare the performance of DRMs constructed using relational features sampled using (a) uniform-random sampling and (b) hide-and-seek sampling.

3.7.2 Materials

Data

In this chapter, and in all the research conducted in this dissertation, we focus on classification problems arising in the field of drug discovery. In particular, these datasets represent an extensive drug evaluation effort at the National Cancer Institute (NCI: https://www.cancer.gov/). Each dataset represents experimentally determined effectiveness of anti-cancer activity of a chemical compound against a number of cell-lines [MOHU03]. These datasets correspond to the concentration parameter GI50, which is the concentration that results in 50% growth inhibition of tumour cells. Some of the datasets have been used in various data mining studies, such as in a study involving the use of graph kernels in machine learning [RSSB05]. These datasets are also used in the study of LRNNs [SAZ+18, Šou20]. We use 73 such anti-cancer datasets, collectively referred to as NCI-50 datasets, for our study on the DRMs. A dataset consists of several hundred to thousands of data instances, each representing a chemical compound in the relational representation of atoms and bonds. Figure 3.10 summarises these datasets.

# of datasets	Avg. # of instances	Avg. # of atoms per instance	Avg. # of bonds per instance	% of positives
73	3032	24	51	0.4–0.9

Figure 3.10: A summary of the NCI-50 datasets (Total number of instances is approx. 220,000). Each instance in a dataset represents a chemical compound in atom-bond representation, along with its associated anti-cancer activity (positive or negative). Positive activity means the compound results in 50% growth inhibition of the tumor cells and negative activity means otherwise.

Each instance in a dataset is represented as a set of `bond` facts along with its anti-cancer activity (positive:1 or negative:0). A bond fact has an arity 6 and is of the form as shown below. Here a chemical compound is referred to using some identity `CompoundID`. Similarly, an atom i in the chemical compound is identified by its positions (a number) in the compound (`AtomiID`) and types (`AtomiType`). The bond is referred to by `BondType`.

$$\texttt{bond}(\texttt{CompoundID}, \texttt{Atom1ID}, \texttt{Atom2ID}, \texttt{Atom1Type}, \texttt{Atom2Type}, \texttt{BondType}).$$

The target class label of a compound is represented with a `class` fact:

$$\texttt{class}(\texttt{CompoundID}, \texttt{Label}).$$

Therefore, a relational data instance (say, `m1`) in relational representation (more details on this representation is in Appendix A) looks like the following:

```
class(m1,pos).
bond(m1,29,26,car,car,ar).
bond(m1,14,11,car,c3,1).
...
```

Background Knowledge

We used the background knowledge used in [VCVD02, ADL+06] with minor modifications to avoid redundant computation and for tractable computation (essentially trading-off completeness for efficiency). It consists of details of various atomic properties, various chemical structures in a chemical compound, such as functional groups and rings. The organisational levels of the background knowledge are as shown in Figure 3.11. The definitions [GC10] of functional groups and rings which are used in this work are much more elaborate than have been reported in the ILP literature. The definitions used were originally developed for tackling industrial-strength problems by the biotechnology company PharmaDM and consist of multiple hierarchies as shown in Figure 3.12 and Figure 3.13. Many of these functional groups consists of multiple sub-functional groups with 'is-a' relationship. For example, hydroxylamine is a amine group. The background knowledge consists of information on accepting and donating groups. For example, methyl group is an inductive donating chemical group. Inductive accepting groups are alcohol, amine, halide, nitro group, methoxy group, acylhalide, acid_car, keton, aldehyde, and nitrile. The ring hierarchy consists of aromatic and non-aromatic rings, which are further divided into hetero or non-hetero rings.

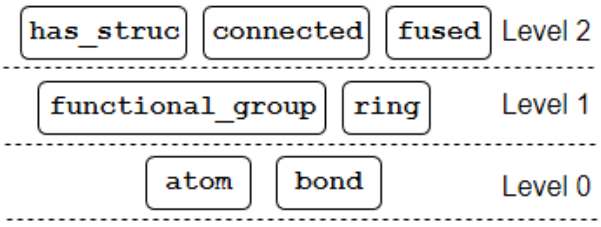


Figure 3.11: Levels of organisation of the background knowledge. Level 0 corresponds to the standard atom and bond information for the molecular compounds; Level 1 refers to the existence of various functional groups and ring structures; Level 2 knowledge is inferred further from Level 0 and 1.

For proprietary reasons, we are not able to show the actual definitions used. However, we are able to show the results of using the definitions of functional groups and rings. In the predicates shown below, `AtomIDs` refers to a list of atoms (their positions in the

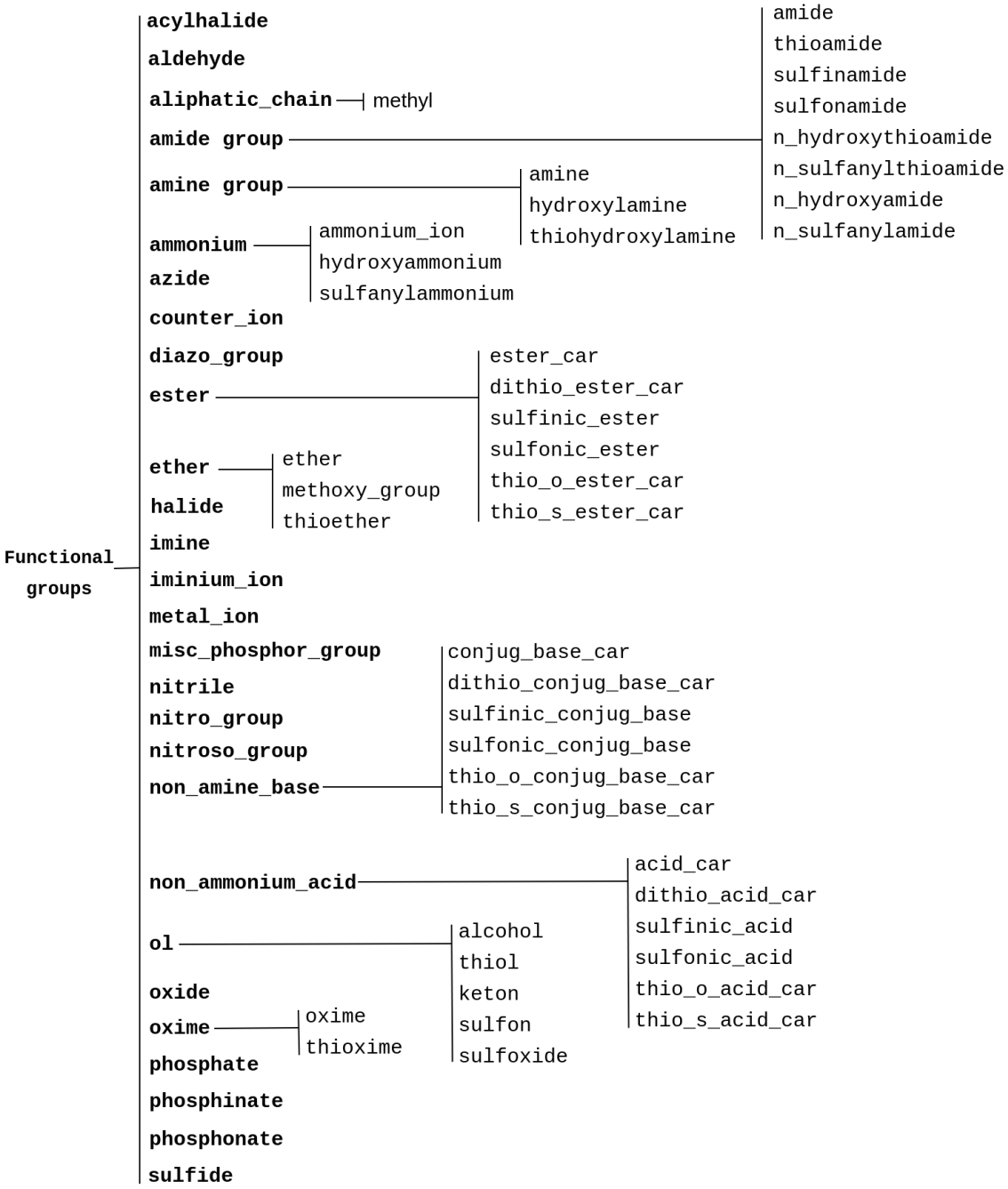


Figure 3.12: Hierarchy of various functional groups in the background knowledge.

compound), `Length` refers to the length of the list, `AtomIDs` and `Type` refers to the type of structure (functional group or ring). For efficiency, we have restricted the background predicate definition of ring predicates to produce rings of maximum length 8.

`functional_group`(`CompoundID`, `AtomIDs`, `Length`, `Type`).

`ring`(`CompoundID`, `RingID`, `AtomIDs`, `Length`, `Type`).

The definitions of functional groups and rings are used to infer the presence of compos-

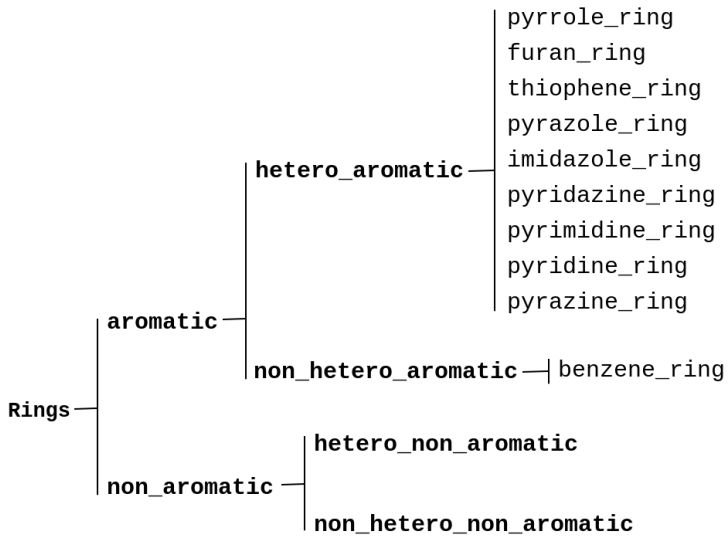


Figure 3.13: Hierarchy of various ring structures in the background knowledge.

ite structures, defined using higher-level relations. In this chapter, these relations define the presence of fused rings, connected rings, and substructures. They are represented by the following relations:

- `has_struc`(`CompoundID`, `AtomIDs`, `Length`, `Struc`): A compound with `CompoundID` contains a structure `Struc` of length `Length` containing `AtomIDs`.
- `fused`(`CompoundID`, `Struc1`, `AtomIDs1`, `Struc2`, `AtomIDs2`): A compound identified with `CompoundID` contains a pair of fused structures `Struc1` and `Struc2` with `AtomIDs1` and `AtomIDs2` respectively (that is, there is at least 1 pair of common atoms).
- `connected`(`CompoundID`, `Struc1`, `AtomIDs1`, `Struc2`, `AtomIDs2`): A compound identified with `CompoundID` contains a pair of structures `Struc1` and `Struc2` with `AtomIDs1` and `AtomIDs2` respectively that are not fused but connected by a bond between an atom in `Struc1` and an atom in `Struc2`.

Algorithms and Machines

The datasets and the background knowledge are written in Prolog. The relational features are constructed and sampled using the ILP engine, Aleph [Sri01]. The sampling step is based on the stochastic clause sampling procedure available within Aleph, for which an extensive study is available in [Sri99a]. The propositionalisation of the relational features is carried out using the facility availability within Aleph. We used Python based Keras [C+15] with Tensorflow as backend [AA+15] for implementing the deep neural networks.

All the primary experiments presented in this chapter (that is, feature construction, learning of the deep neural networks, etc.) are conducted in Linux (Ubuntu) based Dell

workstation with 64GB of main memory, 16 processing cores, a single 2GB NVIDIA Graphics Processing Unit (GPU).

3.7.3 Method

Let D be a dataset of labelled relational data-instances $\{(e_1, y_1), \ldots, (e_N, y_N)\}$, where y_i is the class-label associated with an example e_i. We also assume that we have access to background knowledge B, a set of mode declarations M, a depth-limit d. Our method for investigating the performance of DRMs is simplified below.

(1) Randomly split D into D_{Tr} (training set) and D_{Te} (test set);

(2) Construct a DRM on D_{Tr} using propositionalisation of relational features obtained using random sampling without background knowledge ;

(3) Construct a DRM on D_{Tr} using propositionalisation of relational features obtained using random sampling with background knowledge ;

(4) Construct a DRM on D_{Tr} using propositionalisation of relational features obtained using hide-and-seek sampling with background knowledge ;

(5) Obtain the predictive performance of DRM constructed in step (2) on D_{Te} ;

(6) Obtain the predictive performance of DRM constructed in step (3) on D_{Te} ;

(7) Obtain the predictive performance of DRM constructed in step (4) on D_{Te} ;

(8) Compare the performance obtained in step (6) and step (5);

(9) Compare the performance obtained in step (7) and step (6).

The following additional details are relevant to the relational feature construction and sampling steps:

- In all our experiments, "with background knowledge" (or "with domain-knowledge") would refer to the inclusion of predicates in the background knowledge as described in subsection 3.7.2; and, "without background knowledge" (or "without domain-knowledge") would mean that the only predicates used are the `bond` predicates as described in subsection 3.7.2.

- For our experiments that compares DRMs with and without background knowledge, the bound on the number of relational features to be sampled in our simple random sampling procedure (The input $MaxDraws$ in Procedure 1) is set to 5000.

- For the hide-and-seek sampling of relational features, we assume the goodness of features is their Laplace score (the Laplace score of a feature is $\frac{npos+1}{npos+nneg+2}$ where $npos$ and $nneg$ are the number of positive and negative instances, respectively, for which the feature is $TRUE$); and any feature in the top 50-percentile of scores is acceptable as an input feature for the DRM (that is, $p = 0.5$). That is, target features are to be found in the top 50-percentile of possible relational features is taken to be a "hidden ball". This gives a small bias towards good features, but does not restrict the DRM from identifying better features by combination in its hidden layers. In all experiments, α is fixed at 0.95. Therefore, the value of the sample size (s) in Procedure 2 (Line 4) is 5. That is, the number of features sampled before selecting a good feature is very small.

- For comparing the performance of DRMs (with hide-and-seek sampling) against that of DRMs (with random sampling), we vary the value of $MaxDraws$ from 50 (minimum) to 5000 (maximum). The exact values are in the results section. These features have no more than 3 literals in the body. The resulting Boolean feature vector representation after propositionalisation of the relational features is sparse.

The following details are relevant to the structure of a DRM:

- The number of inputs in the deep net is the number of relational features obtained by the procedure described earlier (for the experiments conducted in this dissertation, this number is as shown in Figure 3.10). The depth (number of layers) of a deep neural network and size of each hidden layer remains arbitrary as there is no theory in deep learning to guide in this aspect. However, a standard strategy to search a "good" model structure is via cross-validation.

- Since we are dealing with sparse binary representations of input, we expect that a neural network with a small number of hidden neurons should be sufficient to deal with learning the input–output mapping function. Therefore, in our cross-validation-based structure tuning, we allow networks varying from 1 to 4 hidden layers.

- The number of neurons in each hidden layer is from a small set (here $\{5, 10\}$). The neurons in the hidden layers are rectified linear units (ReLU: It is a scalar function defined as $ReLU(z) = \max(0, z)$). The output layer has a single neuron with a sigmoid activation function.

- To mitigate issues of over-fitting, we apply dropout [SHK+14] after every layer in the network except the output layer. The dropout rate is set to 0.5.

The following details are relevant to the construction (training) and the evaluation (testing) of the DRMs:

- We use the Adam optimiser [KB15] with the following parameters for learning the parameters (weights) of the DRM: The learning rate is fixed at 0.001 and the other hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $decay = 0$.

- The loss function minimised during the training of a DRM is cross-entropy between the true class-label (encoded as an one-hot vector: represents the true probability distribution over the class labels) and predicted class-probability vector (represents a computed probability distribution over the class labels).

- The mini-batch size is set to 32 and the maximum number of training epochs is 1200.

- We use early-stopping [Pre98] to control overfitting the model during its training.

- The evaluation metric is accuracy of the trained model on a test set (a subset of the whole dataset as described earlier): a randomly drawn test set consisting of 30% of the total number of instances.

- Comparisons of the predictive performance of DRMs are conducted using Wilcoxon signed-rank test, using the standard implementation within MATLAB.

3.7.4 Results

The results of the experiments here are in Figure 3.14 through to Figure 3.17. The principal qualitative conclusions from these tabulations are as follows:

1. The inclusion of background predicates makes a substantial difference to DRM performance, suggesting that DRMS are able to utilise domain-knowledge to improve performance, without requiring an increase in data.

2. DRMs with hide-and-seek selection perform better than those with the simple random sampling strategy.

Recall that our primary objective in this chapter is that inclusion of domain-knowledge into deep neural network results in improvement of predictive performance. Here the null hypothesis is that the predictors being compared have performance values from the same population (that is, differences in performance will be symmetrically distributed around 0). For the actual differences observed, the p-value is tabulated in Figure 3.15 demonstrating that this result is statistically significant (based on 71 wins in 73 different cases). This means, the DRMs constructed from the relational features are able to utilise the provided domain information (via the domain predicates in the background knowledge).

The performances of DRMs, constructed with uniformly sampled relational features and utility-based sampled features (hide-and-seek), are compared with regard to different

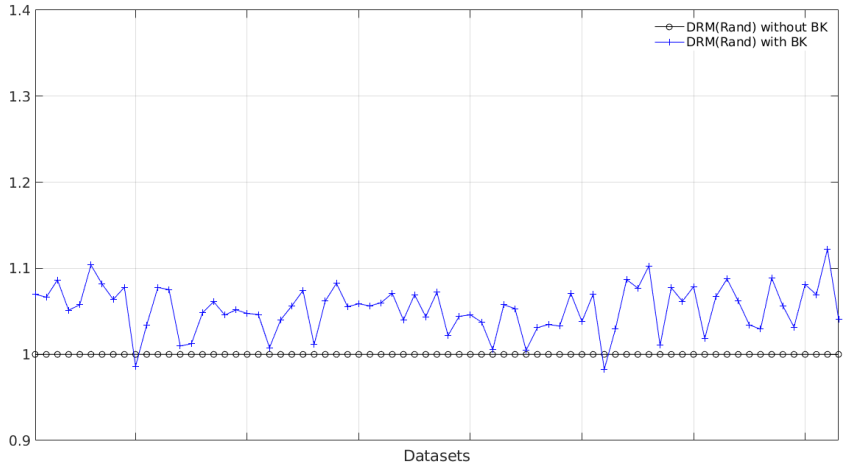


Figure 3.14: Improvements in predictive performance of DRMs, when provided with domain-knowledge through propositionalisation of relational features constructed using simple random sampling strategy by an ILP engine. The average number of relational features across the datasets is roughly 3800. Here X-axis represents the datasets (total 73 NCI datasets), and Y-axis shows the gain in predictive performance with respect to the baseline. Baselines ("1") are the models without domain-knowledge. The corresponding quantitative comparison is shown in Figure 3.15.

Model	Higher/Lower/Equal (p-value)
DRM (Rand)	71/2/0 (< 0.001)

Figure 3.15: Comparison of predictive performance of DRM (Random Sampling) with and without domain-knowledge. The average number of relational features across the datasets is roughly 3800. The tabulations are the number of datasets on which DRM has higher, lower or equal predictive accuracy (obtained on a holdout set) than DRM without domain-knowledge. Statistical significance is computed by the Wilcoxon signed-rank test.

number of features in Figure 3.17. It is evident that other than the last row, DRMs with hide-and-seek selection perform better than those with the simple random sampling strategy. This suggests that if the number of input features are restricted to being small (due to limitations of hardware, or for reasons of efficiency), then hide-and-seek sampled features would be a better choice. It is curious that with a large number of input features (here, $>$ 3500 or so), there is no significant statistical advantage from hide-and-seek sampling. This may be due to the fact that the DRM has a sufficiently diverse set of input features from uniform selection to be able to construct good features in its intermediate hidden layers.

Some Additional Comparisons

We now turn to the question: How does DRMs perform against some recent approaches to learning from relational data? To answer this question, we chose two different approaches: (1) an approach that represents weighted first-order logic programs with neural networks,

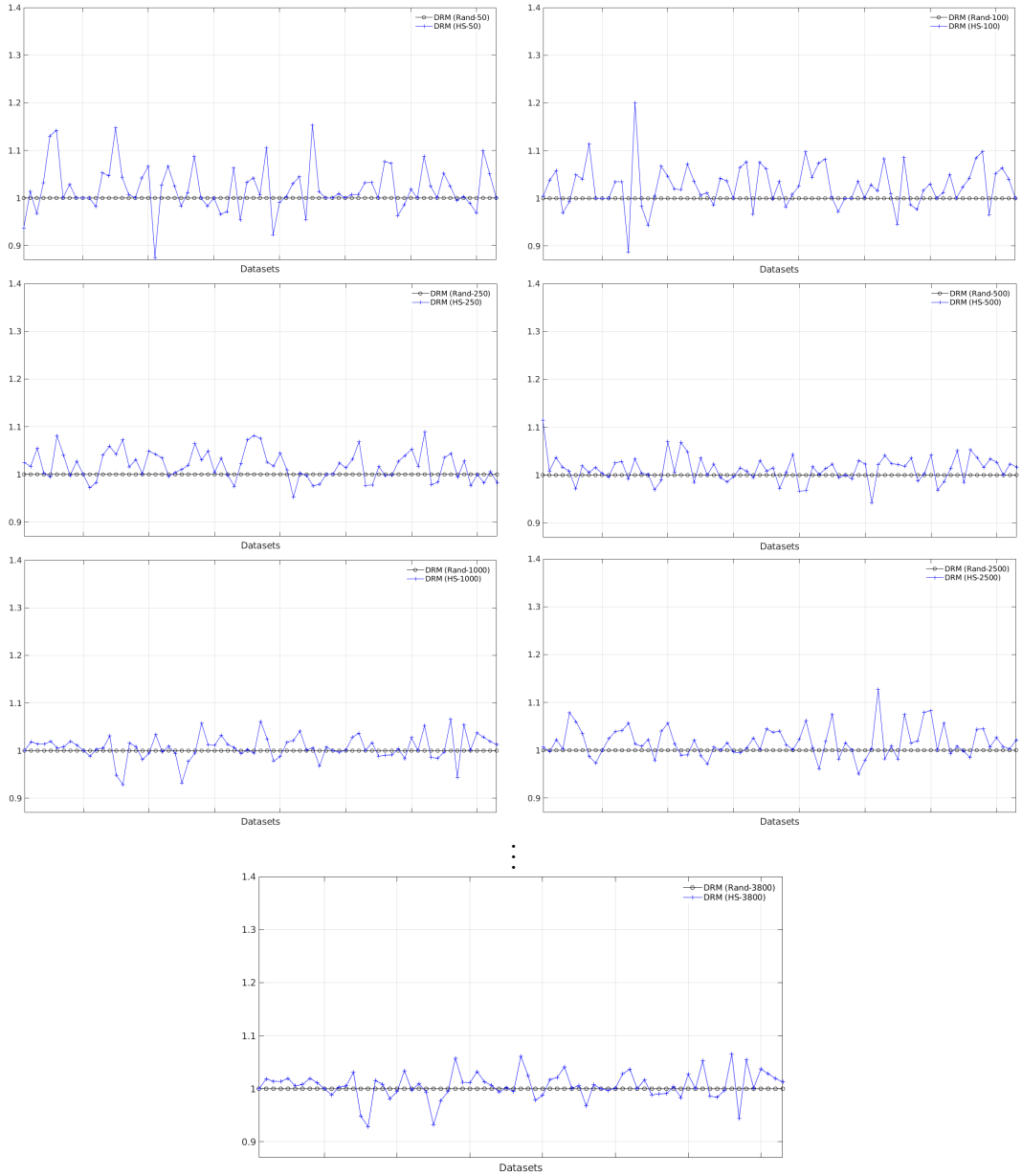


Figure 3.16: Qualitative comparison of predictive performance of DRMs (Hide-and-Seek:"HS" vs Random:"Rand") with different number of relational features: $\{50, 100, 250, 500, 1000, 2500, 3800\}$; The number 3800 is to match the average number of features sampled using simple random sampling. Here X-axis represents the datasets (total 73 NCI datasets), and Y-axis shows the gain in predictive performance with respect to the baseline. Baseline here is the normalised performance of DRM-Rand: the "1" line. The corresponding quantitative comparison is shown in Figure 3.17.

called LRNNs [SAZ+18], and (2) an approach that constructs deep neural networks using propositionalisation of literals in bottom-clauses in ILP, called CILP++ [FZG14]. Note that LRNNs do not use an explicit propositionalisation steps. CILP++ uses an explicit propositionalisation step called the Bottom-Clause Propositionalisation or BCP [FZG14] and the feature construction step does not involve any form of stochastic sampling. The feature constructed by BCP are Boolean, and the representation turns out to be very

# of Features	Higher/Lower/Equal (p-value)
50	43/18/14 (< 0.01)
100	50/14/9 (< 0.01)
250	48/21/4 (< 0.01)
500	51/21/1 (< 0.01)
1000	44/25/4 (< 0.01)
2500	50/21/2 (< 0.01)
3800	39/22/1 (0.22)

Figure 3.17: Comparison of predictive performance of DRM constructed with relational features sampled using hide-and-seek sampling strategy against DRM constructed using relational features sampled using simple random sampling. The last row contains 3800 features to match the average number of features sampled using simple random sampling. The tabulations are the number of datasets on which DRM(Hide-and-Seek) has higher, lower or equal predictive accuracy (obtained on a holdout set) than DRM(Rand). Statistical significance is computed by the Wilcoxon signed-rank test.

sparse, with the dimension ranging from 18000 to 52000 across our 73 datasets. For our experiments here, we construct MLPs using these Boolean features, a detailed description of our experimental setup is provided in section B.1. In our tabulations, we call the MLP model constructed with BCP features as BCP-MLP. In Figure 3.18, we provide quantitative comparisons of our DRMs against LRNNs and BCP-MLP. The results show that the DRMs, when provided with domain predicates, perform better than both these approaches. However, we adopt a conservative stand here while comparing the performance of DRMs against LRNNs despite low p-values. First, we note that LRNNs do not have access to a large set of domain predicates, as has been used in this work. It only uses some definitions of n-membered rings [Šou20]. Secondly, the reader is no doubt aware of the usual precautions when interpreting p-values obtained from multiple comparisons.

DRM (Hide-and-Seek) # of features	Accuracy (DRM vs. other methods) Higher/Lower/Equal (p-value)	
	LRNN	BCP+MLP
3800	68/5/0 (< 0.001)	69/2/2 (< 0.001)

Figure 3.18: Comparison of predictive performance of DRM against LRNN [SAZ+18] and BCP+MLP [FZG14]. The DRM used here is the one constructed using 3800 relational features sampled using hide-and-seek sampling. The tabulations are the number of datasets on which DRM has higher, lower or equal predictive accuracy (obtained on a holdout set) than its counterparts. Statistical significance is computed by the Wilcoxon signed-rank test.

3.7.5 Limitations of DRMs

DRMs share the principal limitations of propositionalisation approaches to ILP problems, namely: (a) Much depends on the expressive power of the features used as input; (b) For any language with sufficient expressive power, it is intractable to provide all features within the language; and (c) Recursive definitions for prediction are not necessary, and that the background knowledge is assumed to be sufficient. Of these, we set aside (c) as being a constraint inherent to this form of modelling, and focus instead on the first two limitations.

Practitioners of ILP will be well aware of relational composition of features by sharing existential variables. Thus, in our running example of the trains problem, let us assume that we have the features

$$C_1 : \forall X \ (p(X) \leftarrow \exists Y \ (has\_car(X, Y), short(Y)))$$

and

$$C_2 : \forall X \ (p(X) \leftarrow \exists Y \ (has\_car(X, Y), closed(Y))).$$

Then, depending on the data, the DRM's internal layer may not be able to distinguish correctly between

$$C : \forall X \ (p(X) \leftarrow \exists Y \ (has\_car(X, Y), short(Y), closed(Y)))$$

and

$$C' : \forall X \ (p(X) \leftarrow \exists Y, Z \ (has\_car(X, Y), has\_car(X, Z), short(Y), closed(Z))).$$

This means that to correctly capture shared relationships, we have to ensure that we include such features at the input layer (in this example, C will have to be provided as an input feature). Neural models that attempt explicitly to capture relationships among variables will not suffer from this limitation (separate internal nodes would represent C and C' for example, given C_1 and C_2 as input features).

In [SSR12], different classes of features with varying expressive power were identified. Figure 3.19 shows a significant drop in the performance of the DRM when input features are drawn from the class of "simple" features (all features in the unrestricted class of features can be constructed from some combination of features from the simple class: see [MS98, SSR12]). This suggests that DRMs require features from a fairly expressive class: we conjecture that features provided to the DRM have to be at least from the class of *independent* features identified in [SSR12] for them to have reasonable performance. This brings in the second limitation listed above. Despite recent advances in commodity hardware capabilities, the number of input features for the datasets here for the class of

independent features can range from the 10s of 1000s to the 100s of 1000s, even when features are restricted to containing no more than 3 literals (the language constraint $\mathcal{L}$), as was done here. This is beyond the routine capabilities of the existing hardware support for deep neural networks, and some form of selection appears inevitable, along with the possible limitations. Current commodity hardware does not allow us to practically use more features than a few thousand. A further limitation of DRMs is that the structural information of a relational data instance is lost due to the propositionalisation step that transforms a set of relational features to a flattened representation, which is an obvious necessity for constructing MLPs.

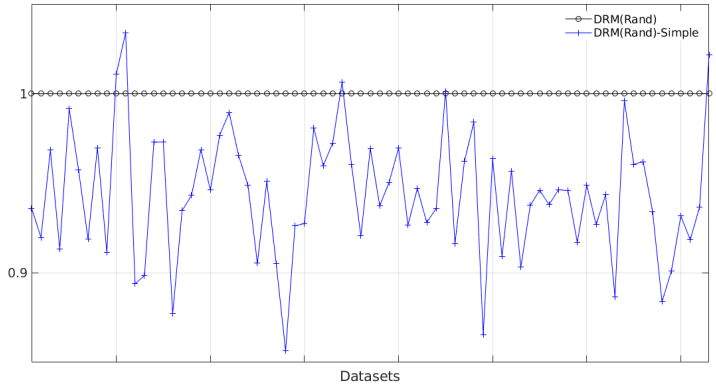


Figure 3.19: Degradation of DRM performance when expressivity of features is decreased from an unrestricted class to the class of relational features obtained using simple features as discussed in [MS98].

Computational Cost

Although not a limitation directly associated with a DRM, one critical issue is that the sampling of relational features, in particular, the hide-and-seek sampling procedure requires a significant amount of computational cost: First, to sample a lot of relational features to find a feature with good utility (Recall that we have to sample s boxes to find at least one good solution in section 3.5 and that the value of s depends on α and p. To obtain reasonably good features, the value of α should be close to 1 and p should be close to 0. Such a setting will result in a large value of s.). This process wastes significant computational effort in sampling and testing the features for their utility scores. Second, the sampling procedure involves a test for subsumption equivalence that could discard the features already selected in a final feature set. We highlight this issue with an example: For selecting 500 relational features using hide-and-seek, the sampling procedure has to sample approximately 2500 features and compute their utility scores. Further, with some other settings in our hide-and-seek sampling procedure, the number of features to be sampled could be $120,000$ to construct a DRM with good predictive performance (That is, a DRM constructed with approximately 4000 features). A more complete

quantitative comparison is tabulated in Figure 3.20. Note that the numbers mentioned here do not include the count needed due to the test for the subsumption equivalence and the costs incurred due to the propositionalisation step. Therefore, despite being simple machinery that incorporates domain-knowledge via propositionalisation, a DRM, constructed using the hide-and-seek based relational features has a significantly high computational demand. We observe a similar cost overhead for constructing MLPs with the relational features obtained using BCP: The overhead here is due to the number of input features required (in our experiments, this number tends to be 50000) to build a sufficiently good deep neural network.

# of	$\alpha = 0.99$		$\alpha = 0.95$		$\alpha = 0.90$	
Features	$p = 0.1$	$p = 0.5$	$p = 0.1$	$p = 0.5$	$p = 0.1$	$p = 0.5$
1000	43709	6644	28434	4322	21855	3322
2000	87418	13288	56867	8644	43709	6644
3000	131127	19932	85300	12966	65564	9966
4000	174835	26576	113733	17288	87418	13288

Figure 3.20: The minimum effort required to sample various number of relational features using the hide-and-seek sampling. The values tabulated are the number of relational features drawn from the large space features to obtain the number of features in the first column.

3.8 Summary

This chapter investigated the inclusion of symbolic domain-knowledge into MLPs, using a simple approach called propositionalisation of relational features. The resulting deep neural network model is called a Deep Relational Machine (DRM). In the literature, the relational features are sampled by an ILP engine using a uniform distribution, which may not guarantee that good relational features are selected for a DRM. Here we viewed the selection of the relational features as an instance of discrete stochastic search. In this game, a hider (refers to a "good" relational feature, in an implementation term) hides in one of n locations using a non-uniform hider distribution, which is unknown to the seeker, and the seeker has to find the hider in a minimal number of misses. A natural assumption would be that the seeker could minimise the number of misses if it uses a distribution equal to the hider's distribution. But, the surprising result is that this is not the case, as evident from our theoretical and empirical observations. We extended this study to design a sampling strategy for selecting "good" relational features, called hide-and-seek sampling. Our hide-and-seek sampling procedure selects relational features for a DRM with good utility-scores for a given problem. On the empirical front, the DRMs were evaluated at a small scale: The original proposal for DRMs was evaluated

on 3 datasets [Lod13]. In this chapter, we conducted a large-scale evaluation of DRMs on over 70 real-world datasets arising in the field of drug discovery. Our experiment validates the premise that the inclusion of domain-knowledge helps in the improvement of the predictive performance of deep neural networks. The results provided here present substantial statistical evidence that: (a) Despite their apparent simplicity, the predictive performance of DRMs represents substantial high-water marks for relational problems; (b) The performance of DRMs improves significantly with the inclusion of domain-knowledge; (c) DRMs are benefited significantly if provided with "good" features obtained by the hide-and-seek sampling strategy, provided that the size of the input is not very large. However, we also observed that: (d) The performance of DRMs can degrade significantly if features are not drawn from a sufficiently expressive language; and (e) There is a significant computational cost involved in the sampling of good relational features for a DRM.

Chapter 4

Simplified Inclusion of Relational Information using Vertex-Enrichment*

A key limitation of DRMs is that they require "flattening" any relational information–both in the data and in domain-knowledge–by propositionalisation. As we saw in the previous chapter, the performance of DRMs can depend quite crucially on the relational features used as propositions. If the relational feature does not contain variable co-references, for example, then there is no way of introducing this once the feature has been propositionalised. In this chapter, we move to a form of DNN that is capable of directly manipulating relational data; and introduce a first attempt at the inclusion of relational domain-knowledge into these DNNs.

Recent advances in the area of deep learning have seen a surge in research on learning from graph-structured data, including techniques for learning deep graph embeddings, generalisations of convolutions in CNNs to graphs, and adopting message-passing mechanisms for graph representation learning. These advances have led to new results in several scientific problems, including the problems dealt with in this dissertation. In general, in this chapter, we will be concerned with deep neural networks for graph-structured data, which are known as Graph-based Neural Networks or, simply, Graph Neural Networks (GNNs).

Although GNNs have been popularised recently (2017 and onwards), GNN-like models are not new. These kinds of models were first proposed in [SS97, BPZ97], and later, in [GMS05, SGT+08] the term "Graph Neural Network" or "GNN" was proposed, which referred to the presently prevailing approach of recursive aggregation of information in a graph. The major boost to the field of GNNs followed the introduction of graph

*The content of this chapter is based on the following:
T. Dash, A. Srinivasan, L. Vig, "Incorporating symbolic domain knowledge into graph neural networks", *Machine Learning*, 2021; https://doi.org/10.1007/s10994-021-05966-z.

convolution [KW17] and the notion of a graph embedding [CWPZ18, ZYZZ18]. The last few years have witnessed numerous advancements in the field of GNNs, primarily in the area of their implementations and applications. Two methodical and comprehensive surveys on GNNs are available in: [ZCH+20] and [WPC+20].

Many of the developments in GNN methodologies are powerful and successful in different real-world applications. However, there has been little or no progress in incorporating domain-knowledge into GNNs, and the focus is mainly on learning GNNs from relational data, in particular, graphs alone. Instead, it is assumed that the iterative propagation of messages (information) from one part of a graph to another via the GNNs' aggregation-and-combine mechanism would result in finding out these domain-concepts automatically, albeit inexplicable to humans [BHB+18]. We believe that incorporating domain-concepts into GNNs in some principled manner would allow constructing powerful predictive models. Further, to the best of our knowledge, GNN applications to date have been restricted to simple node-and-edge features, and have not attempted to encode any significant domain-knowledge. In this chapter, we investigate a simplified method of incorporating symbolic domain-knowledge while learning GNNs from relational data. In particular, our proposal in this chapter is to enrich graphs with domain-knowledge via a technique we call "vertex enrichment". We assess the use of domain-knowledge in this manner using the relational datasets and background knowledge studied in the previous chapter. Overall, the principal contributions of the chapter are as follows: (1) To the field of graph neural networks, this chapter presents a large-scale empirical study using real-world datasets on the inclusion of domain-knowledge. To the best of our knowledge, the number of graphs used and the number of relations encoding domain-knowledge are the most extensive to date; (2) To the field of neuro-symbolic modelling, the technique of vertex-enrichment described in this chapter provides a simple but an effective way of incorporating symbolic relations into graph-based neural networks.

4.1 Graph Neural Networks (GNNs)

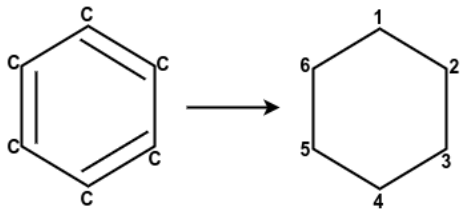
GNNs are primarily developed for learning from data represented as graphs. In this dissertation, our focus will be on GNNs concerned with graph classification. For completeness and clarity of presentation, we provide below the basics of graphs, such as directed and undirected graphs, the role of a neighbourhood function in a graph, and labelled graphs. Further, our examples in this chapter will be about molecular graphs; however, this is just for the convenience of explanation, and our proposed technique is applicable to many other scientific areas, in general.

Definition 4.1 (Graphs)**.** *A graph G is a pair (V, E) where V is a set of vertices, E is a set of edges and a subset of $V \times V$. A graph is said to be undirected if for every*

$(v_i, v_j) \in E$, *we have* $(v_j, v_i) \in E$.

In this chapter, we will be concerned with undirected graphs. We note that for such graphs, E can be represented more compactly as a set consisting of 1- or 2-element subsets of V. We will return to this later, as we extend the consideration to hypergraphs. For molecular graphs, of the kind considered here, self-loops do not occur.

Example 4.1 (Molecules as graphs)**.** *A benzene ring (shown below) can be represented as a graph, in which vertices correspond to atoms and edges correspond to bonds [MW+97].*



The graph-representation of the molecule on the left is (V, E) *where*

$$V = \{1, 2, 3, 4, 5, 6\},$$
$$E = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (4, 5), (5, 4), (5, 6), (6, 5), (6, 1), (1, 6)\}.$$

We will need the concept of the *neighbourhood* of a vertex in an undirected graph. In this chapter, by "graph" we will mean an undirected graph.

Definition 4.2 (Neighbourhood)**.** *Given a graph* $G = (V, E)$*, a neighbourhood function* σ *is a is a map from* V *to* 2^V *defined as* $\sigma(v) = \{v_i \in V : (v, v_i) \in E\}$.

Example 4.2. *For the graph in Example 4.1, we get* $\sigma(1) = \{2, 6\}, \sigma(2) = \{1, 3\}, \ldots,$ $\sigma(6) = \{1, 5\}$.

GNNs are concerned with labelled undirected graphs, that is, each vertex (and each edge) of a graph is associated with a labelling, which refers to some properties associated with that that vertex or that edge. In GNN terminology, a vertex-label or an edge-label is called a "message".

Definition 4.3 (Graph Labellings)**.** *Let* $\mathcal{V}$ *be a set of vertex labels and* $\mathcal{E}$ *be a set of edge labels. Then a vertex-labelling of a graph* $G = (V, E)$ *is a function* $\psi : V \to 2^{\mathcal{V}}$ *and an edge-labelling is a function* $\epsilon : E \to 2^{\mathcal{E}}$*. We will denote a labelled graph by the tuple* $(V, E, \sigma, \psi, \epsilon)$.

In the definition above, we do not commit to any specific data structure that should be used to implement the label set. This could be, for example, a Boolean-valued array of size $|\mathcal{V}|$.

Example 4.3. *The vertex labels of the graph given in Example 4.1 can be the atom-types (Carbon, C), and edge labels can be the bond-types (single bond: 1, double bond: 2). The label for the vertex 1 is* $\psi(1) = \cdots = \psi(6) = \{C\}$. *The labelling for the edges are* $\epsilon((1,2)) = \epsilon((2,1)) = \{2\}$, $\epsilon((2,3)) = \epsilon((3,2)) = \{1\}$ *and so on.*

Although not evident in this example, vertex- and edge-labels can have more than one element (hence the mapping to $2^{\mathcal{V}}$ and $2^{\mathcal{E}}$). This will be necessary later. We will use the term *graph* interchangeably to denote the tuple (V, E) or the tuple $(V, E, \sigma, \psi, \epsilon)$.

The defining property of a GNN is that it uses some form of neural message-passing in which messages (vertex- or edge-labels) are exchanged between vertices of a graph and updated using a neural network [GSR+17]. The message-passing process is conceptually implemented by using a function called *Relabel*. This involves an iterative update of the vertex- (and edge-) labels. The output of the function is a relabelled graph, where the vertex- (and edge-) labels are updated.

Definition 4.4 (Relabel)**.** *Given a graph* $(V, E, \sigma, \psi, \epsilon)$. *Relabel is a function that returns a graph* $(V, E, \sigma, \psi', \epsilon')$, *where the functions* ψ' *and* ϵ' *may be different to* ψ *and* ϵ.

Since we are interested in classifying graphs, that is, given a set of class labels $\mathcal{Y}$, we want to construct a function that maps a graph of the form $(V, E, \sigma, \psi, \epsilon)$ to a class-label in $\mathcal{Y}$. This requires a graph-level representation (also called a graph-embedding [Ham20]), meaning, every labelled graph is encoded as a d-dimensional real-valued feature vector. This is conceptually implemented using a function called Vec that vectorises a relabelled graph. This feature vector is then input to a (multi-layered) neural network, denoted by a function NN that maps the feature vector to a set of class-labels. The whole pipeline of graph classification is shown in Figure 4.1. Many GNN implementations, including the ones used in this dissertation, assume the graph to be undirected and ignore the edge-labelling in ϵ.

Definition 4.5 (Vec)**.** *Let* $\mathcal{G}$ *denote the set of graph-tuples of the form* $(V, E, \sigma, \psi, \epsilon)$. *For* $d \geq 1$ *and a function* $Vec : \mathcal{G} \to \mathbb{R}^d$, *a vectorisation of the graph is* $Vec((V, E, \sigma, \psi, \epsilon))$.

Definition 4.6 (GNN)**.** *Let* $NN : \mathbb{R}^d \to \mathcal{Y}$ *denote a neural network that maps a real-valued vector to a set of class-labels. Given a* $G = (V, E, \sigma, \psi, \epsilon)$, $GNN(G) = NN(Vec(Relabel(G)))$.

4.1.1 General working principle of GNNs

Let $G = (V, E, \sigma, \psi, \epsilon)$ be a labelled graph as described above. Let X_v denote a vector that represents the initial labelling (ψ) of a vertex $v \in V$. That is, X_v is the feature-vector associated with the vertex v. The relabelling function $Relabel : (V, E, \sigma, \psi, \epsilon) \mapsto (V, E, \sigma, \psi', \epsilon)$

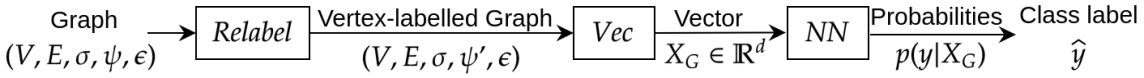


Figure 4.1: A diagrammatic representation of graph classification using a GNN. Graphs are of tuples of the form $(V, E, \sigma, \psi, \epsilon)$, where V is a set of vertices; E is a set of edges; σ is some neighbourhood function; ψ is a vertex-labelling; and ϵ is an edge-labelling. Often σ is left out, and derived from the edges in E.

(iteratively) updates the labelling of the vertices in G. This process involves two procedures: (a) AGGREGATE: for every vertex, this procedure aggregates the information from neighboring vertices; and (b) COMBINE: this procedure updates the label of a vertex by combining its present label with its neighbors', as obtained by AGGREGATE procedure. Mathematically, at some iteration k, the labelling of a vertex v (denoted by h_v) is updated as follows:

$$a_v^{(k)} = \mathsf{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right),$$
$$h_v^{(k)} = \mathsf{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right),$$

where $\mathcal{N}(v)$ denotes the set of vertices adjacent to v. Initially (at $k = 0$), $h_v^{(0)} = X_v$.

The graph vectorisation function $Vec : (V, E, \sigma, \psi', \epsilon) \mapsto \mathbb{R}^d$ constructs a vector representation of the entire graph (also called a graph embedding). This step is carried out after the representations of all the vertices are relabelled by some iterations over AGGREGATE and COMBINE. The vectorised representation of an entire graph, denoted by G here, can be obtained using a READOUT procedure that aggregates vertex features from the final iteration ($k = K$):

$$h_G = \mathsf{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right)$$

In practice, AGGREGATE and COMBINE procedures are implemented using graph convolution and pooling operations. The READOUT procedure is usually implemented using a global or hierarchical pooling operation. Variants of GNNs result from modifications to these 3 procedures: AGGREGATE, COMBINE and READOUT. The functional forms of AGGREGATE, COMBINE and READOUT are provided in an appendix section. Below we provide some brief notes on some GNN variants that are implemented in our experiments.

4.1.2 Note on GNN variants

We focus mainly on the following variants of the AGGREGATE-COMBINE procedures:

1. Spectral graph convolution, GCN [KW17]: This is a spectral method for graph convolution that uses convolutional aggregator. This is a simple and well-behaved

layer-wise propagation rule for neural network models which operate directly on graphs.

2. Multistage graph convolution, k-GNN [MRF+19]: This convolution method can perform convolution operations using multiple-sized neighbourhoods (the authors call this "higher order" graph convolution).

3. Graph convolution with attention, GAT [VCC+18]: This is a spatial method of graph convolution that uses an "attention" mechanism, that estimates the importance of vertices in the neighbourhood of a vertex.

4. Sample-and-aggregate graph convolution, GraphSAGE [HYL17]: Here the convolution procedure samples from a distribution that is constructed from feature-vectors of vertices in the neighbourhood of a vertex.

5. Graph convolution with auto-regressive moving average, ARMA [BGLA21]: This is a convolution method that employs a polynomial function of the feature-vectors in the neighbourhood of a vertex.

In our implementations that we will describe later, in addition to the graph convolution methods mentioned above, we use a graph-pooling step that applies down-sampling to graphs. This operation allows to obtain refined graph representations at each layer. Like in convolutional neural networks, a (graph-)pooling operation follows a (graph-)convolution operation. The primary aim of including a graph pooling operation after each graph convolution is that this operation can reduce the graph representation while ideally preserving important structural information. In the research conducted in this dissertation, we use a popular structural-attention based graph pooling method [LLK19].

To implement the READOUT procedure, we use hierarchical graph-pooling method proposed by [CVJ+18]. Therefore, a GNN variant for us will refer to a GNN constructed with one of the above mentioned graph-convolution operators, the graph-pooling operator and the hierarchical operator. A detailed conceptual and mathematical description of these convolution and pooling operators are provided in Appendix A.

4.2 Inclusion of n-ary relations into GNNs by Enriching Vertex-Labels

GNNs, as we have described them so far, deal with node- and edge-labels in an undirected graph, in which edges are sets of vertex-pairs. That is, the edges represent a symmetric binary relation. However, for many real-world problems—including the ones considered in this chapter—we have access to domain-knowledge which relate more than just pairs

of vertices. For example, if a molecule is represented as a graph (with atoms as vertices, and an edge denoting a bond between a pair of vertices), then a benzene-ring is a relation amongst 6 distinct vertices, with some specific constraints on the vertices and edges. Here, we will consider domain-knowledge to be a set of relations, each of which can be expressed as a hypergraph.

Definition 4.7 (Hypergraphs)**.** *A hypergraph H is the pair (V, E'), where V is a set of vertices and E' is a non-empty subset of 2^V. Each element of E' is called a hyperedge.*

Example 4.4. *A hypergraph of the molecular graph given in Example 4.1 can be*

$$H = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{3,4,5,6\}, \{2,4,5\}, \{1,2,3,4,5,6\}\}).$$

We note that since hyperedges are sets, there is no distinction between permutations of vertices in a hyperedge. So, as defined here, we will take hyperedges as being undirected. Hypergraph labellings can be defined similarly as before, using a pair of functions for vertex- and edge-labels. We will reuse the notation ψ and ϵ for these functions, with annotations to clarify what is meant. The neighboorhood relation σ is left unspecified here (one obvious definition is $\sigma(v_i) = \{v_j : h \in E', \{v_i, v_j\} \subseteq h\}$). In this chapter, we are interested in n-ary relations that can be expressed as hypergraphs.

Definition 4.8 (n-ary Relation as a Labelled Hypergraph)**.** *A n-ary relation R defined over vertices of a graph $G = (V, E)$ is a hypergraph $H = (V, E')$, and every hyperedge $h \in E'$ has n elements from V. We will denote this as $R(G) = H$. Let ψ_G denote a vertex-labelling over G and R/n denote the predicate-symbol for R. With some abuse of notation, the vertex-labelling function for $R(G) = H = (V, E')$ is as follows:*

$$\psi_H(v) = \begin{cases} \psi_G(v) \cup \{R/n\} & \text{if } \exists h \in E' s.t.\ v \in h \\ \emptyset & \text{otherwise} \end{cases}$$

and the hyperedge-labelling function is

$$\epsilon_H(h) \ = \ \{R/n\} \quad (h \in E').$$

That is, the vertex-labelling of a vertex v in the hypergraph H is a set containing the existing vertex-label of v in G augmented by the predicate-symbol R/n vertex-label.

Example 4.5. *Consider a relation for a benzene ring:*

$$\begin{aligned} benzene(a_1, a_2, a_3, a_4, a_5, a_6) \leftarrow & \\ & cycle(a_1, a_2, a_3, a_4, a_5, a_6) \wedge \\ & aromatic(a_1, a_2, a_3, a_4, a_5, a_6). \end{aligned}$$

One possible vertex-labelling is

$$\psi_H(1) = \cdots = \psi_H(6) = \{C, benzene/6\}$$

(here, C denotes "carbon"). A hyperedge-labelling may contain:

$$\epsilon_H(\{1, 2, 3, 4, 5, 6\}) = \{benzene/6\}.$$

The extension to multiple relations, not all of the same arity, is straightforward.

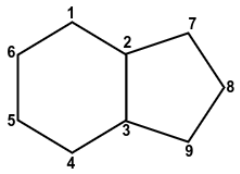
Definition 4.9 (Multiple Relations as a Labelled Hypergraph)**.** *Let $R_1, \ldots, R_k$ be relations defined on vertices of a graph $G = (V, E)$, s.t. $R_i(G) = (V, E_i')$. Then $\bigcup_{i=1}^{k} R_i(G)$ is the hypergraph $H = (V, E')$ where $E' = \bigcup_{i=1}^{k} E_i'$. The corresponding labelling functions are*

$$\psi_H(v) = \bigcup_{i=1}^{k} \psi_{H_i}(v)$$

and

$$\epsilon_H(v) = \bigcup_{i=1}^{k} \epsilon_{H_i}(v).$$

Example 4.6. *Consider the following molecular graph with two relations: benzene/6 and pyrrole/5.*



One possible vertex-labelling for this graph is

$$\begin{aligned} \psi_H(1) = \psi_H(4) = \psi_H(5) = \psi_H(6) &= \{C, benzene/6\} \\ \psi_H(2) = \psi_H(3) &= \{C, benzene/6, pyrrole/5\} \\ \psi_H(7) &= \{N, pyrrole/5\} \\ \psi_H(8) = \psi_H(9) &= \{C, pyrrole/5\} \end{aligned}$$

and a hyperedge-labelling is

$$\epsilon_H(\{1,2,3,4,5,6\}) = \{benzene/6\}$$
$$\epsilon_H(\{2,7,8,9,3\}) = \{pyrrole/5\}.$$

In principle, provided we are able to define a neighbourhood function σ for hypergraphs, the definition of GNNs in Defn. 4.6 does not change. We would however like to use one of the standard GNN implementations described in the previous section, which restricts graphs with 2-vertex edges, and edge-labels to singleton sets. With some loss of information, we extract a suitable graph from a hypergraph.

Definition 4.10 (Vertex-Enriched Graphs)**.** *Let $G = (V, E)$ be a graph, with neighbourhood function σ, vertex-labelling function ψ, and edge-labelling function ϵ. Here, E is a subset of $V \times V$. Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ be a set of relations defined on G, and $\bigcup R_i(G)$ be the hypergraph $H = (V, E')$ with vertex-labelling function ψ' as in Defn. 4.9. Then $G' = (V, E, \sigma, \psi', \epsilon)$ is called a vertex-enriched form of $G = (V, E, \sigma, \psi, \epsilon)$. We denote this by $VE(G, \mathcal{R}) = G'$.*

Example 4.7. *The molecular graph G for Example 4.6 is*

$$G = (\{1,2,3,4,5,6,7,8,9\}, \{(1,2),(2,1),\ldots,(1,6),(6,1),(2,7),(7,2),\ldots,(9,3)(3,9)\}).$$

A vertex-labelling of G is

$$\psi(1) = \cdots = \psi(6) = \psi(8) = \psi(9) = \{C\}$$
$$\psi(7) = \{N\}$$

The vertex-labelling of the vertex-enriched graph G', after the inclusion of the relations in Example 4.6 is:

$$\psi'(1) = \psi'(4) = \psi'(5) = \psi'(6) = \{C, benzene/6\}$$
$$\psi'(2) = \psi'(3) = \{C, benzene/6, pyrrole/5\}$$
$$\psi'(7) = \{N, pyrrole/5\}$$
$$\psi'(8) = \psi'(9) = \{C, pyrrole/5\}.$$

The edge-labelling and neighborhood functions do not change after relation enrichment.

The vertex-enriched graph thus extends the vertex-labelling of a graph G, with the vertex-labels from the hypergraph H obtained from relations $R_1, \ldots, R_k$ defined on G. Procedure 3 provides a set of formal steps to enrich a graph G given a set of relations $\mathcal{R}$.

Procedure 3 requires identification of subgraphs of the original graph. That is, for every relation $R_i \in \mathcal{R}$, the corresponding hyperedge H_i is a subset of vertices $\{v_1, \ldots, v_n\} \in V$, such that $(v_1, \ldots, v_n) \in R_i$. This step requires the identification of all subsets of vertices of the graph constituting hyperedge as above. For a graph (V, E), this can, in the worst case require an examination of $\binom{|V|}{n}$ combinations. Therefore, for arbitrary sized graphs and subgraphs, this is computationally hard. In practice, we will be forced to impose bounds on the size of V_s and on the size of the subgraph.

We note that the process of vertex-enrichment is a simplification of the full relational information available. For example, in the example above, if an atom (represented by a vertex in the molecular graph) is part of more than 1 benzene ring, then its vertex-enrichment will only contain a single entry for *benzene*/6, indicating that it is part of 1 or more benzene rings. We discuss these limitations in detail later.

Procedure 3 Vertex-Enrichment of a graph G, given a set of relations $\mathcal{R}$. The new label of a vertex includes all the relations of which the vertex is part. The procedure takes as input a graph G, a set of relations $\mathcal{R}$, and returns a vertex-enriched graph G'.

1: **procedure** EnrichGraph($G = (V, E, \sigma, \psi, \epsilon)$, $\mathcal{R} = \{R_1, \ldots, R_k\}$)
2: Let $\psi' := \psi$
3: **for all** $R_i \in \mathcal{R}$ **do**
4: Let $R_i \subseteq V^n$
5: $\mathcal{H}_i = \{\{v_1, \ldots, v_n\} : (v_1, \ldots, v_n) \in R_i\}$
6: Let $V_s = \bigcup_{H_j \in \mathcal{H}_i} H_j$
7: **for all** $v_j \in V_s$ **do**
8: $\psi'(v_j) := \psi'(v_j) \cup \{R_i/n\}$
9: **return** $G' = (V, E, \sigma, \psi', \epsilon)$

The vertex-enriched graph obtained here are immediately suitable for the GNN implementations we consider in this chapter: The vertex-labels of a graph are sets and a GNN implementation cannot handle sets. We provide a simple technique to transform these graphs into a form suitable for GNN implementations.

4.2.1 Vertex-Enriched GNNs

For use by a GNN, a vertex-label in a labelled graph needs to be transformed to a fixed-length feature-vector. In our implementations of GNN variants, each vertex-label is transformed to a fixed-length multi-hot Boolean-valued vector associated with the vertex. We define a function $Vectorise$ to do this transformation.

Definition 4.11 (Vectorisation)**.** *Let $\mathcal{G}$ be a set of vertex-enriched graphs, where each graph G' is of the form $(V, E, \sigma, \psi', \epsilon)$. Assume a set of domain relations, denoted by $\mathcal{R}$, available in background knowledge B. Let $\mathcal{R}$ consist of k relations: $R_1, \ldots, R_k$. We define*

a label-vectorisation function $\psi'' : V \times \{1, \ldots, k\} \to \{0, 1\}$ *as*

$$\psi''(v, i) = \begin{cases} 1 & \text{if } R_i \in \psi'(v) \\ 0 & \text{otherwise} \end{cases}$$

Thus, the vectorisation of the label of a vertex $v \in V$ *is* $(\psi''(v, 1), \ldots, \psi''(v, k))$*. Then, the graph* $G'' = (V, E, \sigma, \psi'', \epsilon)$ *is the vectorised form of* G' *and is denoted as* $G'' = Vectorise(G', \mathcal{R})$*.*

Definition 4.12 (Vertex-Enriched GNN)**.** *Let* $G = (V, E, \sigma, \psi, \epsilon)$*, and* $Vectorise$*,* $Relabel$*,* Vec *and* NN *be as before. Then, a Vertex-enriched GNN is*

$$VEGNN(G) = NN(Vec(Relabel(Vectorise(VE(G, \mathcal{R}), \mathcal{R})))).$$

Example 4.8. *Let's consider the example of a molecular graph* G *given given in Example 4.6 and its corresponding vertex-enriched graph* G' *in Example 4.7. For this example, let's assume* $\mathcal{R} = \{C, N, O, benzene/6, furan/5, pyrrole/5\}$*. The vectorised form of* G' *will have vertices associated with the following vectors of length* 6*:*

v	$\psi''(v)^\top$
1	$[1, 0, 0, 1, 0, 0]$
2	$[1, 0, 0, 1, 0, 1]$
3	$[1, 0, 0, 1, 0, 1]$
4	$[1, 0, 0, 1, 0, 0]$
5	$[1, 0, 0, 1, 0, 0]$
6	$[1, 0, 0, 1, 0, 0]$
7	$[0, 1, 0, 0, 0, 1]$
8	$[1, 0, 0, 0, 0, 1]$
9	$[1, 0, 0, 0, 0, 1]$

Notice that the atom types such as carbon, nitrogen, oxygen, denoted by C*,* N*,* O*, respectively, are treated as relations with arity* 0*.*

Procedure 4 and Procedure 5 are two simple procedures to construct and evaluate VEGNNs. These procedures assumes two sub-procedures: TrainGNN, that trains a standard GNN using a set of labelled graph-instances; and Evaluate, that computes the predictive performance of a trained model.

Procedure 4 Procedure to construct a VEGNN model. The procedure takes as inputs: a set of of labelled data-instances $D_{Tr} = \{(g_1, y_1), \ldots, (g_N, y_N)\}$, where g_i is the graph-representation of a relational data-instance and y_i is a class-label associated with g_i; a set of domain-relations $\mathcal{R}$; a set of hyperparameters P; and returns a VEGNN model.

1: **procedure** TRAINVEGNN($D_{Tr}, \mathcal{R}$)
2: $D'_{Tr} = \{(g'_i, y_i) : (g_i, y_i) \in D_{Tr}$ and $g'_i = VE(g_i, \mathcal{R}\}$
3: $D''_{Tr} = \{(g''_i, y_i) : (g'_i, y_i) \in D'_{Tr}$ and $g''_i = Vectorise(g'_i, \mathcal{R})\}$
4: Let $VEGNN =$ TRAINGNN(D''_{Tr}, P)
5: **return** $VEGNN$

Procedure 5 Procedure to test a trained VEGNN model. The procedure takes as inputs: a trained model $VEGNN$; a set of of labelled data-instances $D_{Te} = \{(g_1, y_1), \ldots, (g_M, y_M)\}$; a set of domain-relations $\mathcal{R}$; and returns the predictive performance of $VEGNN$.

1: **procedure** TESTVEGNN($D_{Te}, \mathcal{R}$, P)
2: $D'_{Te} = \{(g'_i, y_i) : (g_i, y_i) \in D_{Te}$ and $g'_i = VE(g_i, \mathcal{R}\}$
3: $D''_{Te} = \{(g''_i, y_i) : (g'_i, y_i) \in D'_{Te}$ and $g''_i = Vectorise(g'_i, \mathcal{R})\}$
4: Let $\hat{\mathbf{y}} = \{\hat{y}_i = VEGNN(g''_i) : (g''_i, \cdot) \in D''_{Te}\}$
5: Let $perf =$ EVALUATE($\mathbf{y}, \hat{\mathbf{y}}$) ▷ where $\mathbf{y} = \{y_i : (g''_i, y_i) \in D''_{te}\}$
6: **retun** $perf$

4.3 Empirical Evaluation

4.3.1 Aims

The aim of this empirical study is to evaluate the performance of VEGNNs that includes domain-knowledge using the vertex-enrichment technique. That is,

- We investigate whether the performance of a VEGNN that includes domain-knowledge using vertex-enrichment is better than the performance of a GNN that does not include domain-knowledge.

4.3.2 Materials

The datasets and background knowledge used here are the same 73 classification problems used in the previous chapter (see section 3.7.2 and section 3.7.2). Here we only describe changes in Materials specific to the experiments in this chapter.

Algorithms and Machines

The data and background knowledge are written in Prolog. A Prolog program is used to extract the set of vertices for which a domain-relation is true. We use YAP compiler for execution of this logic program. All the deep neural network experiments are conducted in a Python environment. The GNN models are implemented by using the PyTorch

Geometric library [FL19], which is a popular geometric deep learning extension for PyTorch [PGM+19] and it provides graph pre-processing routines and makes the definition of graph convolution easier to implement.

For all the experiments, we use a machine with Ubuntu (16.04 LTS) operating system, and hardware configuration such as: 64GB of main memory, 16-core Intel Xeon processor, a NVIDIA P4000 graphics processor with 8GB of video memory.

4.3.3 Method

In all experiments, we refer to GNN variants as $GNN_{1,\ldots,5}$ and the corresponding vertex-enriched versions are $VEGNN_{1,\ldots,5}$. For the methodology outlined below, let D be a set of labelled data-instances represented as graphs: $\{(g_1, y_1), \ldots, (g_N, y_N)\}$ where y_i is a class-label associated with the graph g_i. For constructing the VEGNNs, we assume that we have access to: (a) a set of domain relations $\mathcal{R}$, (b) implementations of the GNN variants, and (c) the procedures TrainVEGNN and TestVEGNN as described in Procedure 4 and Procedure 5, respectively. Our method for investigating the performance of $VEGNN$s is straightforward:

(1) Randomly split D into D_{Tr} (train-set) and D_{Te} (test-set);

(2) Construct a GNN model using D_{Tr} (GNN without the domain-relations in $\mathcal{R}$). Let GNN_i be the resulting GNN model, where $i = 1, \ldots, 5$ refers the GNN variant;

(3) Construct a VEGNN model using D_{Tr} (GNN with the domain-relations in $\mathcal{R}$). Let $VEGNN_i$ be the resulting VEGNN model;

(4) Obtain the predictive performance of the GNN model constructed in Step (2) on D_{Te};

(5) Obtain the predictive performance of the VEGNN model constructed in Step (3) on D_{Te};

(6) Compare the predictive performance of $VEGNN_i$ against that of GNN_i ($i = 1, \ldots, 5$).

The following additional details are relevant:

- We have used a 70:30 train-test split for each of the datasets. 10% of the train-set is used as a validation set for hyperparameter tuning.
- The relations in $\mathcal{R}$ are those described in section 3.7.2.

- The general workflow involved in GNNs is described in section 4.1. A diagram of the components involved in implementing that workflow for constructing a VEGNN is shown in Figure 4.2. As shown in the figure, a GNN in our implementations consists of three graph convolution blocks and three graph pooling blocks. The convolution and pooling blocks interleave each other (that is, C-P-C-P-C-P).

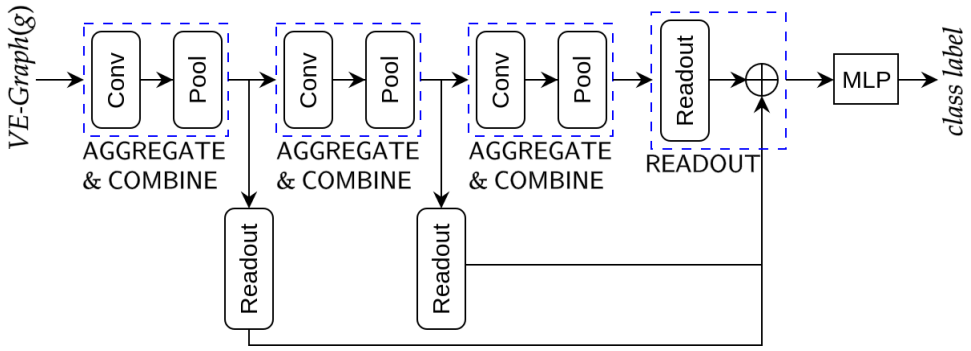


Figure 4.2: Components involved in implementing the workflow in section 4.1 for VEGNN models. The input is the vectorised representation of a vertex-enriched graph, denoted here as $VE\text{-}Graph(g)$ for an graph data-instance g. The blocks 'Conv' and 'Pool' refer to the graph-convolution and graph-pooling operations, respectively. The 'Readout' operation constructs a graph representation by accumulating information from all the vertex in the graph obtained after the pooling operation. The final graph representation is obtained in the READOUT block by an element-wise sum (shown as $\oplus$) of the individual graph-representations obtained after each AGGREGATE-COMBINE block. MLP stands for Multilayer Perceptron.

- The convolution blocks can be of one of the five variants, discussed earlier in subsection 4.1.2.
- The graph pooling block uses self-attention pooling [LLK19] with a pooling ratio of 0.5. We use the graph-convolution formula proposed in [KW17] for calculating the self-attention scores used in the self-attention pooling.
- Due to the large number of experiments (resulting from multiple datasets and multiple GNN variants), the hyperparameters in the convolution blocks are set to the default values within the PyTorch Geometric library [FL19].
- We use a hierarchical pooling architecture that uses the readout mechanism proposed by Cangea *et al.* [CVJ+18]. The readout block aggregates node features to produce a fixed size intermediate representation for the graph. The final fixed-size representation for the graph is obtained by element-wise addition of the three readout representations.
- The representation length ($2m$) is determined by using a validation-based approach. The parameter grid for m is: $\{8, 128\}$, representing a small and a large embedding, respectively.

- The final representation is then fed as input to a 3-layered MLP. We use a dropout layer with a fixed dropout rate of 0.5 after the first layer of MLP.

- The input layer of the MLP contains $2m$ units, followed by two hidden layers with m units and $\lfloor m/2 \rfloor$ units, respectively. The activation function used in the hidden layers is `relu`. The output layer uses `logsoftmax` activation.

- The loss function used is the negative log-likelihood between the target class-labels and the predictions from the model.

- We denote the $VEGNN$ variants as: $VEGNN_{1,\ldots,5}$ based on the type of graph convolution method used.

- We use the Adam optimiser [KB15] for training the VEGNNs ($VEGNN_{1,\ldots,5}$). The learning rate is 0.0005, weight decay parameter is 0.0001, the momentum factors are set to the default values of $(\beta_1, \beta_2) = (0.9, 0.999)$.

- The maximum number of training epochs is 1000. The batch size is 128.

- We use an early-stopping mechanism [Pre98] to avoid overfitting during training. The resulting model is then saved and can be used for evaluation on the independent test-set (D_{Te}). The patience period for early-stopping is fixed at 50.

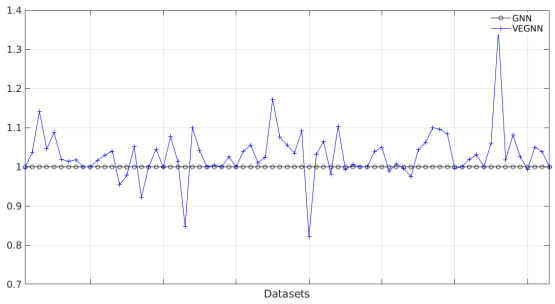
- The predictive performance of a VEGNN model refers to its predictive accuracy on the independent test-set.

- Comparison of performance is done using the Wilcoxon signed-rank test, using the standard implementation within MATLAB (R218b).

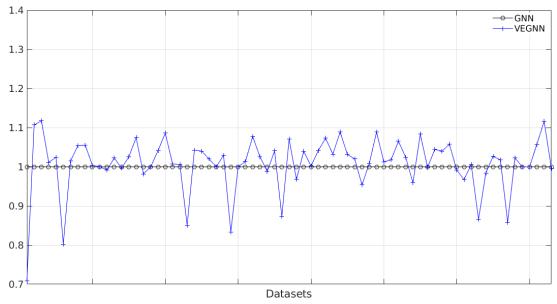
4.3.4 Results

The main results from the experiments are shown qualitatively in Figure 4.3. The principal finding from the tabulations is that inclusion of domain-knowledge into GNNs (that is, the use of vertex-enriched GNNs) results in an improvement in predictive accuracy for all variants of GNN.

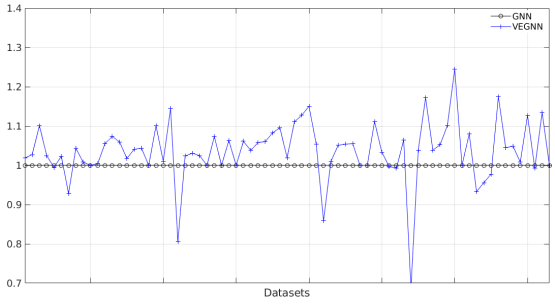
We examine the results in more detail: From Figure 4.3, it is evident that the performance of graph-based networks improves with the inclusion of domain-knowledge. A quantitative tabulation of wins, losses and draws is in Figure 4.4. These results again provide sufficient grounds to answer positively the primary research question addressed in this dissertation, namely: do DNNs (here, represented by GNNs) benefit from the inclusion of domain-knowledge?



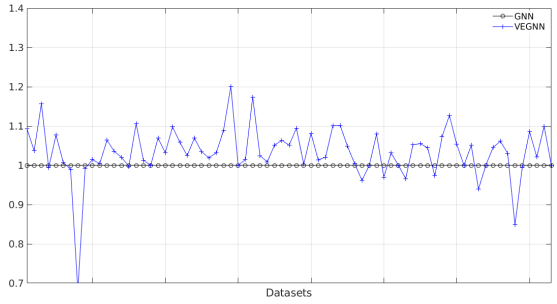
(a) GNN_1 (median gain $\approx 3\%$)



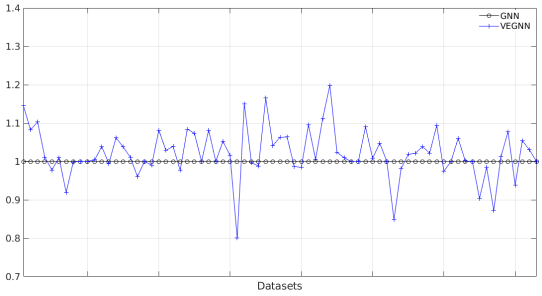
(b) GNN_2 (median gain $\approx 3\%$)



(c) GNN_3 (median gain $\approx 4\%$)



(d) GNN_4 (median gain $\approx 3\%$)



(e) GNN_5 (median gain $\approx 2\%$)

Figure 4.3: Qualitative comparison of predictive performance of VEGNNs against Baseline (that is, GNN variants without access to domain-relations). Performance refers to estimates of predictive accuracy (obtained on a holdout set), and all performances are normalised against that of baseline performance (taken as 1). No significance should be attached to the line joining the data points: this is only for visual clarity.

Some Additional Comparisons

Although not of direct relevance to the primary research conjecture of this dissertation (that is, the inclusion of domain-knowledge can significantly improve the performance of DNNs), it is nevertheless useful to ask how the two approaches we have investigated so far compare against each other. To answer this question, we perform a quantitative comparison between VEGNNs and Deep Relational Machines (DRMs) constructed using propositionalisation of relational features sampled using our proposed hide-and-seek sampling strategy in Chapter 3. We provide a tabulation of this comparison in Figure 4.5. The finding suggests that DRMs can perform at the same or higher level of predictive performance as VEGNNs. At the outset, it may seem easy to conclude that simple machinery like DRMs are better than VEGNNs for the inclusion of domain-knowledge, we

GNN Variant	Accuracy ($VEGNN$ vs. GNN) Higher/Lower/Equal (p-value)
GNN_1	48/14/11 (< 0.001)
GNN_2	48/19/6 (0.005)
GNN_3	53/11/9 (< 0.001)
GNN_4	54/12/7 (< 0.001)
GNN_5	43/19/11 (0.002)

Figure 4.4: Quantitative comparison of predictive performance of $VEGNN$s against GNNs. Here GNN refers to the graph-based neural network without domain-knowledge, and $VEGNN$ refers to the network vertex-enriched with the generic domain-knowledge described in section 3.7.2. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

highlight some key issues that are not apparent from these tabulations: (1) DRMs need to be provided with a sufficient number of relational features (here, 250) to match the same level of performance as a VEGNN; (2) DRMs need to be provided with an expressive set of relational features to reach the same level of performance as a VEGNN; (3) For a DRM, there is significant computational effort is required to draw these 250 features using sampling. As discussed in Chapter 3, the sampling procedure incurs a huge computational cost to select a set of 250 features where selecting just one relational feature requires: sampling a lot more than one feature, evaluating them for their utilities, and discarding the features with bad utilities. Whereas VEGNNs do not involve any such sampling step and therefore the computational cost remains relatively minimal.

We further compare VEGNNs against another propositionalisation based technique for inclusion of domain-knowledge, called BCP [FZG14] that we investigated in Chapter 3. A comparison of VEGNNs against MLPs constructed with BCP features are provided in Figure 4.6. The results here reaffirm that though propositionalisation based techniques are simple, and they require significant computational overhead to perform well: In this case, the number of input BCP features for an MLP ranges from 18000 to 52000 (Refer section 3.7 for more details). This kind of number leads to more complex MLPs that require huge training effort.

4.3.5 Limitations of VEGNNs

So far we have studied how vertex-enrichment allows relations in the background knowledge to be incorporated into a graph neural network. Although this approach is effective in terms of improving the predictive performance of GNNs, it comes with a very basic limitation. Let us examine the following molecule (a graph data-instance) with 2 fused benzene rings and its corresponding vertex-enriched molecular graph in Figure 4.7. In

GNN Variant	Accuracy ($VEGNN$ vs. DRM) Higher/Lower/Equal (p-value)		
	$\lvert\mathcal{R}'\rvert = 50$	$\lvert\mathcal{R}'\rvert = 100$	$\lvert\mathcal{R}'\rvert = 250$
GNN_1	59/13/1 (< 0.001)	50/22/1 (< 0.001)	21/52/0 (< 0.001)
GNN_2	49/23/1 (< 0.01)	39/33/1 (0.81)	19/54/0 (< 0.001)
GNN_3	54/18/1 (< 0.001)	44/28/1 (0.05)	14/59/0 (< 0.001)
GNN_4	59/13/1 (< 0.001)	52/20/1 (< 0.001)	23/50/0 (< 0.001)
GNN_5	53/19/1 (< 0.001)	42/30/1 (0.06)	17/56/0 (< 0.001)

Figure 4.5: Quantitative comparison of predictive performance of VEGNNs against DRMs. Here $VEGNN$ denotes the vertex-enriched GNN with $\mathcal{R}$, and DRM denotes the Deep Relational Machine constructed using propositionalisation of relational features. The relational features for a DRM are sampled using the hide-and-seek sampling strategy proposed in Chapter 3. The set of the hide-and-seek features is denoted by $\mathcal{R}'$. The comparative performance of VEGNNs against DRMs starts worsening after $|\mathcal{R}'| = 500$, which are not shown here. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

GNN Variant	Accuracy ($VEGNN$ vs. BCP+MLP) Higher/Lower/Equal (p-value)
GNN_1	51/21/1 (< 0.001)
GNN_2	46/26/1 (0.08)
GNN_3	48/24/1 (0.003)
GNN_4	54/18/1 (< 0.001)
GNN_5	47/25/1 (0.005)

Figure 4.6: Quantitative comparison of predictive performance of $VEGNN$s against that of MLPs constructed using BCP features [FZG14]. The tabulations are the number of datasets on which $VEGNN$ has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

the graph it is clear that vertices v_4 and v_5 are members of two different benzene rings, however the vertex-enrichment is not able to capture this fact automatically. That is, given the vertex-labels alone we may not be able to deduce that these two vertices are members of two different benzene rings.

The limitation of vertex-enrichment stated above is more apparent in the following molecular graph (highlighting only the vertex-labels for two vertices) in Figure 4.8. The vertex v_4 is a member of three different benzene rings and as a result two different fused rings, which has not been captured by the proposed vertex-enrichment technique. We could, therefore, say that vertex-enrichment is a simplification method for the inclusion of domain-knowledge into GNNs. Here we prefer the word "simplification" instead of

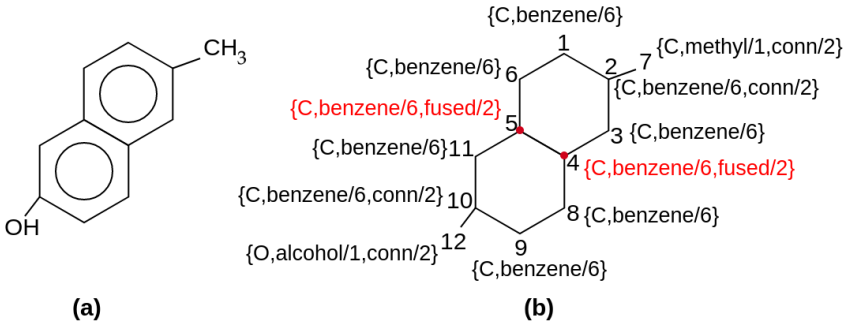


Figure 4.7: Figure showing (a) a molecule with 2 fused benzene rings, (b) its corresponding molecular graph with vertices enriched with domain-relations.

"approximation" for an obvious reason that multiple occurrences of any n-ary domain relation for a vertex in a graph is simplified and shown as a single occurrence.

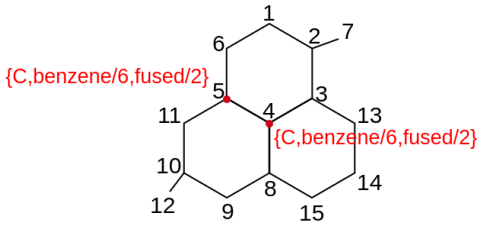


Figure 4.8: Figure highlighting a limitation of the vertex-enrichment technique for a molecular graph.

4.4 Summary

In this chapter, we proposed a simplified technique, called vertex-enrichment, for inclusion of domain-knowledge into graph-based neural networks (or GNNs). The resulting GNNs are called Vertex-Enriched GNNs or VEGNNs. We performed a large-scale evaluation of VEGNNs and compared their predictive performance against GNNs that do not include domain-knowledge. The primary results here clearly show the benefit of having mechanisms to incorporate domain-knowledge into GNNs. To the best of our knowledge, the experiments in this chapter constitute some of the most extensive applications of GNNs to large-scale real-world scientific data arising in the domain of drug-discovery. Further, we studied that vertex-enrichment, being a simplification technique to incorporate all the domain-relations available in the background knowledge, may limit the expressiveness of a VEGNNs.

Chapter 5

Complete Inclusion of Relational Information using Inverse Entailment*

At the end of the previous chapter, we showed that using vertex-enrichment as a technique for inclusion of domain-knowledge results in a loss of information. For example, a vertex in a vertex-enriched graph does not encode information that it is a part of multiple relations with the same predicate symbol and arity. We therefore say vertex-enrichment is a *simplified* approach to inclusion of relational information. In this chapter, we attempt to rectify this deficiency by including all of the relational information available into a graph-based neural network or GNN. The technique we propose is based on inverse entailment developed in the Inductive Logic Programming (ILP) literature. Here, for any given relational data instance, the inverse-entailment technique allows us to identify all the domain-relations entailed by the domain-knowledge via the construction of a *bottom-clause*. We propose a method to represent a bottom-clause using a *bottom-graph* that can be converted into a form suitable for GNN implementations. Overall, this transformation from a bottom-clause to a bottom-graph allows a principled way for complete inclusion of domain-knowledge into GNNs: we use the term "BotGNNs" for this form of graph neural networks. We evaluate the use of domain-knowledge in this manner using the relational datasets and background knowledge studied in the previous chapters. The main contributions of this chapter are as follows: (1) This chapter proposes a systematic technique using the method of inverse entailment in ILP, in particular, Mode-Directed Inverse-Entailment (MDIE) for complete inclusion of symbolic domain-knowledge into GNNs. The proposal in this chapter is a new technique for neuro-symbolic learning, called Bottom-Graph Neural Networks or BotGNNs; (2) This chapter provides a large-

*The content of this chapter is based on the following:
T. Dash, A. Srinivasan, A. Baskar, "Inclusion of domain-knowledge into GNNs using mode-directed inverse entailment", *Machine Learning*, 2021; https://doi.org/10.1007/s10994-021-06090-8.

scale evaluation of BotGNNs, constructed using relational data and domain-knowledge.

5.1 Mode-Directed Inverse Entailment

Mode-directed Inverse Entailment (MDIE) was introduced by Muggleton in [Mug95], as a technique for constraining the search for explanations for data in Inductive Logic Programming (ILP). For this chapter, it is sufficient to focus on variants of ILP that conforms to the following input-output requirements:[1]

Given: (i) B, a set of clauses (constituting background- or domain-) knowledge; (ii) a set of clauses $E^+ = \{p_1, p_2, \ldots, p_N\}$ $(N > 0)$, denoting a conjunction of "positive examples"; and (iii) a set of clauses $E^- = \{\overline{n_1}, \overline{n_2}, \ldots, \overline{n_M}\}$ $(M \geq 0)$, denoting a conjunction of "negative examples", such that

Prior Necessity. $B \not\models E^+$

Find: A finite set of clauses (usually in a subset of first-order logic), $H = \{D_1, D_2, \ldots, D_k\}$ such that

Weak Posterior Sufficiency. For every $D_j \in H$, $B \cup \{D_j\} \models p_1 \vee p_2 \vee \cdots \vee p_N$

Strong Posterior Sufficiency. $B \cup H \models E^+$

Posterior Satisfiability. $B \cup H \cup E^- \not\models \square$

Here $\models$ denotes logical consequence and $\square$ denotes a contradiction. MDIE implementations attempt to find the most-probable H, given B and the data E^+, E^-.[2]

The key concept used in [Mug95] is to constrain the identification of D_j using a *most-specific clause*. The following is adapted from [Mug95].

Remark 5.1 (Most-Specific Clause)**.** *Given background knowledge B and a data-instance e (it does not matter at this point if $e \in E^+$ or $e \in E^-$), any clause D s.t. $B \cup \{D\} \models e$ will satisfy $D \models \overline{B \cup \overline{e}}$. This follows directly from the Deduction Theorem. Let $A = a_1 \wedge a_2 \cdots \wedge a_n$ be the conjunction of ground literals[3] true in all models of $B \cup \overline{e}$. Hence $B \cup \overline{e} \models a_1 \wedge a_2 \wedge \cdots \wedge a_n$. That is, $\overline{a_1 \wedge a_2 \wedge \cdots \wedge a_n} \models \overline{B \cup \overline{e}}$. Let $\bot_B(e)$ denote $\overline{a_1 \wedge a_2 \wedge \cdots \wedge a_n}$. That is, $\bot_B(e)$ is the clause $\neg a_1 \vee \neg a_2 \vee \cdots \vee \neg a_n$. Furthermore, since the a_is' are ground, $\bot_B(e)$ is a ground clause.*

[1] In the following, clauses will usually be in some subset of first-order logic (usually Horn- or definite-clauses). When we refer to a set of clauses, we will usually assume it to be finite. A set of clauses $\mathcal{C} = \{D_1, D_2, \ldots, D_k\}$ will often be used interchangebly with the logical formula $\mathcal{C} = D_1 \wedge D_2 \wedge \cdots \wedge D_k$.

[2] Usually, the entailment relation $\models$ is used to identify logical consequences of some set of logical sentences P. That is, what are the e's s.t. $P \models e$? Here, we are given the e's and B, and are asking what is an H s.t. $P = B \cup H$. In this sense, H is said to be the result of inverting entailment (IE).

[3] In theory, the number of ground literals can be infinite. Practical constraints that restrict this number to a finite size are described shortly.

For any clause D, if $D \models \bot_B(e)$ then $D \models \overline{B \cup \overline{e}}$. Thus any such D satisfies the Weak Posterior Sufficiency condition stated earlier. $\bot_B(e)$ is called the most-specific clause (or "bottom clause") for e, given B.

Thus, bottom clause construction for a data-instance e provides a mechanism for inclusion of all the ground logical consequences given the domain-knowledge B and the instance e.

Example 5.1. *In the following, capitalised letters like X, Y denote variables. Let*

B:

$$parent(X,Y) \leftarrow father(X,Y)$$
$$parent(X,Y) \leftarrow mother(X,Y)$$
$$mother(jane, alice) \leftarrow$$

e:

$$gparent(henry, john) \leftarrow father(henry, jane), mother(jane, john)$$

Here "$\leftarrow$" should be read as "if" and the comma (",") as "and". So, the definition for gparent is to be read as: "henry is a grandparent of john if henry is the father of jane and jane is the mother of john".

The conjunction A of ground literals, true in all models of $B \cup \overline{e}$, is:

$$\neg gparent(henry, john) \wedge father(henry, jane) \wedge mother(jane, john) \wedge mother(jane, alice) \wedge parent(henry, jane) \wedge parent(jane, alice) \wedge parent(jane, john)$$

$\bot_B(e) = \overline{A}$:

$$gparent(henry, john) \leftarrow father(henry, jane),\ mother(jane, john),\ mother(jane, alice),\ parent(henry, jane),\ parent(jane, john),\ parent(jane, alice)$$

The above clause is logically equivalent to the disjunct:

$$gparent(henry, john) \vee \neg father(henry, jane) \vee \cdots \vee \neg parent(jane, alice).$$

We will also write clause like this as the set:

$$\{gparent(henry, john), \neg father(henry, jane), \ldots, \neg parent(jane, alice)\}.$$

$\bot_B(e)$ thus "extends" the example e to include relations in the background knowledge provided: our interest is in the inclusion of the parent/2 relation. The literals in the correct definition of gparent is a "generalised" form of subset of the literals in $\bot_B(e)$. Of course, to find the subset and its generalised form efficiently is a different matter, and is the primary concern of ILP systems used to implement MDIE.

It is common to call the non-negated literal in the disjunct ($gparent(henry, john)$) as the "head" literal, and the negated literals in the disjunct as the "body" literals. In this chapter, we will restrict ourselves to $\bot_B(e)$s' that are definite-clauses (clauses with exactly one head literal). This is for practical reasons, and not a requirement of the MDIE formulation of most-specific clauses.

Construction of $\bot_B(e)$ is called a *saturation* step, reflecting the extension of the example by all potentially relevant facts that are derivable using B and the example e. The domain-knowledge can encode significantly more information than simple binary relations (like *parent* above).

Example 5.2. *Suppose data consist of the atom-and-bond structure of molecules that are known to be toxic. Each toxic molecule can be represented by a clausal formula. For example, a toxic molecule m_1 could be represented by the logical formula (here a_1, a_2 are atoms, c denotes carbon, ar denotes aromatic, and so on):*

$$
\begin{aligned}
&toxic(m_1) \leftarrow \\
&\quad atom(m_1, a_1, c), \\
&\quad atom(m_1, a_2, c), \\
&\quad \vdots \\
&\quad bond(m_1, a_1, a_2, ar), \\
&\quad bond(m_1, a_2, a_3, ar), \\
&\quad \vdots
\end{aligned}
$$

The above clause can be read as: molecule m_1 is toxic, if it contains atom a_1 of type carbon, atom a_2 of type carbon, there is an aromatic bond between a_1 and a_2, and so on. We will see later that this is a definite-clause encoding of a graph-based representation of the molecule m_1.

Given background knowledge definitions (for example, of rings and functional groups), $\bot_B(e)$ would extend the logical definition of e with relevant parts of the background knowledge:

$$
\begin{aligned}
&toxic(m_1) \leftarrow \\
&\quad atom(m_1, a_1, c), \\
&\quad atom(m_1, a_2, c), \\
&\quad \vdots \\
&\quad bond(m_1, a_1, a_2, ar), \\
&\quad bond(m_1, a_2, a_3, ar), \\
&\quad \vdots \\
&\quad benzene(m_1, [a_1, a_2, a_3, a_4, a_5, a_6]), \\
&\quad benzene(m_1, [a_3, a_4, a_8, a_9, a_{10}, a_{11}]), \\
&\quad \vdots \\
&\quad fused(m_1, [a_1, a_2, a_3, a_4, a_5, a_6], [a_3, a_4, a_8, a_9, a_{10}, a_{11}]),
\end{aligned}
$$

$$\vdots$$
$$methyl(m_1, [\ldots]),$$
$$\vdots$$

As seen from this example, the size of $\bot_B(\cdot)$ can be large. More problematically, for complex domain-knowledge, $\bot_B(e)$ may not even be finite. To address this, MDIE introduces the notion of a *depth-bounded* bottom-clause, using *mode* declarations.

5.1.1 Modes

Practical ILP systems like Progol [Mug95] use a *depth-bounded* bottom clause constructed within a mode-language. We first illustrate a simple example of a mode-language specification.

Example 5.3. *A "mode declaration" for an n-arity predicate P (often written as P/n) is one of the following kinds: (a) $modeh(P(a_1, a_2, \ldots, a_n))$; or (b) $modeb(P(a_1, a_2, \ldots, a_n))$. A set of mode-declarations for the predicates in the gparent example is*

$$\mathrm{M} = \{modeh(gparent(+person, -person)), modeb(father(+person, -person)),$$
$$modeb(mother(+person, -person)), modeb(parent(+person, -person))\}.$$

The modeh specifies details about literals that can appear in the head of a clause in the mode-language and the modeb's specify details about literals that can appear in the body of a clause. A "mode declaration" refers to either a modeh or modeb statement. Based on the mode-language specified in [Mug95], each argument a_i in the mode declarations above is one of: (1) $+person$, denoting that the argument in that literal is an 'input' variable of type person.[4] *That is, the variable must have appeared either as a $-person$ variable in a literal that appears earlier in the body of the clause or as a $+person$ variable in the head of the clause; (2) $-person$, denoting that the variable in the literal is an 'output' variable of type person. If an output variable appears in the head of a clause, it must appear as an output variable of some literal in the body. There are no special constraint on output variables in body-literals. That is, they can either be a new variable, or any variable (of the same type) that has appeared earlier in the clause. Later we will see how mode-declarations allow the appearance of ground terms.*

Example 5.4. *Continuing Example 5.3, in the following X, Y, Z are variables of type person. These clauses are all within the mode language specified in [Mug95]:*

(a) $gparent(X, Y) \leftarrow parent(X, Y)$;

[4]Informally, "a variable of type γ" will mean that ground substitutions for the variable are from some set γ. Here, γ is the set $person = \{henry, jane, alice, john, \ldots\}$: that is, $person$ is a unary-relation.

(b) $gparent(X, Y) \leftarrow parent(X, X)$;

(c) $gparent(X, Y) \leftarrow mother(X, Y)$; *and*

(d) $gparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$.

But the following clauses are all not within the mode language in [Mug95]:

(e) $gparent(X, Y) \leftarrow parent(Y, Z)$ *(*Y *does not appear before);*

(f) $gparent(X, Y) \leftarrow parent(X, Y), parent(Z, Y)$ *(*Z *does not appear before);*

(g) $gparent(henry, Y) \leftarrow parent(henry, Z), parent(Z, Y)$ *(+ arguments have to be variables, not ground terms); and*

(h) $gparent(X, Y) \leftarrow parent(Z, jane), parent(Z, Y)$ *(− arguments have to be variables, not ground terms).*

We refer the reader to [Mug95] for more details on the use of modes. Here we confine ourselves to the details necessary for the material in this chapter. We first reproduce the notion of a place-number of a term in a literal following [Plo72].

Definition 5.1 (Term Place-Numbering)**.** *Let* $\pi = \langle i_1, \ldots, i_k \rangle$ *be a sequence of natural numbers. We say that a term* τ *is in place-number* π *of a literal* λ *iff: (1)* $\pi \neq \langle\rangle$*; and (2)* τ *is the term at place-number* $\langle i_2, \ldots, i_k \rangle$ *in the term at the* i_1^{th} *argument of* λ*.* τ *is at a place-number* π *in term* τ'*: (1) if* $\pi = \langle\rangle$ *then* $\tau = \tau'$*; and (2) if* $\pi = \langle i_1, \ldots, i_k \rangle$ *then* τ' *is a term of the form* $f(t_1, \ldots, t_m)$*,* $i_1 \leq m$ *and* τ *is in place-number* $\langle i_2, \ldots, i_k \rangle$ *in* t_{i_1}*.*

Example 5.5. *Let us look at the following examples:*
(a) *In the literal* $\lambda = gparent(henry, john)$*, the term henry occurs in the first argument of* λ *and john occurs in the second argument of* λ*. The place-numbering of henry in* λ *is* $\langle 1 \rangle$ *and of john in* λ *is* $\langle 2 \rangle$*.*
(b) *As a more complex example, let* $\lambda = mem(a, [a, b, c])$ *denote the statement that a is a member of the list* $[a, b, c]$*. The second argument of* λ *is short-hand for the term* $list(a, list(b, list(c, nil)))$ *(usually, the function* $list/2$ *is represented as '.'/2 in the logic-programming literature). Then the term a is a term that occurs in two place-numbers in* λ*:* $\langle 1 \rangle$*, and* $\langle 2, 1 \rangle$*. The term b occurs at place-number* $\langle 2, 2, 1 \rangle$ *in* λ*; the term c occurs at place-number* $\langle 2, 2, 2, 1 \rangle$ *in* λ*; and the term nil occurs at place-number* $\langle 2, 2, 2, 2 \rangle$ *in* λ*.*

We first present the syntactic aspects constituting a mode-language. The meaning of these elements is deferred to the next section.

Definition 5.2 (Mode-Declaration)**.** *The definition is as follows:*

(a) Let Γ be a set of type names. A mode-term is defined recursively as one of: (i) $+\gamma$, $-\gamma$ or $\#\gamma$ for some $\gamma \in \Gamma$; or (ii) $\phi(mt'_1, mt'_2, \ldots, mt'_j)$, where ϕ is a function symbol of arity j, and the $mt'_k s$ are mode-terms. We will call mode-terms of type (i) simple mode-terms and mode-declarations of type (ii) structured mode-terms.[5]

(b) A mode-declaration μ is of the form modeh(λ') or modeb(λ'). Here λ' is a ground-literal of the form $p(mt_1, mt_2, \ldots, mt_n)$ where p is a predicate name with arity n, and the mt_i are mode-terms. We will say μ is a modeh-declaration (resp. modeb-declaration) for the predicate-symbol p/n.[6] *We will also use $ModeLit(\mu)$ to denote λ'.*

(c) μ is said to be a mode-declaration for a literal λ iff λ and $ModeLit(\mu)$ have the same predicate symbol and arity.

(d) Let τ be the term at place-number π in μ, We define

$$ModeType(\mu, \pi) = \begin{cases} (+, \gamma) & \text{if } \tau = +\gamma \\ (-, \gamma) & \text{if } \tau = -\gamma \\ (\#, \gamma) & \text{if } \tau = \#\gamma \\ unknown & \text{otherwise.} \end{cases}$$

(e) If μ is a mode-declaration for literal λ, $ModeType(\mu, \pi) = (+, \gamma)$ for some place-number π, τ is the term at place π in λ, then we will say τ is an input-term of type γ in λ given μ (or simply τ is an input-term of type γ). Similarly we define output-terms and constant-terms.

5.1.2 Depth-Limited Bottom Clauses

Returning now to the most-specific clause $\bot_B(e)$ for a data-instance e, given background knowledge B, it is sufficient for our purposes to understand that the input-output specifications in a set of mode-declarations result in a natural notion of the depth at which any term first appears in $\bot_B(e)$ (terms that appear in the head of the clause are at depth 0, terms that appear in literals whose input terms depend only on terms in the head are at depth 1, and so on. A formal definition follows below.) By fixing an upper-bound d on this depth, we can restrict ourselves to a finite-subset of $\bot_B(e)$.[7] This is called the *depth-limited* bottom clause. Given a set of mode-declarations M, we denote this depth-limited

[5] For all experiments in this chapter, modes consist only of simple mode-terms.

[6] In general there can be several *modeh* or *modeb*-declarations for a predicate-symbol p/n. If there is exactly one mode-declaration for a predicate symbol p/n, we will say the mode declaration for p/n is determinate.

[7] In fact, additional restrictions are also needed on the number of times a relation can occur at any depth. In implementations like [Mug95], this is usually provided as part of the mode declaration.

clause by $\bot_{B,\mathrm{M},d}(e)$ (or simply $\bot_d(e)$), where d is a (pre-specified) depth-limit. We will refer to the corresponding mode-language as a depth-limited mode-language and denote it by $\mathcal{L}_{\mathrm{M},d}$. We first illustrate this with an example before defining depth formally.

Example 5.6. *Using the modes* M *in Example 5.3, we obtain the following most-specific clauses for the gparent example (Example 5.1):*

$\bot_{B,\mathrm{M},1}(e)$:

$$gparent(henry, john) \leftarrow father(henry, jane), parent(henry, jane)$$

$\bot_{B,\mathrm{M},2}(e)$:

$$gparent(henry, john) \leftarrow father(henry, jane), mother(jane, john), mother(jane, alice), parent(henry, jane), parent(jane, john), parent(jane, alice)$$

We now formally define type-definitions and depth for ground-terms.

Definition 5.3 (Type Definitions)**.** *Let* Γ *be a set of types and* T *be a set of ground-terms. For* $\gamma \in \Gamma$ *we define a set of ground-terms* $T_\gamma = \{\tau_1, \tau_2, \ldots\}$*, where* $\tau_i \in \mathrm{T}$*. We will say a ground-term* τ_i *is of type* γ *if* $\tau_i \in T_\gamma$*, and denote by* T_Γ *the set* $\{T_\gamma : \gamma \in \Gamma\}$*. T_Γ will be called a set of type-definitions.*

Definition 5.4 (Depth of a term)**.** *Let* M *be a set of modes. Let* C *be a ground clause. Let* λ_i *be a literal in* C *and let* τ *be an input- or output-term of type* γ *in* λ_i *given some* $\mu \in \mathrm{M}$*. Let* Y_τ *be the set of all other terms in body literals of* C *that contain* τ *as an output-term of type* γ*. Then,*

$$depth(\tau) = \begin{cases} 0 & \text{if } \tau \text{ is an input-term of type } \gamma \\ & \text{in a head literal of } C \\ \min_{\tau' \in Y_\tau} depth(\tau') + 1 & \text{otherwise.} \end{cases}$$

Example 5.7. *In the previous example for* $C = \bot_{B,\mathrm{M},2}(e)$*,* $depth(henry) = 0$*,* $depth(jane) = depth(henry) + 1 = 1$*,* $depth(john) = depth(jane) + 1 = 2$*, and* $depth(alice) = depth(jane) + 1 = 2$*.*

A set of mode-declarations M (see Defn. 5.2), a set of type-definitions T_Γ, and a depth-limit d together define a set of acceptable ground clauses $\mathcal{L}_{T_\Gamma,\mathrm{M},d}$. Informally, $\mathcal{L}_{T_\Gamma,\mathrm{M},d}$ consists of ground clauses in which: (a) all terms are correctly typed; (b) all input terms in a body literal have appeared as output terms in previous body literals or as input terms in any head literal; and (c) all output terms in any head literal appear as output terms in some body literals. In this chapter, we will mainly be interested in definite-clauses (that is, $m = 1$ in the definition that follows).

Definition 5.5 ($\lambda\mu$-Sequence)**.** *Assume a set of type-definitions T_Γ, modes* M*, and a depth-limit d. Let $C = \{l_1, \ldots, l_m, \neg l_{m+1}, \ldots, \neg l_k\}$ be a clause with k ground literals. Then $\langle(\lambda_1, \mu_1),(\lambda_2, \mu_2), \ldots, (\lambda_k, \mu_k)\rangle$ is said to be a $\lambda\mu$-sequence for C iff it satisfies the following constraints:*

(a) (i) The λ's are all distinct and (ii) For $j = 1 \ldots k$, μ_j is a mode-declaration for λ_j; (iii) For $j = 1 \ldots m$, $\lambda_j = l_j$ and $\mu_j = modeh(\cdot)$; (iv) For $j = (m+1) \ldots k$, $\lambda_j = l_i$ where $\neg l_i \in C$, and $\mu_j = modeb(\cdot)$.

(b) If τ is an input-term of type γ in λ_j given μ_j, then

(i) $\tau \in T_\gamma$; and

(ii) if $j > m$:

- *There is an input-term τ of type γ in one of $\lambda_1, \ldots, \lambda_m$ given $\mu_1, \ldots, \mu_m$; or*
- *There is an output-term τ of type γ in λ_i $(m < i < j)$ given μ_i.*

(c) If τ is an output-term of type γ in λ_j given μ_j, then

(i) $\tau \in T_\gamma$; and

(ii) if $j \leq m$:

- *τ is an output-term of type γ for some λ_i $(m < i \leq k)$ given μ_i.*

(d) If τ is a constant-term of type γ in λ_j given μ_j, then $\tau \in T_\gamma$.

(e) There is no term τ at any place π in any λ_j s.t. the $depth(\tau) > d$.

Definition 5.6 (Mode-Language)**.** *Assume a set of type-definitions T_Γ, modes* M*, and a depth-limit d. The mode-language $\mathcal{L}_{T_\Gamma,\mathrm{M},d}$ for T_Γ, M, d is $\{C$: either $C = \emptyset$ or there exists a $\lambda\mu$-sequence for $C\}$.*

Example 5.8. *Let $\mathrm{M} = \{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6\}$ where the mode declarations μ_is are as follows: $\mu_1 = modeh(p(+int))$, $\mu_2 = modeh(p(+real))$, $\mu_3 = modeb(q(+int))$, $\mu_4 = modeb(q(+real))$, $\mu_5 = modeb(r(+int))$, $\mu_6 = modeb(r(+real))$. Let the depth-limit $d = 1$. Let C be a ground definite-clause $p(1) \leftarrow q(1), r(1)$. That is, $C = \{p(1), \neg q(1), \neg r(1)\}$. Let: $\lambda_1 = p(1), \lambda_2 = q(1), \lambda_3 = r(1)$. Then, C is in $\mathcal{L}_{\mathrm{M},d}$. The $\lambda\mu$-sequences for C are $\langle(\lambda_1, \mu_1), (\lambda_2, \mu_3), (\lambda_3, \mu_5)\rangle$; $\langle(\lambda_1, \mu_1), (\lambda_3, \mu_5), (\lambda_2, \mu_3)\rangle$; $\langle(\lambda_1, \mu_2), (\lambda_2, \mu_4), (\lambda_3, \mu_6)\rangle$; $\langle(\lambda_1, \mu_2), (\lambda_3, \mu_6), (\lambda_2, \mu_4)\rangle$.*

We note that Defn. 5.6 does not allow the following two sequences to to be $\lambda\mu$-sequences: $\langle(\lambda_1, \mu_1), (\lambda_2, \mu_4), (\lambda_3, \mu_5)\rangle$ and $\langle(\lambda_1, \mu_2), (\lambda_2, \mu_4), (\lambda_3, \mu_5)\rangle$, because a 1 of type int is treated as being different to a 1 of type real.

We note that although the meanings of $+$, $-$ and $\#$ are the same here as in [Mug95], clauses in $\mathcal{L}_{T_\Gamma,M,d}$ here are restricted to being ground (in [Mug95], clauses are required to have variables in $+$ and $-$ places of literals).

5.2 BotGNNs

In this section, we describe a method to translate the depth-limited most-specific clauses of the previous section ($\bot_{B,\mathrm{M},d}(\cdot)$'s) into a form that can be used by standard variants of GNNs. We illustrate the procedure first with an example.

Example 5.9. *Consider $\bot_{B,\mathrm{M},2}(e)$ in Example 5.6. The tabulation below shows the literals in $\bot_{B,\mathrm{M},2}(e)$ and matching modes.*

S.No.	Literal (λ)	Mode (μ)
1	$gparent(henry, john)$	$modeh(gparent(+person, -person))$
2	$father(henry, jane)$	$modeb(father(+person, -person))$
3	$mother(jane, john)$	$modeb(mother(+person, -person))$
4	$mother(jane, alice)$	$modeb(mother(+person, -person))$
5	$parent(henry, jane)$	$modeb(parent(+person, -person))$
6	$parent(jane, john)$	$modeb(parent(+person, -person))$
7	$parent(jane, alice)$	$modeb(parent(+person, -person))$

The table below shows the ground-terms (τs) in literals appearing in $\bot_{B,\mathrm{M},2}(e)$ and their types (γs), obtained from the corresponding term-place number in the matching mode.

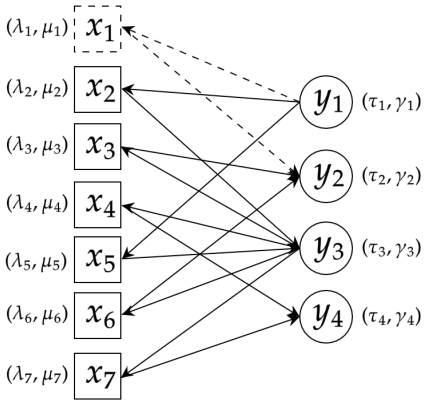
S.No.	Term (τ)	Type (γ)
1	$henry$	$person$
2	$john$	$person$
3	$jane$	$person$
4	$alice$	$person$

The information in these tables can be represented as a directed bipartite graph as shown in Figure 5.1. The square-shaped vertices represent (λ, μ) pairs in the first table, and the round-shaped vertices represent (τ, γ) pairs in the second table. Arcs from a (λ, μ) (square-) vertex to a (τ, γ) (round-) vertex indicates term τ is designated by mode μ as an output or constant term ($-$ or $\#$) of type γ in literal λ. Conversely, an arc from an (τ, γ) vertex to an (λ, μ) vertex indicates that term τ is designated by mode μ as an input term ($+$) of type γ in literal λ.

$\perp_{B,\mathrm{M},2}(e)$:

gparent(*henry*, *john*) ←
father(*henry*, *jane*),
mother(*jane*, *john*),
mother(*jane*, *alice*),
parent(*henry*, *jane*),
parent(*jane*, *john*),
parent(*jane*, *alice*)

(a)



(b)

Figure 5.1: For the *gparent* example: (a) depth-limited bottom-clause $\perp_{B,\mathrm{M},2}(e)$; and (b) the corresponding clause-graph where the vertex-labels (λ, μ)s and (τ, γ)s are as provided in the preceding tables. The "dashed" square-box and the "dashed" arrow are shown to indicate the vertex specifying the head of the clause. The subscripts used in the labels correspond to the S.No. in the tables, for example, (λ_3, μ_3) refers to the third-row in the first table in this example; and, similarly, (τ_4, γ_4) refers to the fourth row in the second table.

The structure in Figure 5.1 is called a bottom-graph in this chapter. BotGNNs are GNN models constructed from graphs based on such clause-graphs. We first clarify some details needed for the construction of clause-graphs.

5.2.1 Notations and Assumptions

Sets. We use the following notations:

(a) Set $\mathcal{E}$ to define a set of relational data-instances[8];

(b) Sets P, F, K to denote predicate-symbols, function-symbols, and constant-symbols, respectively;

(c) Λ to denote the set of all positive ground-literals that can be constructed using P, F, K; and T to denote the set of all ground-terms that can be constructed using F, K;[9]

(d) Λ_C to denote the set of all literals in a clause C;

(e) B to denote the set of predicate-definitions constituting background knowledge;

(f) M to denote the set of modes for the predicate-symbols in P;

[8]In this chapter, this set consists of definite clauses.
[9]A *term* is defined recursively as a constant from K, a variable, or a function symbol from F applied to term. A ground term is a term without any variables.

(g) Let Γ' to denote the set of type-names used by modes in M. In addition, we assume a special type-name $\mathbb{R}$ to denote a numeric type. We denote by Γ the set $\Gamma' \cup \{\#t : t \in \Gamma' \text{ s.t. } \#t \text{ occurs in some mode } \mu \in \mathrm{M}\}$;

(h) LM to denote the set $\{(\lambda, \mu) : \lambda \in \Lambda, \mu \in \mathrm{M}, \mu \text{ is a mode-declaration for } \lambda\}$; and ET to denote the set $\{(\tau, \gamma) : \tau \in \mathrm{T}, \gamma \in \Gamma, \tau \text{ is of type } \gamma\}$;

(i) Xs to denote the set $\{x_1, \ldots, x_{|LM|}\}$ and Ys to denote the set $\{y_1, \ldots, y_{|ET|}\}$;

(j) $\mathcal{B}$ to denote the set of bipartite graphs[10] of the form (X, Y, E) where $X \subseteq Xs$, $Y \subseteq Ys$, and $E \subseteq (Xs \times Ys) \cup (Ys \times Xs)$;

(k) $\mathcal{G}$ to denote the set of labelled bipartite graphs $((X, Y, E), \psi)$ where $(X, Y, E) \in \mathcal{B}$ and $\psi : (Xs \cup Ys) \to (LM \cup ET)$;

(l) We will use $CG_\top$ to denote the special graph $((\emptyset, \emptyset, \emptyset), \emptyset) \in \mathcal{G}$.

Functions. We assume bijections $h_x : LM \to Xs$; and $h_y : ET \to Ys$.

Implementation. We will assume the following implementation details:

1. The elements of Γ are assumed to be unary predicate symbols, and the type-definitions T_Γ in Defn. 5.3 will be implemented as predicate-definitions in B. That is, if a ground-term τ is of type $\gamma \in \Gamma$ (that is, $\tau \in T_\gamma$ in Defn. 5.3) then $\gamma(\tau) \in B$. We will therefore refer to the mode-language $\mathcal{L}_{T_\Gamma,\mathrm{M},d}$ in Defn. 5.6 as $\mathcal{L}_{B,\mathrm{M},d}$;
2. An MDIE implementation that, given B, M, d, ensures for any ground definite-clause e returns a unique ground definite-clause $\bot_{B,\mathrm{M},d}(e) \in \mathcal{L}_{B,\mathrm{M},d}$ if it exists or $\emptyset$ otherwise. In addition, if $\bot_{B,\mathrm{M},d}(e) \in \mathcal{L}_{B,\mathrm{M},d}$, we assume the MDIE implementation has been extended to return at least one matching $\lambda\mu$-sequence for $\bot_{B,\mathrm{M},d}$.

5.2.2 Construction of Bottom-Graphs

We now define the graph-structures or simply, the graphs constructed from the depth-limited bottom-clauses.

Definition 5.7 (Literals Set)**.** *Given background knowledge B, a set of modes* M*, and a depth-limit d, let C be a clause in $\mathcal{L}_{B,\mathrm{M},d}$. We define $Lits_{B,\mathrm{M},d}(C)$, or simply $Lits(C)$ as follows:*

(i) If $C = \emptyset$ then $Lits(C) = \emptyset$;

[10] A directed graph $G = (V, E)$ is called bipartite if there is a 2-partition of V into sets X, Y, s.t. there are no vertices $a, b \in X$ (resp. Y) s.t. $(a, b) \in E$. We will sometimes denote such a bipartite graph by (X, Y, E), where it is understood that $V = X \cup Y$.

(ii) If $C \neq \emptyset$, let $\mathcal{LM}$ be the set of all $\lambda\mu$-sequences for C. Then $Lits(C) = \{(\lambda_i, \mu_i) : S \in \mathcal{LM}$ and (λ_i, μ_i) is in sequence $S\}$.

The definition for $Lits(\cdot)$ requires all $\lambda\mu$-sequences to ensure that $Lits$ is well-defined. In practice, we restrict ourselves to the $\lambda\mu$-sequences identified by the MDIE implementation. If these are a subset of all $\lambda\mu$-sequences, then the resulting clause-graph will be "more general" than that obtained with all $\lambda\mu$-sequences (see subsection 5.2.3).

Example 5.10. *We revisit the gparent example. Let* $\mathrm{M} = \{\mu_1, \mu_2, \mu_3, \mu_4\}$*, where*

$$\mu_1 = modeh(gparent(+person, -person)), \mu_2 = modeb(father(+person, -person)),$$
$$\mu_3 = modeb(mother(+person, -person), \mu_4 = modeb(parent(+person, -person)).$$

Let's assume that background knowledge B contains the type-definitions: $person(henry)$, $person(john)$, $person(jane)$, $person(alice)$; and the depth-bound be $d = 2$. Let $C = \bot_{B,\mathrm{M},d}(e)$ as in Example 5.6.

1. *Here C is the set consisting of the literals: $\{gparent(henry, john)$, $\neg father(henry, jane)$, $\neg mother(jane, john)$, $\neg mother(jane, alice)$, $\neg parent(henry, jane)$, $\neg parent(jane, john)$, $\neg parent(jane, alice)\}$.*

2. *$\Lambda_C = \{\lambda_1, \lambda_2, \ldots, \lambda_7\}$ where $\lambda_1 = gparent(henry, john)$, $\lambda_2 = father(henry, jane)$, $\lambda_3 = mother(jane, john)$, $\lambda_4 = mother(jane, alice)$, $\lambda_5 = parent(henry, jane)$, $\lambda_6 = parent(jane, john)$, $\lambda_7 = parent(jane, alice)$.*

3. *$C \in \mathcal{L}_{B,\mathrm{M},d}$ because $S = \langle(\lambda_1, \mu_1), (\lambda_2, \mu_2), (\lambda_3, \mu_3), (\lambda_4, \mu_3), (\lambda_5, \mu_4), (\lambda_6, \mu_4), (\lambda_7, \mu_4)\rangle$ is a $\lambda\mu$-sequence for C. Some other permutations of S will also be $\lambda\mu$-sequences. The reader can verify that the terms in λ-components of S are correctly typed; input terms in the body literals appear after corresponding output terms in body-literals earlier in the λ-components of S, or as input-terms in λ_1; the output-term in λ_1 appears as an output-term in some λ later in the sequence S.*

4. *Then $Lits(C) = \{(\lambda_1, \mu_1), (\lambda_2, \mu_2), (\lambda_3, \mu_3), (\lambda_4, \mu_3), (\lambda_5, \mu_4), (\lambda_6, \mu_4), (\lambda_7, \mu_4)\}$.*

Definition 5.8 (Terms Set)**.** *Given background knowledge B, a set of modes* M*, a depth-limit d, let $C \in \mathcal{L}_{B,\mathrm{M},d}$. We define $Terms_{B,\mathrm{M},d}(C)$, or simply $Terms(C)$ as follows:*

If $Lits(C) = \emptyset$, then $Terms(C) = \emptyset$. Otherwise, for any pair $(\lambda, \mu) \in Lits(C)$, let $Ts((\lambda, \mu)) = \{(\lambda, \mu, \pi) : \pi$ is a place-number s.t. $ModeType(\mu, \pi) = (\cdot, \gamma)$ for some $\gamma \in \Gamma\}$. Then $Terms(C) = \bigcup_{x \in Lits(C)} Ts(x)$.

Example 5.11. *In Example 5.10, $Lits(C) = \{(\lambda_1, \mu_1), (\lambda_1, \mu_1), \ldots, (\lambda_7, \mu_4)\}$. Therefore, $Terms(C) = \{(\lambda_1, \mu_1, \langle 1\rangle), (\lambda_1, \mu_1, \langle 2\rangle), (\lambda_2, \mu_2, \langle 1\rangle), (\lambda_2, \mu_2, \langle 2\rangle), \ldots, (\lambda_7, \mu_4, \langle 1\rangle), (\lambda_7, \mu_4, \langle 2\rangle)\}$.*

Definition 5.9 (Clause-Graphs)**.** *Given background knowledge B, a set of modes* M*, and a depth-limit d, we define a function $ClauseToGraph : \mathcal{L}_{B,\mathrm{M},d} \to \mathcal{G}$ as follows.*

If $C = \emptyset$ then $ClauseToGraph(C) = CG_\top$ (see subsection 5.2.1). Otherwise, the clause-graph of C is given as $ClauseToGraph(C) = (G, \psi) \in \mathcal{G}$ where

- *The graph $G = (X, Y, E)$ is defined as:*
  - $X = \{x_i : (\lambda, \mu) \in Lits(C), x_i = h_x((\lambda, \mu))\}$;
  - $Y = \{y_j : (\lambda, \mu, \pi) \in Terms(C),\ TermType((\lambda, \mu, \pi)) = (\tau, \gamma),\ ModeType(\mu, \pi) \in \{(+, \gamma), (-, \gamma)\},\ y_j = h_y((\tau, \gamma))\} \cup \{y_j : (\lambda, \mu, \pi) \in Terms(C),\ TermType((\lambda, \mu, \pi)) = (\tau, \gamma),\ ModeType(\mu, \pi) = (\#, \gamma),\ y_j = h_y((\tau, \#\gamma))\}$;
  - $E = E_{in} \cup E_{out}$, *where:*

$$E_{in} = \{(y_j, x_i) : (\lambda, \mu, \pi) \in Terms(C), x_i = h_x((\lambda, \mu)), (\tau, \gamma) = TermType((\lambda, \mu, \pi)), y_j = h_y((\tau, \gamma)), ModeType(\mu, \pi) = (+, \gamma)\}, \text{and}$$
$$E_{out} = \{(x_i, y_j) : (\lambda, \mu, \pi) \in Terms(C), x_i = h_x((\lambda, \mu)), (\tau, \gamma) = TermType((\lambda, \mu, \pi)), y_j = h_y((\tau, \gamma)), ModeType(\mu, \pi) \in \{(-, \gamma), (\#, \gamma)\}\}.$$

- *The vertex-labelling function ψ is defined as*

$$\psi(v) = \begin{cases} h_x^{-1}(v) & \text{if } v \in X, \\ h_y^{-1}(v) & \text{if } v \in Y. \end{cases}$$

In subsection 5.2.3, we show $ClauseToGraph(\cdot)$ is an injective function.

Example 5.12. *We continue Example 5.11. Recall $Terms(C) = \{(\lambda_1, \mu_1, \langle 1\rangle), (\lambda_1, \mu_1, \langle 2\rangle), (\lambda_2, \mu_2, \langle 1\rangle), \ldots, (\lambda_7, \mu_4, \langle 2\rangle)\}$. Then, in Defn. 5.9, the term-types are given as follows: $TermType((\lambda_1, \mu_1, \langle 1\rangle)) = (henry, person)$, $TermType((\lambda_1, \mu_1, \langle 2\rangle)) = (john, person)$, $TermType((\lambda_2, \mu_2, \langle 1\rangle)) = (henry, person)$, $\ldots$, $TermType((\lambda_7, \mu_4, \langle 2\rangle)) = (alice, person)$. Then $ClauseToGraph(C)$ is as follows:*

- *$G = (X, Y, E)$ where*
  - $X = \{x_1, x_2, \ldots, x_7\}$, *with* $x_1 = h_x((\lambda_1, \mu_1))$; $x_2 = h_x((\lambda_2, \mu_2))$; $\ldots$; $x_7 = h_x((\lambda_7, \mu_4))$
  - $Y = \{y_1, y_2, y_3, y_4\}$, *with* $y_1 = h_y((henry, person))$; $y_2 = h_y((john, person))$; $y_3 = h_y((jane, person))$; $y_4 = h_y((alice, person))$

– $E = E_{in} \cup E_{out}$, *with*

$$E_{in} = \{(y_1, x_1), (y_1, x_2), (y_1, x_5), (y_3, x_3), (y_3, x_4), (y_3, x_6), (y_3, x_7)\}$$
$$E_{out} = \{(x_1, y_2), (x_2, y_3), (x_3, y_2), (x_4, y_4), (x_5, y_3), (x_6, y_2), (x_7, y_4)\}.$$

- *The vertex-labelling* ψ *is such that* $\psi(x_1) = (\lambda_1, \mu_1)$; $\psi(x_2) = (\lambda_2, \mu_2)$; $\psi(x_3) = (\lambda_3, \mu_3)$; $\psi(x_4) = (\lambda_4, \mu_3)$; $\psi(x_5) = (\lambda_5, \mu_4)$; $\psi(x_6) = (\lambda_6, \mu_4)$; $\psi(x_7) = (\lambda_7, \mu_4)$; $\psi(y_1) = (henry, person)$; $\psi(y_2) = (john, person)$; $\psi(y_3) = (jane, person)$; $\psi(y_4) = (alice, person)$.

The reader can compare this to the graph shown diagrammatically in Figure 5.1.

Example 5.13. *Examples 5.10–5.12 do not illustrate what happens when we have multiple matching mode-declarations. To illustrate this we repeat the exercise with Example 5.8 (for consistency, we now use the symbol* $\mathbb{R}$ *instead of real). In that example,* $\mathrm{M} = \{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_6\}$ *where* $\mu_1 = modeh(p(+int))$, $\mu_2 = modeh(p(+\mathbb{R}))$, $\mu_3 = modeb(q(+int))$, $\mu_4 = modeb(q(+\mathbb{R}))$, $\mu_5 = modeb(r(+int))$, $\mu_6 = modeb(r(+\mathbb{R}))$. *Let the depth-limit* $d = 1$.

1. *Here* $C = \{\ p(1), \neg q(1), \neg r(1)\ \}$.
2. $\Lambda_C = \{\lambda_1, \lambda_2, \lambda_3\}$, *where* $\lambda_1 = p(1)$, $\lambda_2 = q(1)$, $\lambda_3 = r(1)$.
3. $C \in \mathcal{L}_{B,\mathrm{M},d}$ *since there is at least one* $\lambda\mu$*-sequence for* C *(in fact, there are 4 matching* $\lambda\mu$*-sequences: see Example 5.8).*
4. $Lits(C) = \{(\lambda_1, \mu_1), (\lambda_2, \mu_3), (\lambda_3, \mu_5), (\lambda_1, \mu_2), (\lambda_2, \mu_4), (\lambda_3, \mu_6)\}$.
5. *We note that the term* 1 *is at place-number* $\langle 1 \rangle$ *in all the three literals.*
6. *Then* $Terms(C) = \{(\lambda_1, \mu_1, \langle 1 \rangle), (\lambda_2, \mu_3, \langle 1 \rangle), \ldots, (\lambda_3, \mu_6, \langle 1 \rangle)\}$
7. *Then, in Defn. 5.9,* $TermType((\lambda_1, \mu_1, \langle 1 \rangle)) = (1, int)$, $TermType((\lambda_2, \mu_3, \langle 1 \rangle)) = (1, int)$, $\ldots$, $TermType((\lambda_3, \mu_6, \langle 1 \rangle)) = (1, \mathbb{R})$.

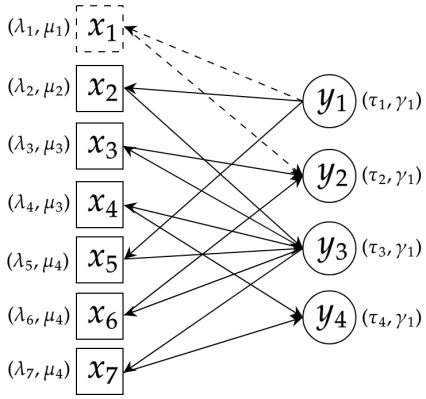
The reader can verify that $ClauseToGraph(C) = (G, \cdot)$ *where* $G = (X, Y, E)$ *s.t.*

- $X = \{x_1, x_2, \ldots, x_6\}$, *with* $x_1 = h_x((\lambda_1, \mu_1))$, $x_2 = h_x((\lambda_2, \mu_3))$,$\ldots$, $x_6 = h_x((\lambda_3, \mu_6))$
- $Y = \{y_1, y_2\}$, *with* $y_1 = h_y((1, int))$ *and* $y_2 = h_y((1, \mathbb{R}))$
- $E = \{(y_1, x_1), (y_1, x_2), (y_1, x_3), ((y_2, x_4), (y_2, x_5), (y_2, x_6)\}$.

It is now straightforward to define graphs from most-specific clauses.

Definition 5.10 (Bottom-Graphs)**.** *Given a data instance* $e \in \mathcal{E}$, *and* B, M, d *as before, let* $\bot_{B,\mathrm{M},d}(e)$ *be the (depth-bounded) most-specific ground definite-clause for* e. *We define* $BotGraph_{B,\mathrm{M},d}(e) \equiv BotGraph : \mathcal{E} \to \mathcal{G}$ *as* $BotGraph(e) = ClauseToGraph(\bot_{B,\mathrm{M},d}(e))$.

Example 5.14. *For our gparent/2 example described through out this chapter, the bottom-graph for the most-specific clause with $d = 2$ is written as: $BotGraph(e) = ClauseToGraph(\bot_{B,\mu,2}(e))$, which is shown in the following diagram. In the diagram, the "dashed" square-boxes and the "dashed" arrows are shown to indicate the vertices specifying the head of the clause:*



The vertex-labelling in the above graph is as obtained in Example 5.12, where γ_1 denotes the type-name person, τ_1,τ_2, τ_3, τ_4 denote the terms henry, john, jane, alice, respectively. The reader can verify that the diagram above is consistent with the bottom-graph shown in Figure 5.1.

We now provide some properties of clause-graphs.

5.2.3 Some Properties of Clause-Graphs

We note the following properties about clause-graphs. For these properties, we assume background knowledge B, a set of modes M, and a depth-limit d as before. Clause-graphs are elements of the set $\mathcal{G}$ and are structures of the form $((X, Y, E), \psi)$ where (X, Y, E) are bipartite graphs from the set $\mathcal{B}$ (see subsection 5.2.1). We assume $\mathcal{G}$ contains the element $CG_\top = ((\emptyset, \emptyset, \emptyset), \emptyset)$. We also define the following equality relation over elements of $\mathcal{G}$: $((X_i, Y_i, E_i), \psi_i) = ((X_j, Y_j, E_j), \psi_j))$ iff $X_i = X_j$, $Y_i = Y_j$, $E_i = E_j$ and $\psi_i = \psi_j$. Also, given a clause $C = \{l_1, \ldots, l_m, \neg l_{m+1}, \ldots, \neg l_k\}$, where $1 \leq m < k$, we have Λ_C is the set $\{l_1, \ldots, l_{m+1}, \ldots, l_k\}$.

Definition 5.11 ($\preceq_{cg}$). *Let $CG_1 = (G_1, \psi_1)$, $CG_2 = (G_2, \psi_2)$ be elements of $\mathcal{G}$, where $G_1 = (X_1, Y_1, E_1)$ and $G_2 = (X_2, Y_2, E_2)$ Then $CG_1 \preceq_{cg} CG_2$ iff the following hold: (a) $X_1 \subseteq X_2$; (b) $Y_1 \subseteq Y_2$; (c) $E_1 \subseteq E_2$; and (d) $\psi_1 \subseteq \psi_2$.*

Proposition 5.1. *$\langle \mathcal{G}, \preceq_{cg} \rangle$ is partially ordered.*

Proof: To prove this, let $CG = ((X, Y, E), \psi)$, and $CG_i = ((X_i, Y_i, E_i), \psi_i)$.

Reflexive. If $CG \in \mathcal{G}$ then $CG \preceq_{cg} CG$. This follows trivially since $X \subseteq X$, $Y \subseteq Y$, $E \subseteq E$ and $\psi \subseteq \psi$.

Anti-Symmetric. Let $CG_1, CG_2 \in \mathcal{G}$. If $CG_1 \preceq_{cg} CG_2$ and $CG_2 \preceq_{cg} CG_1$ then $CG_1 = CG_2$. Since $CG_1 \preceq_{cg} CG_2$, and $CG_2 \preceq_{cg} CG_1$ $X_1 \subseteq X_2$ and $X_2 \subseteq X_1$. Therefore $X_1 = X_2$. Similarly $Y_1 = Y_2$, $E_1 = E_2$ and $\psi_1 = \psi_2$, and therefore $CG_1 = CG_2$;

Transitive. Let $CG_1, CG_2, CG_3 \in \mathcal{G}$. If $CG_1 \preceq_{cg} CG_2$ and $CG_2 \preceq_{cg} CG_3$ then $CG_1 \preceq_{cg} CG_3$. Since $CG_1 \preceq_{cg} CG_2$ and $CG_2 \preceq_{cg} CG_3$ then $X_1 \subseteq X_2$ and $X_2 \subseteq X_3$. Therefore $X_1 \subseteq X_3$. Similarly, $Y_1 \subseteq Y_3$, $E_1 \subseteq E_3$ and $\psi_1 \subseteq \psi_3$. Therefore, $CG_1 \preceq_{cg} CG_3$.

■

For $CG_1, CG_2 \in \mathcal{G}$, if $CG_1 \preceq_{cg} CG_2$, then we will say CG_1 is more general than CG_2. We note without formal proof that if $CG \in \mathcal{G}$ then $CG_\top \preceq_{cg} CG$.

Remark 5.2. *The following are the consequences of the Defns. 5.7–5.9, and Defn. 5.11:*

(i) Let $C, D \in \mathcal{L}_{B,\mathrm{M},d}$, $CG_1 = ClauseToGraph(C)$, and $CG_2 = ClauseToGraph(D)$ where $CG_1 = ((X_1, Y_1, E_1), \psi_1)$ and $CG_2 = ((X_2, Y_2, E_2), \psi_2)$. If $(X_1 \subseteq X_2)$ then $CG_1 \preceq_{cg} CG_2$. By construction, $X_1 \subseteq X_2$ iff $Lits(C) \subseteq Lits(D)$. It follows that $Terms(C) \subseteq Terms(D)$, and $Y_1 \subseteq Y_2$. Since E_1 contains all the relevant arcs between X_1 and Y_1 and E_2 contains all the relevant arcs between X_2 and Y_2; E_2 will contain all the elements of E_1. Since h_x, h_y are bijections, $\psi_1 \subseteq \psi_2$. Hence $CG_1 \preceq_{cg} CG_2$;

(ii) Let $C, D \in \mathcal{L}_{B,\mathrm{M},d}$. The clause-graphs of C and D are $CG_1 = ClauseToGraph(C) = ((X_1, Y_1, E_1), \psi_1)$ and $CG_2 = ClauseToGraph(D) = ((X_2, Y_2, E_2), \psi_2)$, respectively. Let $\mathcal{LM}_1$ be the set of $\lambda\mu$-sequences for C and $\mathcal{LM}_2$ be the set of $\lambda\mu$-sequences for D. If $\mathcal{LM}_1 \subseteq \mathcal{LM}_2$ then $CG_1 \preceq_{cg} CG_2$. It is evident that $Lits(C) \subseteq Lits(D)$. Therefore $X_1 \subseteq X_2$, which further combines with the observation (i) above, we get $CG_1 \preceq_{cg} CG_2$.

Lemma 5.1 (*Lits*)**.** *The function $Lits : \mathcal{L}_{B,M,d} \to 2^{LM}$ (defined in Defn. 5.7) is well-defined. That is, if $C = D$, then $Lits(C) = Lits(D)$.*

Proof: Assume the contrary. That is, $C = D$ and $Lits(C) \neq Lits(D)$. Since $C = D$, $\Lambda_C = \Lambda_D$. Further, since $Lits(C) \neq Lits(D)$, for some $\lambda_i \in \Lambda_C, \Lambda_D$, there must exist $\mu_i \in \mathrm{M}$ s.t. $(\lambda_i, \mu_i) \in Lits(C)$ and $(\lambda_i, \mu_i) \notin Lits(D)$ or *vice versa*. This is not possible because $Lits(C)$ and $Lits(D)$ contain all $\lambda\mu$-sequences for C, D. ■

Lemma 5.2. *Let $C, D \in \mathcal{L}_{B,\mathrm{M},d}$. Let the corresponding clause-graphs of these two clauses be $CG_1 = ClauseToGraph(C)$ and $CG_2 = ClauseToGraph(D)$. if $C = D$ then $CG_1 = CG_2$.*

Proof: The result holds trivially if $C = D = \emptyset$; therefore we consider $C, D \neq \emptyset$. Let $CG_1 = ((X_1, Y_1, E_1), \psi_1)$ and $CG_2 = ((X_2, Y_2, E_2), \psi_2)$. Since $C = D$, by Lemma 5.1 $Lits(C) = Lits(D)$. From Defn. 5.8, $Terms(C) = Terms(D)$ iff $Lits(C) = Lits(D)$. From Defn. 5.9, $X_1 = X_2$ iff $Lits(C) = Lits(D)$ and $Y_1 = Y_2$ iff $Terms(C) = Terms(D)$. If $X_1 = X_2$ and $Y_1 = Y_2$ then $E_1 = E_2$. Since h_x, h_y are bijections, $\psi_1 = \psi_2$. This implies, $CG_1 = CG_2$. ■

Proposition 5.2 (*ClauseToGraph*)**.** *The function* $ClauseToGraph : \mathcal{L}_{B,\mathrm{M},d} \rightarrow \mathcal{G}$ *(defined in Defn. 5.9) is injective.*

Proof: Let C and D in $\mathcal{L}_{B,\mathrm{M},d}$, and the corresponding clause-graphs of these two clauses be $CG_1 = ClauseToGraph(C)$ and $CG_2 = ClauseToGraph(D)$. We need to show that if $CG_1 = CG_2$ then $C = D$.

Let $CG_1 = (G_1, \psi_1)$ and and $CG_2 = (G_2, \psi_2)$, where $G_1 = (X_1, Y_1, E_1)$ and $G_2 = (X_2, Y_2, E_2)$. Since $CG_1 = CG_2$, $(G_1, \psi_1) = (G_2, \psi_2)$. That is, $X_1 = X_2, Y_1 = Y_2$ and $\psi_1 = \psi_2$. Suppose $C \neq D$. Then, either there is some literal in C that is not in D or *vice versa*. Let $\lambda_i \in C$, and $\lambda_i \notin D$. Let λ_i be the corresponding literal in Λ_C, and $\lambda_i \notin \Lambda_D$. Then since $C \in \mathcal{L}_{B,\mathrm{M},d}$ there must be at least one $\mu_i \in \mathrm{M}$ s.t. $(\lambda_i, \mu_i) \in Lits(C)$. Let $x = h_x((\lambda_i, \mu_i)) \in X_1$. Since h_x is a bijection, and $\lambda_i \notin D$, there will be no other λ and μ such that $h_x((\lambda, \mu)) = x$. Hence $x \notin X_2$. This is a contradiction, because $X_1 = X_2$. Similarly, we can prove for $\lambda_i \in D$ and $\lambda_i \notin C$. ■

Proposition 5.3 (Left-Inverse)**.** $ClauseToGraph(\cdot)$ *has a left-inverse.*

Proof: We show that there is a function $GraphToClause : \mathcal{G} \rightarrow \mathcal{L}_{B,\mathrm{M},d}$ s.t. for all $C \in \mathcal{L}_{B,\mathrm{M},d}$, $GraphToClause(ClauseToGraph(C)) = C$.

Let $CG = ClauseToGraph(C)$. So, $CG = (G, \psi)$, where $G = (X, Y, E)$. For each $x_i \in X$, consider the following sets:

1. $L^+ = \{\lambda_i : x_i \in X, \psi(x_i) = (\lambda_i, \mu_i), \mu_i = modeh(\cdot)\}$;
2. $L^- = \{\neg\lambda_i : x_i \in X, \psi(x_i) = (\lambda_i, \mu_i), \mu_i = modeb(\cdot)\}$.

Let $GraphToClause(CG) = C'$ where $C' = L^+ \cup L^-$. We claim $C = C'$. Assume $C \neq C'$. Then there must be some literal $l_i \in C$ s.t. $l_i \notin C'$ (or *vice versa*). Let the corresponding literal in Λ_C be λ_i. Since $C \in \mathcal{L}_{B,\mathrm{M},d}$, there must be some $\lambda\mu$-sequence (Defn. 5.5) for C s.t. some $(\lambda_i, \mu_i) \in Lits(C)$ (Defn. 5.7) and $h_x((\lambda_i, \mu_i)) \in X$ (Defn. 5.9). Then, by the construction above, $l_i \in C'$, which is a contradiction. Suppose $l_i \in C'$ and $l_i \notin C$. Then there cannot be any (λ_i, μ_i) s.t. $h_x((\lambda_i, \mu_i)) \in X$. By construction, $l_i \notin C'$, which is a contradiction. Therefore there is no $l_i \in C$ and $l_i \notin C'$, or *vice versa*, and hence $C = C'$. ■

Remark 5.3. *We note the following without formal proofs:*

(i) In Defn. 5.10, if there exists a unique $\bot_{B,\mathrm{M},d}(e) \in \mathcal{L}_{B,\mathrm{M},d}$ *then* $BotGraph_{B,\mathrm{M},d}(e)$ *is unique. The proof follows from* $BotGraph_{B,\mathrm{M},d}(e) = ClauseToGraph(\bot_{B,\mathrm{M},d}(e))$.

(ii) In Defn. 5.12, $Antecededent : \mathcal{G} \to \mathcal{G}$ *is well-defined. That is, if* $Antecedent(CG_1) \neq Antecedent(CG_2)$ *then* $CG_1 \neq CG_2$. *Again the proof follows from the contrapositive which is easily seen to hold. Also, we note that Antecedent is many-to-one, that is, it is possible that* $CG_1 \neq CG_2$, *and* $Antecedent(CG_1) = Antecedent(CG_2)$.

(iii) In Defn. 5.13, $UGraph : \mathcal{G} \to \mathcal{G}$ *is well-defined. That is, if* $UGraph(CG_1) \neq UGraph(CG_2)$ *then* $CG_1 \neq CG_2$. *This follows from the contrapositive which is easily shown to hold (that is, if* $CG_1 = CG_2$ *then* $UGraph(CG_1) = UGraph(CG_2)$*).*

Remark 5.4 (Size of a Bottom-Graph)**.** *We find the bound on the size of a clause-graph obtained from a most-specific clause–the bottom-graph–by using the result in [Mug95, Theorem 26].*

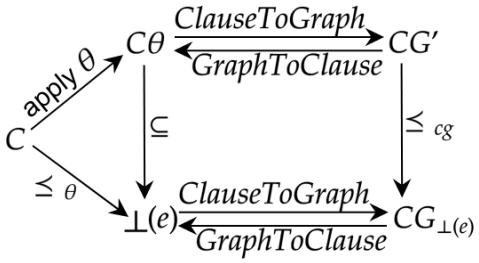
Let $\bot_{B,\mathrm{M},d}(e)$ *denote a most-specific clause for an example* e, *given* B, M, d, *and* $((X, Y, E), \psi)$ *denote the corresponding clause-graph. Let* j^+ *denote an upper-bound on the number of* $+$ *arguments in modeb declarations in* M *and the number of* $-$,# *arguments in modeh declarations in* M. *Let* j^- *denote an upper-bound on the number of* $-$,# *arguments in modeb declarations in* M *and the number of* $+$ *arguments in modeh declarations in* M. *Then the size of* $\bot_{B,\mathrm{M},d}$ *is bounded by* $(r|\mathrm{M}|j^+j^-)^{dj^+}$, *where* r *is a constant called the "recall" number (see [Mug95] for details). For each literal in* $\bot_{B,\mathrm{M},d}$ *there is 1 vertex in* X. *Also, for every argument in each literal in* $\bot_{B,\mathrm{M},d}$ *there is a vertex in* Y. *It is straightforward to see that the size of the corresponding clause-graph is bounded by* $(r|\mathrm{M}|j^+j^-)^{dj^+}(1 + j^+ + j^-)$.

Finally, we relate the clausal explanations found by some MDIE systems using the ordering $\preceq_\theta$ defined over clauses in [Plo70].[11] Given background knowledge B and a clause e, we will say a clause C is a clausal explanation for e if $B \cup \{C\} \models e$.

Remark 5.5 (Relation to Clausal Explanations)**.** *Let* $\bot_{B,\mathrm{M},d}(e)$ *be the ground most-specific definite-clause using MDIE. Let* C *be a clause (not necessarily ground). We show the following: If* $C\theta \subseteq \bot_{B,\mathrm{M},d}(e)$ *and* $C\theta \in \mathcal{L}_{B,\mathrm{M},d}$, *then there exists a clause-graph* CG' *s.t.* $CG' \preceq_{cg} ClauseToGraph(\bot_{B,\mathrm{M},d}(e))$ *and* $GraphToClause(CG') = C\theta$.

Denoting $\bot_{B,\mathrm{M},d}(e)$ *as* $\bot(e)$ *and* $ClauseToGraph(\bot_{B,\mathrm{M},d})$ *as* $CG_{\bot(e)}$, *the relationships described in this remark is shown diagrammatically as:*

[11] $C_1 \preceq_\theta C_2$ if there exists some substitution θ s.t. $C_1\theta \subseteq C_2$. By convention C_1 is said to be more-general than C_2, and C_2 is said to be more-specific than C_1. It is known that if $C_1 \preceq_\theta C_2$, then $C_1 \models C_2$.



Let $ClauseToGraph(\bot_{B,\mathrm{M},d}(e)) = (G, \psi)$*, with* $G = (X, Y, E)$*. In the following,* $Pos(l) = p$ *if* $l = \neg p$ *is a negative literal, otherwise* $Pos(l) = l$*. Consider the structure* $CG' = (G', \psi')$*, with* $G' = (X', Y', E')$ *obtained as follows:*

(a) $X' = \{x_i : x_i \in X, l_i \in C\theta,\ \lambda_i = Pos(l_i),\ \psi(x_i) = (\lambda_i, \mu_i)\}$*;*

(b) $E' = \{(x_i, y_j) : x_i \in X', (x_i, y_j) \in E\} \cup \{(y_j, x_i) : x_i \in X', (y_j, x_i) \in E\}$*;*

(c) $Y' = \{y_j : (x_i, y_j) \in E' \text{ or } (y_j, x_i) \in E'\}$*;*

(d) For $v \in X' \cup Y'$*,* $\psi'(v) = \psi(v)$*.*

It is evident that G' *is a directed bipartite graph, and* ψ' *is defined for every vertex in* G'*. So,* $CG' \in \mathcal{G}$*. By construction,* CG' *has the following properties: (i)* $X' \subseteq X$ *and* $Y' \subseteq Y$*; (ii)* $E' \subseteq E$*; and (iii)* $\psi' \subseteq \psi$*. Therefore* $CG' \preceq_{cg} (G, \psi)$*. Since the vertices in* X' *are obtained using only the literals in* $C\theta$*, it follows that* $GraphToClause(CG) = C\theta$*.*

Since $C\theta \subseteq \bot_{B,\mathrm{M},d}(e)$*,* $C\theta \models \bot_{B,\mathrm{M},d}(e)$*. Further, since* $C \preceq_\theta C\theta$*,* $C \models C\theta$*. It follows that* $B \cup C \models B \cup \bot_{B,\mathrm{M},d}(e)$*. Since* $B \cup \bot_{B,\mathrm{M},d}(e) \models e$*, then* $B \cup C \models e$*. That is,* C *is a clausal explanation for* e*.*

The bottom-graphs defined here are not immediately suitable for GNNs for the task of graph-classification. Some graph-transformations are needed before providing them as input to a GNN. We describe these transformations next.

5.2.4 Transformations for Graph Classification by a GNN

We now describe functions used to transform bottom-graphs into a form suitable for the GNN implementations we consider in this chapter. The definite-clause representation of graphs that we use (an example follows below) contains all the information about the graph in the antecedent of the definite-clause. The following function extracts the corresponding parts of the bottom-graph.

Definition 5.12 (Antecedent-Graphs)**.** *We define function* $Antecedent : \mathcal{G} \to \mathcal{G}$ *as follows. Let* $(G, \psi) \in \mathcal{G}$*, where* $G = (X, Y, E)$ *is a directed bipartite graph. Let* $X_h = \{x \in X : \psi(x) = (\lambda, \mu) with\ \mu = modeh(\cdot)\}$*. We define* (G', ψ') *where* $G' = (X', Y', E')$ *such that*

- $X' = X - X_h$,
- $Y' = \{y \in Y : \exists x \in X' \text{ s.t. } (x, y) \in E \text{ or } (y, x) \in E\}$,
- $E' = E - \{(v_i, v_j) : v_i \in X_h\} - \{(v_j, v_i) : v_i \in X_h\}$,

and $\psi'(v_i) = \psi(v_i)$ *for all* $v_i \in X' \cup Y'$. *Then* $\mathit{Antecedent}((G, \psi)) = (G', \psi')$.

Most GNN implementations, including those used in this chapter, require graphs to be undirected [Ham20]. Furthermore, an undirected graph representation allows an easy exchange of messages across multiple relations (the X-nodes) resulting in unfolding their internal dependencies. We define a function that converts directed clause-graphs to undirected clause-graphs as follows:

Definition 5.13 (Undirected Clause-Graphs)**.** *We define a function* $\mathit{UGraph} : \mathcal{G} \to \mathcal{G}$ *as follows. Let* $(G, \psi) \in \mathcal{G}$, *where* $G = (X, Y, E)$ *is a directed bipartite graph. We define* (G', ψ'), *where* $G' = (X', Y', E')$ *such that*

- $X' = X$,
- $Y' = Y$,
- $E' = E \cup \{(v_j, v_i) : (v_i, v_j) \in E\}$,

and $\psi'(v_i) = \psi(v_i)$ *for all* $v_i \in X' \cup Y'$. *Then* $\mathit{UGraph}((G, \psi)) = (G', \psi')$.

In fact, graphs for GNNs are not actually in $\mathcal{G}$. GNN implementations usually require vertices in a graph to be labelled with numeric feature-vectors. This requires a modification of the vertex-labelling to be a function from vertices to real-vectors of some finite length. The final transformation converts the vertex-labelling of a graph in $\mathcal{G}$ into a suitable form.

Definition 5.14 (Vectorise)**.** *Let* $(G, \psi) \in \mathcal{G}$, *where* $G = (X, Y, E)$. *Assume we are given a set of modes* M. *Let* $\Gamma_\#$ *be the set of all type-names* $\gamma \in \Gamma - \{\mathbb{R}, \#\mathbb{R}\}$ *such that* $\#\gamma$ *in some mode* $\mu \in \mathrm{M}$. *Let* $\mathrm{T}_\# \subseteq \mathrm{T}$ *be the set of ground-terms of types in* $\Gamma_\#$.[12]

For $v \in X \cup Y$, *let us define the following four functions from* $X \cup Y$ *to the set of all*

[12] That is, $\Gamma_\#$ is the set of all #-ed, non-numeric type-names in M, and $\mathrm{T}_\#$ is the set of all ground-terms of #-ed non-numeric types.

real vectors of finite length:

$$f_\rho(v) = \begin{cases} onehot(P, r) & \text{if } v \in X, h(v) = (\lambda, \cdot) \text{ and } predsym(\lambda) = r \\ \mathbf{0}^{|P|} & \text{otherwise} \end{cases}$$

$$f_\gamma(v) = \begin{cases} onehot(\Gamma, \gamma) & \text{if } v \in Y \text{ and } h(v) = (\tau, \gamma) \\ \mathbf{0}^{|\Gamma|} & \text{otherwise} \end{cases}$$

$$f_\tau(v) = \begin{cases} onehot(\mathrm{T}_\#, \tau) & \text{if } v \in Y \text{ and } h(v) = (\tau, \#\gamma) \text{ and } \gamma \notin \mathbb{R} \\ \mathbf{0}^{|\mathrm{T}_\#|} & \text{otherwise} \end{cases}$$

$$f_\mathbb{R}(v) = \begin{cases} [\tau] & \text{if } v \in Y \text{ and } h(v) = (\tau, \#\mathbb{R}) \\ \mathbf{0}^1 & \text{otherwise} \end{cases}$$

where $\mathbf{0}^d$ denotes the zero-vector of length d; $predsym(l)$ is a function that returns the name and arity of literal l; and $onehot(S, x)$ denotes a one-hot vector encoding of $x \in S$.[13]

Let $Vectorise$ be a function defined on $\mathcal{G}$ as follows: $Vectorise((G, \psi)) = (G, \psi')$ where $\psi'(v) = f_\rho(v) \oplus f_\gamma(v) \oplus f_\tau(v) \oplus f_\mathbb{R}(v)$ for each $v \in X \cup Y$. Here $\oplus$ denotes vector concatenation.

We note that the vectors in the vertex-labelling from $Vectorise$ should not be confused with the vector obtained using the Vec function employed within a GNN (see Figure 4.1) in section 4.1. The purpose of that function is to obtain a low-dimensional real-valued vector representation for an entire graph (usually for problems of graph-classification).

Example 5.15. *Recall the most-specific clause for the $gparent(henry, john)$ in Example 5.1:*

$$gparent(henry, john) \leftarrow$$
$$father(henry, jane),\ mother(jane, john),\ mother(jane, alice),$$
$$parent(henry, jane),\ parent(jane, john),\ parent(jane, alice).$$

The clause-graph and corresponding antecedent-graph are shown below.
Assume the following sets:

$$P = \{gparent/2, father/2, mother/2, parent/2\},$$
$$\Gamma = \{person\},$$
$$\Gamma_\# = \emptyset.$$

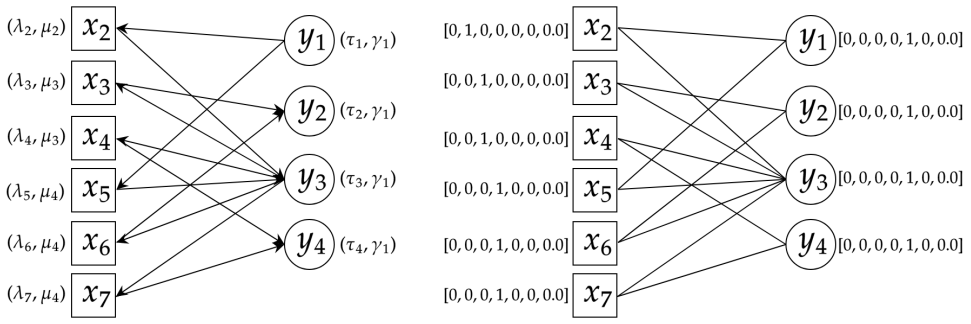
Additionally, since the mode-language in Example 5.1 does not have any #'ed arguments, $\mathrm{T}_\# = \emptyset$. Therefore, f_ρ is a 4-dimensional (one-hot encoded) vector (since $|P| = 4$); f_γ

[13] A one-hot vector encoding of an element x in a set S assumes a 1-1 mapping N from elements of S to $\{1, \ldots, |S|\}$. If $x \in S$ and $onehot(S, x) = \mathbf{v}$ then $\mathbf{v}$ is a vector of dimension $|S|$ s.t. $N(x)$'th entry in $\mathbf{v}$ is 1 and all other entries in $\mathbf{v}$ are 0.

is a 1-dimensional vector (since $|Gamma| = 1$); f_τ is a 1-dimensional vector containing 0 (since $|\mathrm{T}_{\#}| = 0$); and $f_{\mathbb{R}}$ is a 1-dimensional vector containing 0 (since there are no #'ed numeric terms)). A full tabulation of the vectors involved is provided below, along with the new vertex-labelling that results. In the table, the vertex labels are as obtained in Example 5.12; γ_1 is used to denote the type person.

v	$\psi(v)$	$f_\rho(v)^\top$	$f_\gamma(v)^\top$	$f_\tau(v)^\top$	$f_{\mathbb{R}}(v)^\top$	$\psi'(v)^\top$
x_2	(λ_2, μ_2)	$[0,1,0,0]$	$[0]$	$[0]$	$[0.0]$	$[0,1,0,0,0,0,0.0]$
x_3	(λ_3, μ_3)	$[0,0,1,0]$	$[0]$	$[0]$	$[0.0]$	$[0,0,1,0,0,0,0.0]$
x_4	(λ_4, μ_3)	$[0,0,1,0]$	$[0]$	$[0]$	$[0.0]$	$[0,0,1,0,0,0,0.0]$
x_5	(λ_5, μ_4)	$[0,0,0,1]$	$[0]$	$[0]$	$[0.0]$	$[0,0,0,1,0,0,0.0]$
x_6	(λ_6, μ_4)	$[0,0,0,1]$	$[0]$	$[0]$	$[0.0]$	$[0,0,0,1,0,0,0.0]$
x_7	(λ_7, μ_4)	$[0,0,0,1]$	$[0]$	$[0]$	$[0.0]$	$[0,0,0,1,0,0,0.0]$
y_1	(τ_1, γ_1)	$[0,0,0,0]$	$[1]$	$[0]$	$[0.0]$	$[0,0,0,0,1,0,0.0]$
y_2	(τ_2, γ_1)	$[0,0,0,0]$	$[1]$	$[0]$	$[0.0]$	$[0,0,0,0,1,0,0.0]$
y_3	(τ_3, γ_1)	$[0,0,0,0]$	$[1]$	$[0]$	$[0.0]$	$[0,0,0,0,1,0,0.0]$
y_4	(τ_4, γ_1)	$[0,0,0,0]$	$[1]$	$[0]$	$[0.0]$	$[0,0,0,0,1,0,0.0]$

The following figures show: (a) the antecedent graph and (b) the vectorised, undirected, antecedent graph for the gparent example. We call the structure in (b) as a BotGNN-Graph, the definition of which is provided later.



(a) Antecedent-graph

(b) Vectorised, undirected, antecedent-graph

The example above does not have any #-ed arguments in the modes M. In the following example, we consider modes that have #-ed arguments (of types: $\mathbb{R}$ and not $\mathbb{R}$) and repeat the same exercise: starting with the construction of the bottom-graph. Then we show how the function *Vectorise* results in a vectorised graph suitable for a GNN.

Example 5.16. *Let* M *be the set of modes* $\{\mu_1, \mu_2, \mu_3\}$ *where* $\mu_1 = modeh(p(+\mathbb{R}))$, $\mu_2 = modeb(q(+\mathbb{R}, \#colour))$, $\mu_3 = modeb(r(\#colour, \#\mathbb{R}))$. *Let the depth-limit* $d = 1$ *and that the background knowledge contains the type-definitions* $colour(white)$ *and*

colour$(black)$. *Let* C *be a ground definite-clause* $p(1.0) \leftarrow q(1.0, white), r(white, 1.0)$. *The following are obtained based on the definitions:*

- $C = \{p(1.0), \neg q(1.0, white), \neg r(white, 1.0)\}$.
- $\Lambda_C = \{\lambda_1, \lambda_2, \lambda_3\}$, *where* $\lambda_1 = p(1.0)$, $\lambda_2 = q(1.0, white)$, $\lambda_3 = r(white, 1.0)$.
- C *is in* $\mathcal{L}_{B,\mathrm{M},d}$ *since there is at least one* $\lambda\mu$*-sequence for* C. *Here we have one such sequence:* $\langle(\lambda_1, \mu_1), (\lambda_2, \mu_2), (\lambda_3, \mu_3)\rangle$.
- $Lits(C) = \{(\lambda_1, \mu_1), (\lambda_2, \mu_2), (\lambda_3, \mu_3)\}$.
- $Terms(C) = \{(\lambda_1, \mu_1, \langle 1\rangle), (\lambda_2, \mu_2, \langle 1\rangle), (\lambda_2, \mu_2, \langle 2\rangle), (\lambda_3, \mu_3, \langle 1\rangle)(\lambda_3, \mu_3, \langle 2\rangle)\}$.
- $TermType((\lambda_1, \mu_1, \langle 1\rangle)) = (1.0, \mathbb{R})$, $TermType((\lambda_2, \mu_2, \langle 1\rangle)) = (1.0, \mathbb{R})$, $TermType((\lambda_2, \mu_2, \langle 2\rangle)) = (white, \#colour)$, $TermType((\lambda_3, \mu_3, \langle 1\rangle)) = (1.0, \#\mathbb{R})$ $TermType((\lambda_3, \mu_3, \langle 2\rangle)) = (white, \#colour)$.

Then, $ClauseToGraph(C) = (G, \psi)$, *where* $G = (X, Y, E)$ *s.t.*

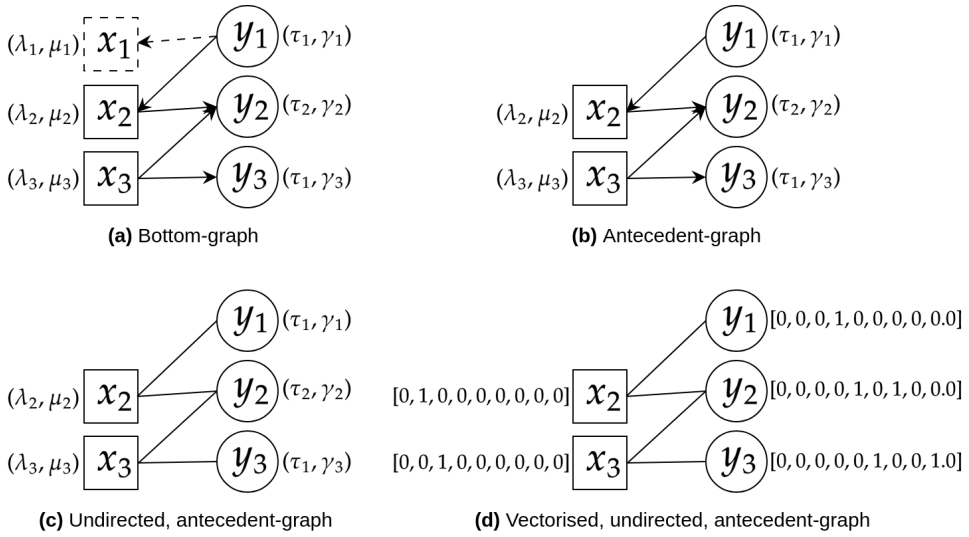
- $X = \{x_1, x_2, x_3\}$, *where* $x_1 = h_x((\lambda_1, \mu_1))$, $x_2 = h_x((\lambda_2, \mu_2))$ *and* $x_3 = h_x((\lambda_3, \mu_3))$
- $Y = \{y_1, y_2, y_3\}$, *where* $y_1 = h_y((1.0, \mathbb{R}))$, $y_2 = h_y((white, \#colour))$, $y_3 = h_y((1.0, \#\mathbb{R}))$
- $E = \{(y_1, x_1), (y_1, x_2), (x_2, y_2), (x_3, y_2), (x_3, y_3)\}$,

and the vertex-labelling ψ *is given as follows:* $\psi(x_1) = (\lambda_1, \mu_1)$, $\psi(x_2) = (\lambda_2, \mu_2)$, $\psi(x_3) = (\lambda_3, \mu_3)$, $\psi(y_1) = (1.0, \mathbb{R})$, $\psi(y_2) = (white, \#class)$, $\psi(y_3) = (1.0, \#\mathbb{R})$.
In this example, we assume the following sets: $P = \{p/1, q/2, r/2\}$, $\Gamma = \{\mathbb{R}, \#colour, \#\mathbb{R}\}$, $\Gamma_{\#} = \{\#colour\}$, $\mathrm{T}_{\#} = \{white, black\}$.

The graph (G, ψ) *constructed above is the bottom-graph for this particular example. The feature-vectors obtained from the functions in* $Vectorise$ *are tabulated below. In the table,* $\tau_1 = 1.0$, $\tau_2 = white$, $\gamma_1 = \mathbb{R}$, $\gamma_2 = \#colour$, $\gamma_3 = \#\mathbb{R}$.

v	$\psi(v)$	$f_\rho(v)^\top$	$f_\gamma(v)^\top$	$f_\tau(v)^\top$	$f_{\mathbb{R}}(v)^\top$	$\psi'(v)^\top$
x_2	(λ_2, μ_2)	$[0, 1, 0]$	$[0, 0, 0]$	$[0, 0]$	$[0.0]$	$[0, 1, 0, 0, 0, 0, 0, 0, 0.0]$
x_3	(λ_3, μ_3)	$[0, 0, 1]$	$[0, 0, 0]$	$[0, 0]$	$[0.0]$	$[0, 0, 1, 0, 0, 0, 0, 0, 0.0]$
y_1	(τ_1, γ_1)	$[0, 0, 0]$	$[1, 0, 0]$	$[0, 0]$	$[0.0]$	$[0, 0, 0, 1, 0, 0, 0, 0, 0.0]$
y_2	(τ_2, γ_2)	$[0, 0, 0]$	$[0, 1, 0]$	$[1, 0]$	$[0.0]$	$[0, 0, 0, 0, 1, 0, 1, 0, 0.0]$
y_3	(τ_1, γ_3)	$[0, 0, 0]$	$[0, 0, 1]$	$[0, 0]$	$[1.0]$	$[0, 0, 0, 0, 0, 1, 0, 0, 1.0]$

The following figure shows how the final vectorised graph is constructed from the bottom-graph (the dotted square-box and the dotted arrow are shown to indicate the vertex specifying the head of the clause C*):*



The functions $Antecedent$, $UGraph$ and $Vectorise$ transform bottom-graphs into a form suitable for GNNs by straightforward composition.

Definition 5.15 (Graph Transformation)**.** *We define a transformation function over* $\mathcal{G}$ *as* $TransformGraph(G) = Vectorise(UGraph(Antecedent((G, \psi))))$.

We now have all the pieces for obtaining graphs suitable for GNNs.

Definition 5.16 (BotGNN Graphs)**.** *Given a data instance* $e \in \mathcal{E}$*, background knowledge* B*, a set of modes* M*, a depth-limit* d *as before, we define* $BotGNNGraph_{B,\mathrm{M},d}(e) \equiv BotGNNGraph(e) = TransformGraph(BotGraph_{B,\mathrm{M},d}(e))$.

Figure 5.2 summarises the sequence of computations used in this chapter. We will use the term $BotGNN$ to describe GNNs constructed from BotGNN graphs.

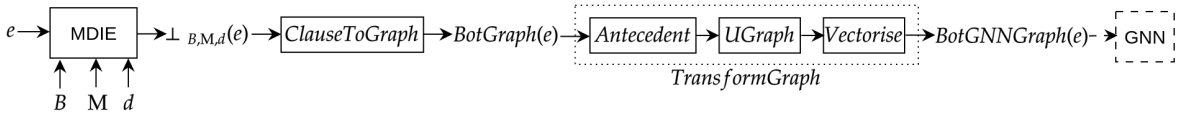
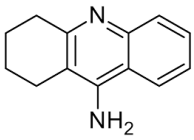


Figure 5.2: Construction and use of bottom-graphs for use by GNNs in this chapter. We note that constituting the transformation of bottom-graphs are for the GNN implementations used in this chapter.

Procedure 6 and Procedure 7 use the definitions, we have introduced, to construct and test $BotGNN$ models. In practice, Step 3 of Procedure 6 and Step 3 of Procedure 7 involve some pre-processing that converts the information in BotGNN graphs into a syntactic form suitable for the implementations used. The procedures assume that data provided as graphs can be represented as definite clauses (Steps 3 in Procedure 6 and 3 in Procedure 7). We illustrate this with an example.

Example 5.17. *The chemical Tacrine is a drug used in the treatment of Alzheimer's disease. It's molecular formula is* $C_{13}H_{14}N_2$*, and its molecular structure is shown in diagrammatic form below:*



One representation of this molecular graph as a definite clause is

$$
\begin{aligned}
&graph(tacrine) \leftarrow \\
&\quad atom(tacrine, a_1, c), \\
&\quad atom(tacrine, a_2, c), \\
&\quad \vdots \\
&\quad atom(tacrine, a_{13}, c), \\
&\quad atom(tacrine, a_{14}, n), \\
&\quad \vdots \\
&\quad bond(tacrine, a_1, a_2, 1), \\
&\quad bond(tacrine, a_2, a_3, 2), \\
&\quad \vdots
\end{aligned}
$$

More generally, a graph $g = (V, E, \psi, \phi)$ (where V denotes the vertices, E denotes the edges, ψ and ϕ are vertex and edge-label mappings, respectively) can be transformed into definite clause of the form $graph(g) \leftarrow Body$, where Body is a conjunction of ground-literals of the form $vertex(g, v_1)$, $vertex(g, v_2)$, ...; $edge(g, e_1)$, $edge(g, e_2)$, ...; $vlabel(g, v_1, \psi(v_1))$, $vlabel(g, v_2, \psi(v_2))$, ...; and $elabel(g, e_1, \psi(e_1))$, $elabel(g, e_2, \psi(e_2))$, ... and so on where $V = \{v_1, v_2, \ldots\}$, $E = \{e_1, e_2, \ldots\}$. More compact representations are possible, but in the experimental section following, we will be using this kind of simple transformation (for molecules, the transformation is done automatically from a standard molecular representation).

Procedure 6 Procedure to construct a *BotGNN* model, given training data $D_{tr} = \{(g_i, y_i)\}_1^N$, where each g_i is a graph and y_i is the class-label for g_i, Background knowledge B, modes M, depth-limit d, and some procedure $TrainGNN$ that trains a graph-based neural network

---

1: **procedure** TrainBotGNN(D_{tr}, B, M, d, $TrainGNN$)
2: $D'_{tr} = \{ (g'_i, y_i) : (g_i, y_i) \in D_{tr}$, e_i be a ground definite-clause representing g_i, $g'_i = BotGNNGraph_{B,\mathrm{M},d}(e_i) \}$
3: Let $BotGNN = TrainGNN(D'_{tr})$
4: **return** $BotGNN$

5.2.5 Note on Differences to Vertex-Enrichment

In this section we clarify some differences of BotGNNs with the approach of vertex-enrichment in GNNs (or VEGNNs) discussed in Chapter 4. An immediate difference is in the nature of the graphs handled by the two approaches. Broadly, VEGNNs require

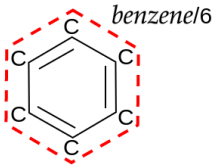
Procedure 7 Procedure to obtain predictions of a *BotGNN* model on a data set given background knowledge B, modes M, depth-limit d, and data D consisting of a set of graphs $\{g_i\}_1^N$

1: **procedure** TESTBOTGNN(D, B, M, d)
2: Let $D' = \{(g_i, g_i') : g_i \in D,\ e_i$ is the definite-clause representation of $g_i,\ g_i' = BotGNNGraph_{B,\mathrm{M},d}(e_i)\}$
3: Let $Pred = \{(g_i, \hat{y}_i) : (g_i, g_i') \in D', \hat{y}_i = BotGNN(g')\}$
4: **return** $Pred$

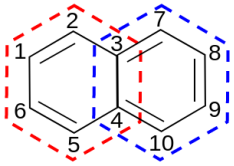
data in a graphical form. VEGNNs retain the most of the original graph-structure, but modify the feature-vectors associated with each vertex of the graph (more on this below). BotGNNs on the other hand do not require data to be a graph. Instead, any data representable as a definite clause are reformulated using the bottom-clause into BotGNN graphs. Recall these are bipartite-graphs, in which both vertices and their labels have a different meaning to the graphs in VEGNNs.

A subtler difference between BotGNNs and VEGNNs arises from how the relational information is included within the graphs constructed in each case. The difference is best illustrated by an example below.

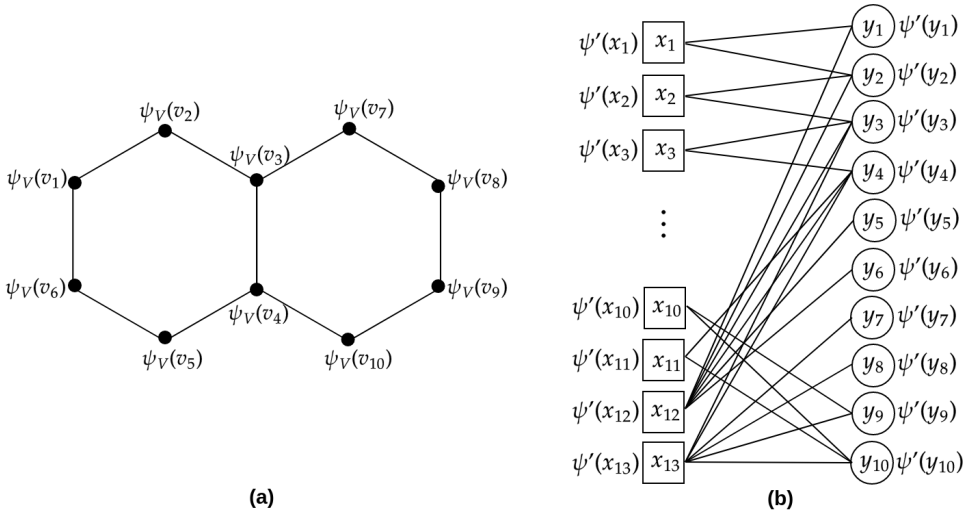
Example 5.18. *Suppose we consider a molecule containing the atoms and bonds, and we want to include the* 6*-ary relation of a benzene ring as shown below.*



In VEGNNs, graphs are represented as tuples of the form $(V, E, \sigma, \psi, \phi)$*, where* V *is the set of vertices (here atoms in the molecule);* E *denotes the edges (bonds in the molecule);* σ *is a neighbourhood function;* ψ *denotes an initial vertex-labelling; and* ϕ *denotes an initial edge-labelling. For each* $v \in V$*, let* $\psi(v)$ *be a real-valued vector of finite dimension. In Chapter 4, any* n*-ary relation in domain-knowledge is treated as a hyperedge, where a hyperedge is a set of* n *vertices in the graph. For any vertex* v *in a graph, let* $h(v)$ *denote the set of predicate symbols such that the corresponding hyper-edge contains* v*. Let* $g/1$ *be a function that maps sets of predicate-symbols to a fixed-length Boolean-valued vector (a "multi-hot" encoding). Thus, a VEGNN is a GNN that operates on graphs obtained from labelled graphs of the form* $(V, E, \sigma, \psi_V, \phi)$*, where* $\psi_V(v) = \psi(v) \oplus g(h(v))$ *(here* $\oplus$ *denotes a concatenation operation). In a VEGNN,* $h(v)$ *is* $\{benzene/6\}$ *for* $v = v_1, \ldots, v_{10}$ *in the graph below (representing the compound naphthalene):*



Thus, the information that v_3, v_4 are members of 2 different benzene rings is not captured in the VEGNNs vertex-labelleing, and we have to rely on the GNN machinery to re-derive this information from the graph structure (if this information is needed). In a BotGNN on the other hand, the two benzene rings are separate vertices in the bipartite graph, which share edges to vertices representing v_3 and v_4. The broad structure of the VEGNN (only vertex-labels are shown for clarity) and the BotGNN graphs for naphthalene are shown in figures (a) and (b) respectively:



$\psi_V/1$ in (a) refers to the vertex-encoding function in Chapter 4, and ψ' in (b) refers to the function defined in Defn. 5.14. For the experimental data in this chapter, the vertex-encoding in Chapter 4 results in vectors whose dimensions are about 10 times more than the $\psi_V/1$ from Defn. 5.14.

The approach to n-ary relations employed by VEGNNs is thus somewhat akin to a *clique-expansion* of the graph containing vertices for terms. In a clique-expansion, all vertices in a hyper-edge—elements of some n-ary relation—are connected together by a labelled hyper-edge. This can introduce a lot of new edges, and some mechanism is needed to distinguish between multiple occurrences of the same relation (an example is the multiple occurrences of benzene rings above). VEGNNs can be seen as achieving the effect of such a clique-expansion, without explicitly adding the new edges, but they do not address the problem of multiple occurrences. BotGNNs can be seen instead as a *star-expansion* of the graph containing vertices for terms. In such a star-expansion, new nodes denoting the relation are introduced, along with edges between the relation-vertex and the term-vertices that are part of the relation (that is, the hyper-edge). Star-expansions of graphs thus contain 2 kinds of vertices, which is similar to the graph constructed by a BotGNN.

5.3 Empirical Evaluation

5.3.1 Aims

Our aim in this section is to investigate the utility of using *BotGNN*s as a technique for including domain-knowledge. That is,

- We investigate whether the performance of a BotGNN that includes domain-knowledge using MDIE is better than the performance of a GNN that does not include domain-knowledge.

As before, later in the chapter, we will also provide additional comparisons against all other methods developed so far in the dissertation.

5.3.2 Materials

Data and Background Knowledge

For the empirical evaluation of our proposed BotGNNs, we use the same 73 benchmark datasets that are used in our studies on DRMs (in Chapter 3) and VEGNNs (in Chapter 4). We use the same background knowledge used in the previous two chapters. Each dataset consists of a set of chemical compounds, which are then converted into bottom-graphs. A simple summary of the resulting bottom-graph datasets is provided below.

# of datasets	Avg # of instances	Avg. of $\|X\|$	Avg. of $\|Y\|$	Avg. of $\|E\|$
73	3032	81	42	937

Figure 5.3: Dataset summary. Each bottom-graph can be represented using $(G, \cdot)$, where $G = (X, Y, E)$, where X represents the vertices corresponding to the relations, Y represents the vertices corresponding to ground terms in the bottom-clause constructed by MDIE, and E represents the edges between X and Y. The last 3 columns are the average number of X, Y and E in each bottom-graph in a dataset.

Algorithms and Machines

The datasets and the background knowledge are written in Prolog. We use the ILP engine, Aleph [Sri01] for the construction of bottom-clauses using MDIE. A Prolog program is then used to extract the relations and ground terms from the bottom-clause. We use YAP compiler for execution of all our Prolog programs. The files containing the relations and ground terms are then parsed by UNIX and MATLAB scripts to construct bottom-graph datasets in a format suitable for GNNs for which we follow the format prescribed

in [KKM+16]. These details are mainly representations of adjacency matrix, vertex labels (feature vectors), class labels, etc.

The GNN variants used here are described in the next section. All the experiments are conducted in a Python environment. The GNN models have been implemented by using the PyTorch Geometric library [FL19]—a popular geometric deep learning extension for PyTorch [PGM+19] enabling easier implementations of various graph convolution and pooling methods.

For all the experiments, we use a machine with Ubuntu (16.04 LTS) operating system, and hardware configuration such as: 64GB of main memory, 16-core Intel Xeon processor, a NVIDIA P4000 graphics processor with 8GB of video memory.

5.3.3 Method

Let D be a set of data-instances represented as graphs $\{(g_1, y_1), \ldots, (g_N, y_N)\}$, where y_i is a class label associated with the graph g_i. We also assume that we have access to background-knowledge B, a set of modes M, a depth-limit d. Our method for investigating the performance of $BotGNN$s uses is straightforward:

(1) Randomly split D into D_{Tr} and D_{Te};

(2) Let $BotGNN$ be the model from Procedure 6 (TrainBotGNN) with backround knowledge B, modes M, depth-limit d, training data D_{Tr} and some GNN implementation (see below);

(3) Let GNN be the model from the GNN implementation without background knowledge, and with D_{Tr};

(4) Let $D'_{Te} = \{g_i : (g_i, y_i) \in D_{Te}\}$;

(5) Obtain the predictions for D'_{Te} of $BotGNN$ using Procedure 7 (TestBotGNN) with background knowledge B, modes M, and depth-limit d;

(6) Obtain the predictions for D'_{Te} using GNN; and

(7) Compare the performance of $BotGNN$ and GNN.

We closely follow the method used in Chapter 4 for the construction of BotGNNs. Relevant details are as follows:

- We have used a 70:30 train-test split for each of the datasets. 10% of the train-set is used as a validation set for hyperparameter tuning.

- The general workflow involved in GNNs was described in Chapter 4 (refer section 4.1). A diagram of the components involved in implementing that workflow for constructing a BotGNN is shown in Figure 5.4. As shown in the figure, a GNN in our implementations consists of three graph convolution blocks and three graph pooling blocks. The convolution and pooling blocks interleave each other (that is, C-P-C-P-C-P).

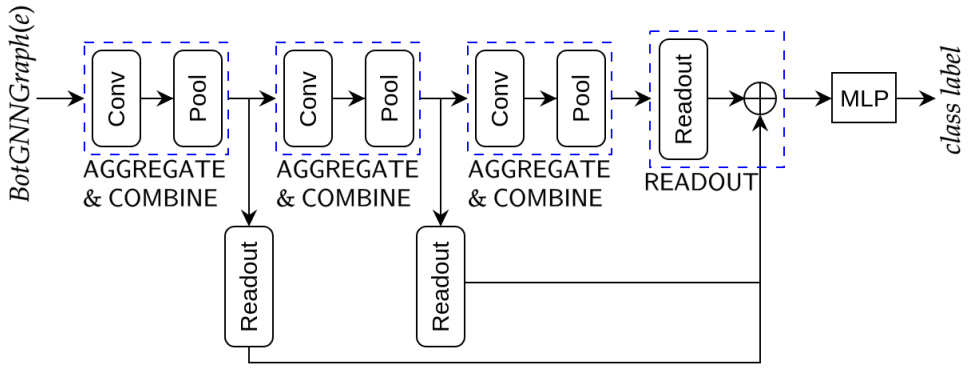


Figure 5.4: Components involved in implementing the workflow in section 4.1 for BotGNN models. 'Conv' and 'Pool' refer to the graph-convolution and graph-pooling operations, respectively. The 'Readout' operation constructs the representation of a graph by accumulating information from all the vertex in the graph obtained after the pooling operation. The final graph-representation is obtained in the READOUT block by an element-wise sum (shown as $\oplus$) of the individual graph representations obtained after each AGGREGATE-COMBINE block. MLP stands for Multilayer Perceptron.

- The graph pooling block uses self-attention pooling [LLK19] with a pooling ratio of 0.5. We use the graph-convolution formula proposed in [KW17] for calculating the self-attention scores.

- Due to the large number of experiments (resulting from multiple datasets and multiple GNN variants), the hyperparameters in the convolution blocks are set to the default values within the PyTorch Geometric library.

- We use a hierarchical pooling architecture that uses the readout mechanism proposed by Cangea *et al.* [CVJ+18]. The readout block aggregates node features to produce a fixed-size intermediate representation for the graph. The final fixed-size representation for the graph is obtained by element-wise addition of the three readout representations.

- The representation length ($2m$) is determined by using a validation-based approach. The parameter grid for m is $\{8, 128\}$, representing a small and a large embedding, respectively.

- The final representation is then fed as input to a 3-layered MLP. We use a dropout layer with a fixed dropout rate of 0.5 after the first layer of MLP.

- The input layer of the MLP contains $2m$ units, followed by two hidden layers with m units and $\lfloor m/2 \rfloor$ units, respectively. The activation function used in the hidden layers is `relu`. The output layer uses `logsoftmax` activation.

- The loss function used is the negative log-likelihood between the target class-labels and the predictions from the model.

- We denote the *BotGNN* variants as $BotGNN_{1,\ldots,5}$, based on the type of graph convolution method used.

- We use the Adam optimiser [KB15] for training the BotGNNs ($BotGNN_{1,\ldots,5}$). The learning rate is 0.0005, weight decay parameter is 0.0001, the momentum factors are set to the default values of $(\beta_1, \beta_2) = (0.9, 0.999)$.

- The maximum number of training epochs is 1000. The batch size is 128.

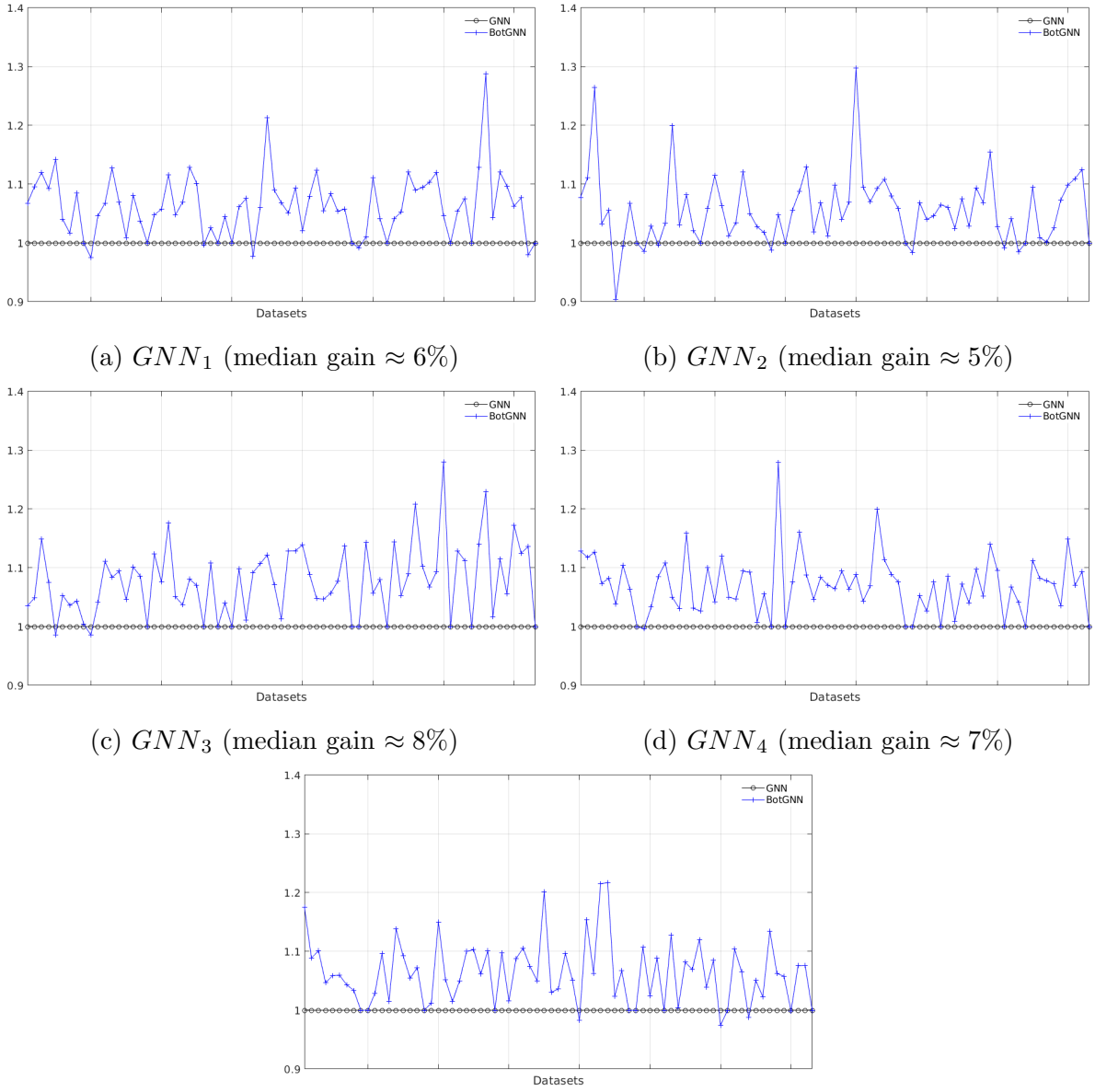
- We use an early-stopping mechanism [Pre98] to avoid overfitting during training. The resulting model is then saved and can be used for evaluation on the independent test-set. The patience period for early-stopping is fixed at 50.

- The predictive performance of a BotGNN model refers to its predictive accuracy on the independent test-set.

- Comparison of the predictive performance of BotGNNs against GNNs and VEGNNs is conducted using the Wilcoxon signed-rank test, using the standard implementation within MATLAB (R2018b).

5.3.4 Results

The quantitative comparisons of predictive performance of *BotGNN*s against baseline *GNN*s are presented in Figure 5.6. The tabulation shows number of datasets on which *BotGNN* has higher, lower or equal predictive accuracy. The principal conclusion from these tabulations is: *BotGNN*s perform significantly better than their corresponding counterparts that do not have access to any information other than the atom-and-bond structure of a molecule achieving a gain in predictive accuracy of 5-8% across variants as shown in the qualitative comparison shown in Figure 5.5. This is irrespective of the variant of GNN used, suggesting that the technique is able to usefully integrate domain-knowledge into learning. In overall, the results here provide sufficient evidence that incorporating domain-knowledge into deep neural networks significantly improves their predictive performance.



(a) GNN_1 (median gain $\approx 6\%$)

(b) GNN_2 (median gain $\approx 5\%$)

(c) GNN_3 (median gain $\approx 8\%$)

(d) GNN_4 (median gain $\approx 7\%$)

(e) GNN_5 (median gain $\approx 6\%$)

Figure 5.5: Qualitative comparison of predictive performance of BotGNNs against Baseline (that is, GNN variants without access to domain-relations). Performance refers to estimates of predictive accuracy (obtained on a holdout set), and all performances are normalised against that of baseline performance (taken as 1). No significance should be attached to the line joining the data points: this is only for visual clarity.

Some Additional Comparisons

We now turn to a question of practical interest: Are $BotGNN$s better than $VEGNN$s that are studied in the preceding chapter? To answer this question, we provide some comparisons $BotGNN$s against $VEGNN$s for the same set of datasets and domain-knowledge. The methodology adopted for this comparison is similar to what was done for comparing $BotGNN$s against GNNs (refer subsection 5.3.3). Figure 5.7 provides a tabulation of this comparison for all the variants of GNNs. The results suggest that

GNN Variant	Accuracy ($BotGNN$ vs. GNN) Higher/Lower/Equal (p-value)
1	59/5/9 (< 0.001)
2	59/8/6 (< 0.001)
3	61/2/10 (< 0.001)
4	63/1/9 (< 0.001)
5	60/4/9 (< 0.001)

Figure 5.6: Comparison of predictive performance of $BotGNN$s against GNNs. The tabulations are the number of datasets on which $BotGNN$ has higher, lower or equal predictive accuracy (obtained on a holdout set) than GNN. Statistical significance is computed by the Wilcoxon signed-rank test.

$BotGNN$s perform significantly better than $VEGNN$s with access to the same background knowledge. This suggests that $BotGNN$s do more than the vertex-enrichment approach used by $VEGNN$s.

GNN Variant	Accuracy ($BotGNN$ vs. $VEGNN$) Higher/Lower/Equal (p-value)
1	54/11/8 (< 0.001)
2	61/9/3 (< 0.001)
3	54/10/9 (< 0.001)
4	55/11/7 (< 0.001)
5	52/9/12 (< 0.001)

Figure 5.7: Comparison of predictive performance of $BotGNN$s against $VEGNN$s. The tabulations are the number of datasets on which $BotGNN$ has higher, lower or equal predictive accuracy (obtained on a holdout set) than a $VEGNN$. Statistical significance is computed by the Wilcoxon signed-rank test.

In a previous section (subsection 5.2.5) we have described differences between BotGNNs and VEGNNs arising from an encoding of the data into a bipartite graph representation. Possible reasons for the significant difference in performance of $BotGNN$s and $VEGNN$s are twofold: (1) The GNN variants are unable to use edge-label information. In the VEGNN-style graphs for the data, this information corresponds to the type of bonds. However, this information is contained in vertices associated with the bond-literals in BotGNN-style graphs, which can be used by the GNN-variants; and (2) The potential loss in relational information in VEGNN-style graphs as described in subsection 5.2.5. A further difference, not apparent from the tabulations, is in the feature-vectors associated with each vertex. For the data here, vertex-labels for VEGNNs described in Chapter 4 results in each vertex being associated with a 1400-dimensional vector. For BotGNNs, this is about 130.

Next, we turn to the second question: Are *BotGNN*s better than propositionalisation? To answer this question, we provide a comparison of the predictive performance of *BotGNN*s against the two different methods studied in Chapter 3, which are: (a) DRMs constructed with relational features sampled using the hide-and-seek sampling, and (b) MLPs constructed relational features constructed using Bottom-Clause Propositionalisation (BCP [FZG14]). For (a), the quantitative comparison between BotGNNs and DRMs is provided in Figure 5.8 with different number of input features for a DRM. At the outset, it may look like DRMs are as good as BotGNNs, if provided with sufficient number of relational features; and there arises one more question: Why should we bother using BotGNNs at all? BotGNNs are more powerful and capable than DRMs for at least the following reasons: (1) DRMs need to be provided with a sufficient number of relational features (In Figure 5.8 this turns out to be 1000) to match the same level of performance as a BotGNN; (2) DRMs need to be provided with an expressive set of relational features to reach the same level of performance as a BotGNN; (3) For a DRM, there is significant computational effort is required to draw these 1000 features using sampling. As discussed in Chapter 3, the sampling procedure incurs a huge computational cost to select a set of 1000 features where selecting just one relational feature requires the following computational steps: sampling a lot more than one feature, evaluating them for their utilities, and discarding the features with bad utilities. Whereas BotGNNs do not involve any such sampling step and therefore the computational cost remains relatively minimal. So our answer to the primary question of whether BotGNNs do more than propositionalisation is a "yes".

GNN Variant	Accuracy (*BotGNN* vs. *DRM*) Higher/Lower/Equal (p-value)			
	No. of features = 50	$\cdots$	No. of features = 500	No. of features = 1000
GNN_1	64/8/1 (< 0.001)	$\cdots$	46/27/0 (0.15)	39/34/0 (0.98)
GNN_2	63/9/1 (< 0.001)	$\cdots$	31/42/0 (0.17)	29/44/0 (0.05)
GNN_3	65/7/1 (< 0.001)	$\cdots$	42/31/0 (0.66)	37/36/0 (0.46)
GNN_4	65/7/1 (< 0.001)	$\cdots$	43/30/0 (0.18)	40/33/0 (0.72)
GNN_5	67/5/1 (< 0.001)	$\cdots$	44/29/0 (0.26)	36/37/0 (0.83)

Figure 5.8: Quantitative comparison of predictive performance of *BotGNN*s against *DRM*s. *DRM* denotes the Deep Relational Machine constructed using propositionalisation of relational features. The relational features for a DRM are sampled using the hide-and-seek sampling strategy proposed in Chapter 3. The comparative performance of BotGNNs against DRMs starts worsening after 1000 features, which are not shown here. The tabulations are the number of datasets on which *BotGNN* has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test.

Our answer is further supported by the quantitative comparison provided in Figure 5.9

where the number of relational features constructed using BCP is approximately 18000-52000 across the 73 datasets, which is far more than the length of the graph representation constructed by a GNN. The results here reaffirm that though propositionalisation based techniques are simple, and they require significant computational overhead to perform well in practice.

GNN Variant	Accuracy (*BotGNN* vs. BCP+MLP) Higher/Lower/Equal (p-value)
1	58/10/5 (< 0.001)
2	58/11/4 (< 0.001)
3	61/6/6 (< 0.001)
4	62/6/5 (< 0.001)
5	60/6/7 (< 0.001)

Figure 5.9: Comparison of predictive performance of BotGNNs with an MLP constructed using BCP-based relational features. The tabulations are the number of datasets on which a *BotGNN* has higher, lower or equal predictive accuracy (obtained on a holdout set) than BCP+MLP.

Further, a more useful difference between the BotGNN approach and propositionalisation is that techniques relying on the latter usually separate the feature- and model-construction steps. A *BotGNN*, like any GNN, constructs a vector-space embedding for the graphs it is provided. However, this embedding is obtained as part of an end-to-end model construction process. This can be substantially more compact than the representation used by methods that employ a separate propositionalisation step (see Figure 5.10).

Method	Vector Representation	Vector Dimension (Range)
BotGNN	Real, dense	16–256
DRM	Boolean, sparse	1000s
BCP+MLP	Boolean, very sparse	18000–52000

Figure 5.10: Characterisation of vector-representation used for model-construction by BotGNNs, DRMs and BCP+MLP. Minimum/maximum values of the range are only shown to 3 meaningful digits (the actual values are not relevant here). The graph-representations (also, called graph-embeddings) for BotGNNs are constructed internally by the GNN. By "sparse" we mean that there are many 0-values, and by "very sparse", we mean the values are mostly 0.

Finally, we turn to a question that we have so far not considered in the dissertation, namely: how well does a deep neural network with domain-knowledge compare against an ILP learner? ILP represents the pre-eminent approach for dealing both with relational data and symbolic domain-knowledge. Given the comparisons we have shown

so far, we will treat *BotGNN*s as the state-of-the-art for direct inclusion of symbolic domain-knowledge into a GNN. Figure 5.11(a) shows a routine comparison of *BotGNN*s against the Aleph system [Sri01], which is probably the most widely-used ILP engine to date [CD20]. We caution against drawing the obvious conclusion, since the results are obtained without attempting to optimise any parameters of the ILP learner (only the minimum accuracy of clauses was changed from the default setting of 1.0 to 0.7: this latter value has been shown to be more appropriate in many previous experimental studies with Aleph). A better indication is in Figure 5.11(b), which compares BotGNN performance on older benchmarks for which ILP results after parameter optimisation are available. These suggest that BotGNN performance to be comparable to an ILP approach with optimised parameter settings.[14]

GNN Variant	Accuracy (*BotGNN* vs. ILP) Higher/Lower/Equal (p-value)
1	62/7/4 (< 0.001)
2	60/9/4 (< 0.001)
3	61/7/5 (< 0.001)
4	62/6/5 (< 0.001)
5	62/4/7 (< 0.001)

(a)

Dataset	ILP	BotGNN
DssTox	0.73	0.76
Mutag	0.88	0.89
Canc	0.58	0.64
Amine	0.80	0.84
Choline	0.77	0.72
Scop	0.67	0.65
Toxic	0.87	0.85

(b)

Figure 5.11: Comparison of predictive performance of BotGNNs with an ILP learner (Aleph system): (a) Without hyperparameter tuning in Aleph; (b) With hyperparameter tuning. In (a), the tabulations are the number of datasets on which *BotGNN* has higher, lower or equal predictive accuracy (obtained on a holdout set) than the ILP learner. In (b), each entry is the average of the accuracy obtained across 10-fold validation splits (as in [SKB03])

5.4 Summary

In this chapter, we proposed a general technique to construct graph neural networks from relational data and symbolic domain-knowledge. Our experiments here re-validate our claim on importance of the role of domain-knowledge. The significant improvements in performance that we have observed support our claim that when training data available are small, deep neural network models can benefit significantly from the inclusion of

[14] We note that parameter screening and optimisation is not routinely done in ILP. In [SR11] it is noted: "Reports in the [ILP] literature rarely contain any discussion of sensitive parameters of the system or their values. Of 100 experimental studies reported in papers presented between 1998 and 2008 to the principal conference in the area, none attempt any form of screening for relevant parameters."

domain-knowledge. We have also provided additional results that suggest that the technique may be doing more than a simple "propositionalisation" or "vertex-enrichment". The linking of symbolic and neural techniques, as is done in this chapter, provides an interesting direction of research to neural-symbolic modelling.

Chapter 6

BotGNN as a System Component: An Application to Drug Design*

So far in the dissertation, we have focussed on developing techniques for including domain-knowledge into deep neural networks, and on investigating–using large scale experiments–if that improves the predictive performance of the deep neural network. For the network types and techniques we propose, we have found that inclusion of domain-knowledge can indeed improve the performance of deep-networks significantly. In this chapter, we show how such a domain-enriched deep neural network can be used as a component in a large engineered system. We focus on a problem in drug-design concerned with the generation of new molecules for potential new drugs. Given the results from previous chapters, we will use BotGNNs as the deep neural network technique with domain-knowledge.

6.1 The Problem

The development of a new drug is difficult, wasteful, costly, uncertain, and long [CHBP02, Hai10]. AI techniques have been trying to change this [SWP+20], especially in the early stages culminating in "lead discovery". Figure 6.1 shows the steps involved in this stage of drug-design. In the figure, library screening can be either done by actual laboratory experiments (high-throughput screening) or computationally (virtual screening). This usually results in many false-positives. Hit Confirmation refers to additional assays designed to reduce false-positives. QSAR (quantitative- or qualitative structure-activity relations) consists of models for predicting biological activity using physico-chemical properties of hits. The results of prediction can result in additional confirmatory assays for hits, and finally in one or more "lead" compounds that are taken forward for pre-clinical

*The content of this chapter is based on the following:
T. Dash, A. Srinivasan, L. Vig, A. Roy, "Using domain-knowledge to assist lead discovery in early-stage drug design", *International Conference on Inductive Logic Programming*, 2021; https://doi.org/10.1007/978-3-030-97454-1_6.

testing. This chapter focuses on the problem of lead discovery that goes beyond the efficient identification of chemicals within the almost unlimited space of potential molecules. This space has been approximately estimated at about 10^{60} molecules. A very small fraction of these have been synthesised in research laboratories and by pharmaceutical companies. An even smaller number are available publicly: the well-known ChEMBL database [GHN+17] of drug-like chemicals consists of about 10^6 molecules. Any early-stage drug-discovery pipeline that restricts itself to in-house chemicals will clearly be self-limiting. This is especially the case if the leads sought are for targets in new diseases, for which very few "hits" may result from existing chemical libraries. While a complete (but not exhaustive) exploration of the space of 10^{60} molecules may continue to be elusive, we would nevertheless like to develop an effective way of sampling from this space.

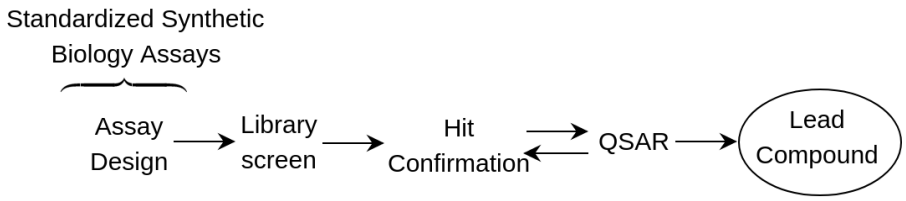


Figure 6.1: Early-stage drug-design (adapted from [WBS+15]).

We would like to implement the QSAR module as a generator of new molecules, conditioned on the information provided by the hit assays, and on domain-knowledge. Our position is that inclusion of domain-knowledge allows the development of more effective conditional distributions than is possible using just the hit assays. Figure 6.2 is a diagrammatic representation of an ideal conditional generator of the kind we require. The difficulty of course is that none of the underlying distributions are known. In this chapter, we describe a neural-symbolic implementation to construct approximations for the distributions.

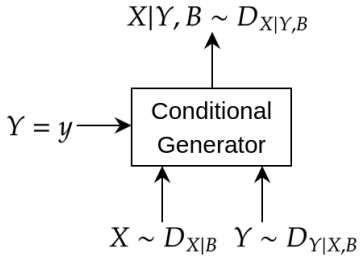


Figure 6.2: An ideal conditional generator for instances of a random-variable denoting data (X) given a value for a random-variable denoting labels (Y) and domain-knowledge (B). Here, $Z \sim D$ denotes a random variable Z is distributed according to the distribution D. If the distributions shown are known, then a value for X is obtainable through the use of Bayes rule, either exactly or through some form approximate inference.

We are interested in generating new small molecules which could act as inhibitors of a biological target, when there is limited prior information on target-specific inhibitors.

This form of drug-design is assuming increasing importance with the advent of new disease threats for which known chemicals only provide limited information about target inhibition [MR19, PIT18]. The initial studies were focused on exploring vast yet unexplored chemical space for a better screening library. In [SKTW17], a recurrent neural network (RNN) based generative model was trained with a large set of molecules and then fine-tuned with small sets of molecules, which are known to be active against the target. Some other works focus on drug-like property optimization, which helped in biasing the models to generate molecules with specific biological or physical properties of interest. Deep reinforcement learning has been very effective in constructing generative models that could generate novel molecules with the target properties [PIT18, BMO+21, SFK+19]. The efficiency of these kinds of models to generate chemically valid molecules with optimized properties has improved significantly [KBBR21, BMO+21]. There are also attempts to build molecule generation models against novel target proteins, where there is a limited ligand dataset for training the model [BKBR21]. Recurrent Neural Networks (RNNs) are a popular choice for molecule generation. For example, [GMLS20] propose a bidirectional generative RNN, that learns SMILES strings in both directions allowing it to better approximate the data distribution. Attention-based sequence models such as transformers have recently been used for protein-specific molecule generation [Gre21]. There are also generative models, for instance, masked graph modelling in [MMBC21], that attempts to learn a distribution over molecular graphs allowing it to generate novel molecule without requiring to deal with sequences. Although the works referred here are shown to be effective, they are often purely deep neural network-based. It is well-known that such deep generative models are data-hungry, and require a large-amount of data (100s of 1000s) of data instances to be able to generate novel data-instances. Furthermore, there has been very little or no attempt in incorporating domain-knowledge into deep neural networks for molecule generation.

In this chapter we investigate the construction of a deep generative model for molecules that can utilise available domain-knowledge. The overall contributions of this chapter are the following: (1) We propose a system consisting of two deep generative models (or generators) and a discriminative model (or discriminator) working in collaboration with each other for conditional generation of novel molecules; (2) We propose a methodology to allow the generators to access the available domain-knowledge for molecule generation; (3) We investigate our approach using the well-studied problem of inhibitors for the Janus kinase (JAK) class of proteins [Wil89]. We assume first that if no data on inhibitors are available for a target protein (JAK2), but a small number of inhibitors are known for homologous proteins (JAK1, JAK3 and TYK2); and (4) We show that the inclusion of relational domain-knowledge results in a potentially more effective generator of inhibitors than simple random sampling from the space of molecules or a generator without access to symbolic relations. We also show how samples from the conditional generator can be

used to identify potentially novel target inhibitors.

6.2 System Design and Implementation

We implement an approximation to the ideal conditional generator using a combination of two generators and a discriminator (see Figure 6.3). We have decomposed the domain-knowledge B in Figure 6.2 into constraints relevant just to the molecule-generator B_G and the knowledge relevant to the prediction of activity B_D (that is, $B = B_G \cup B_D$, and $P(X|B) = P(X|B_G)$ and $P(Y|X,B) = P(Y|X,B_D)$). The discriminator module approximates the conditional distribution $D_{Y|X,B}$, and the combination of the unconditional generator and filter approximates the distribution $D_{X|B}$. The conditional generator then constructs an approximation to $D_{X|Y,B}$. For the present, we assume the unconditional generator and discriminator are pre-trained: details will be provided below. The discriminator is a BotGNN (refer Chapter 5). This is a form of graph-based neural network (GNN) that uses graph-encodings of most-specific clauses (see [Mug95]) constructed using symbolic domain-knowledge B_D.

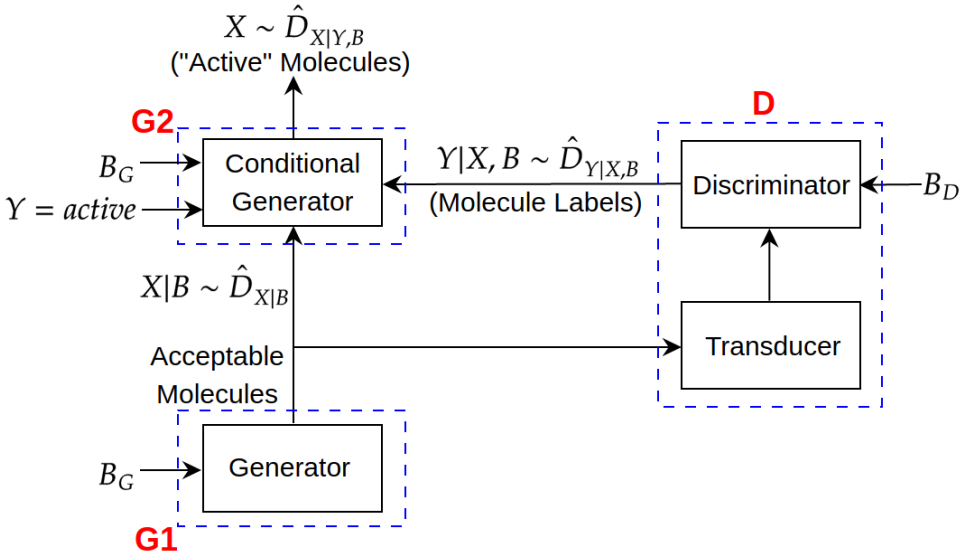


Figure 6.3: Training a conditional generator for generating "active" molecules. For the present, we assume the generator (G1) and discriminator (D) have already been trained (the G1 and D modules generate acceptable molecules and their labels respectively: the $\hat{D}$'s are approximations to the corresponding true distribution). The Transducer converts the output of G1 into a form suitable for the discriminator. Actual implementations used in the chapter will be described below.

The generator-discriminator combination in Figure 6.3 constitutes the QSAR module in Figure 6.1. An initial set of hits is used to train the discriminator. The conditional generator is trained using the initial set of hits and the filtered samples from the unconditional generator and the labels from the discriminator. Although out of the scope

of this chapter, any novel molecules generated could then be synthesised, subject to hit confirmation, and the process repeated.

6.2.1 Generating Acceptable Molecules

The intent of module G1 is to produce an approximation to drawing samples (in our case, molecules) from $D_{X|B_G}$. We describe the actual B_G used for experiments in subsection 6.3.1. For the present it is sufficient to assume that for any instance $X = x$, if $B_G \wedge X = x \models \square$ then $P(x|B_G) = 0$. Here, we implement this by a simple rejection-sampler that first draws from some distribution over molecules and rejects the instances that are inconsistent with B_G.

For drawing samples of molecules, we adopt the text-generation model proposed in [BVV+16]. Our model takes SMILES representations [Wei88] of molecules as inputs and estimates a probability distribution over these SMILES representations. Samples of molecules are then SMILES strings drawn from this distribution.

The SMILES generation module is shown in Figure 6.4. The distribution of molecules (SMILES strings) is estimated using a variational autoencoder (VAE) model. The VAE model consists of an encoder and a decoder, both based on LSTM-based RNNs [HS97]. This architecture forms a SMILES encoder with the Gaussian prior acting as a regulariser on the latent representation. The decoder is a special RNN model that is conditioned on the latent representation.

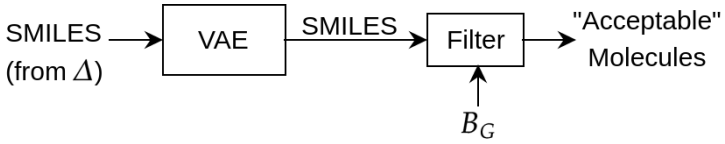


Figure 6.4: Training a generator for acceptable molecules. Training data consists of molecules, represented as SMILES strings, drawn from a database Δ. The VAE is a model constructed using the training data and generates molecules represented by SMILES strings. B_G denotes domain-knowledge consisting of constraints on acceptable molecules. The filter acts as a rejection-sampler: only molecules consistent with B_G pass through.

The architecture of the VAE model is shown in Figure 6.5. The SMILES encoding involves three primary modules: (a) embedding module: It constructs an embedding for the input SMILE; (b) highway module: It constructs a gated information-flow module based on highway network [SGS15]; (c) LSTM module: It is responsible for dealing with sequences. The modules (b) and (c) together form the encoder module. The parameters of the Gaussian distribution is learnt via two fully-connected networks, one each for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, which are standard sub-structures involved in a VAE model. The decoder module consists of LSTM layers followed by a fully-connected (FC) layer. We defer the details on architecture-specific hyperparameters to subsection 6.3.2. The loss function used for

training our VAE model is a weighted version of the reconstruction loss and the KL-divergence between VAE-constructed distribution and the Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{1})$.

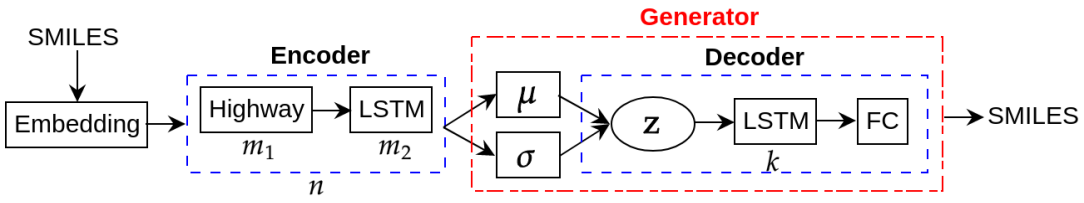


Figure 6.5: Architecture of the VAE in Figure 6.4. m_1, m_2, n, k denote the number of blocks. The decoder along with the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ constitute the generator that generates molecules in SMILES representation.

6.2.2 Obtaining Labels for Acceptable Molecules

The intent of module D is to produce an approximation to draw samples (in our case, labels for molecules) from $D_{Y|X,B_D}$. We describe the actual B_D used for experiments in subsection 6.3.1. The discriminator in D is a BotGNN (see, Chapter 5), which is a form of graph neural network (GNN) constructed from data (as graphs) and background knowledge (as symbolic relations or propositions) using mode-directed inverse entailment (MDIE [Mug95]). In this work, data consists of graph-based representations of molecules (atoms and bonds), and B_D consists of symbolic domain-relations applicable to the molecules. The goal of the discriminator is to learn a distribution over class-labels for any given molecules. Figure 6.6 shows the block diagram of the discriminator block.

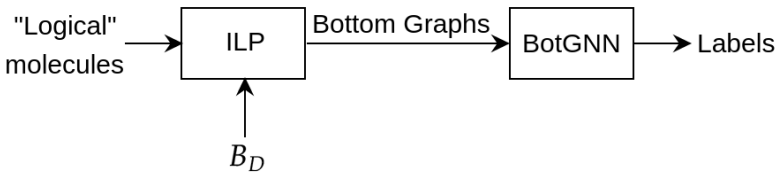


Figure 6.6: Discriminator based on BotGNN. "Logical" molecules refers to a logic-based representation of molecules. Bottom-graphs are a graph-based representation of most-specific ("bottom") clauses constructed for the molecules by an ILP implementation based on mode-directed inverse entailment.

6.2.3 Generating Active Molecules

The intent of module G2 is to produce an approximation to drawing from $D_{X|Y,B}$. That is, we want to draw samples of molecules, given a label for the molecule and domain-knowledge B. We adopt the same architecture as the generator used for drawing from $D_{X|B_G}$ above, with a simple modification to the way the SMILES strings are provided as inputs to the model. We prefix each SMILES string with a class-label: $y = 1$ or

$y = 0$ based on whether the molecule is an active or inactive inhibitor, respectively. The VAE model is also able to accommodate any data that may already be present about the target, or about related targets (it is assumed that such data will be in the form of labelled SMILES strings).

6.3 System Testing

Our aim is to perform a controlled experiment to assess the effect on system performance of the inclusion of high-level symbolic domain-knowledge. Specifically, we investigate the effect on the generation of new inhibitors for the target when (a) no domain-knowledge is available in the form of symbolic relations (but some knowledge is available in a propositional form); and (b) some domain-knowledge is available in the form of symbolic relations. We intend to test if the system is able to generate possible new inhibitors in case (a); and if the performance of the system improves in case (b).

6.3.1 Materials

Data

The data used are as follows: (a) ChEMBL dataset [GHN+17]: 1.9 million molecules in SMILES representation; used to train the generator for legal molecules (G1); (b) JAK2 [KBBR21]: 4100 molecules (3700 active) in SMILES representation; used to test the conditional generator (G1) and to build the proxy model for hit confirmation (see Method section below); (c) JAK2 Homologues (JAK1, JAK3 and TYK2) [KBBR21]: 4300 molecules (3700 active) in SMILES representation; used to train the discriminator (D) and train the conditional generator (G2).

Domain-Knowledge

The domain constraints in B_G are in the form of constraints on acceptable molecules. These constraints are broadly of two kinds: (i) Those concerned with the validity of a generated SMILES string. This involves various syntax-level checks, and is done here by the RDKit molecular modelling package; (ii) Problem-specific constraints on some bulk-properties of the molecule. These are: molecular weight is in the range $(200, 700)$, the octanol-water partition coefficients (logP) must be below 6.0, and the synthetic accessibility score (SAS) must be below 5.0. We use the scoring approach proposed in [ES09] to compute the SAS of a molecule based on its SMILES representation.

The domain-knowledge in B_D broadly divides into two kinds: (i) Propositional, consisting of the following bulk-molecular properties: molecular weight, logP, SAS, number of hydrogen bond donors (HBD), number of hydrogen bond acceptor (HBA), number of

rotatable bonds (NRB), number of aromatic rings (NumRings), Topological Polar Surface Area (TPSA), and quantitative estimation of drug-likeness (QED); (ii) Relational, which is a collection of logic programs (written in Prolog) defining almost 100 relations for various functional groups (such as amide, amine, ether, etc.) and various ring structures (such as aromatic, non-aromatic, etc.). More details on this background knowledge can be found in Chapter 3.

Algorithms and Machines

We use the following softwares for our system implementation: (a) RDKit [L+06]: Molecular modelling software used to compute molecular properties and check for the validity of molecules; (b) Chemprop [SYS+20]: Molecular modelling software used to build a proxy model for hit confirmation; (c) Transducer: In-house software to convert representation from SMILES to logic; (d) Aleph [Sri01]: ILP engine used to generate most-specific clauses for BotGNN; (e) BotGNN: Discriminator for acceptable molecules capable of using relational and propositional domain-knowledge; (f) VAE [KW14]: Generative deep neural network used for generators. We used PyTorch for the implementation of BotGNN and VAE models, and Aleph was used with YAP.

Our experimental works were distributed across two machines: (a) The discriminator (D) was built on a Dell workstation with 64GB of main memory, 16-core Intel Xeon 3.10GHz processors, an 8GB NVIDIA P4000 graphics processor; (b) The generators (G1, G2) are built on an NVIDIA-DGX1 station with 32GB Tesla V100 GPUs, 512GB main memory, 80-core Intel Xeon 2.20GHz processors.

6.3.2 Method

We describe the procedure adopted for a controlled experiment comparing system performance in generating potential inhibitors when (a) domain-knowledge is restricted to commonly used bulk-properties about the molecules; and (b) domain-knowledge includes information about higher-level symbolic relations consisting of ring-structures and functional groups, along with the information in (a). In either case, the method used to generate acceptable molecules (from module G1 in Figure 6.3) is the same.

Let B_0 denote domain-knowledge consisting of bulk-molecular properties used in the construction of QSARs for novel inhibitors, B_1 denote the definitions in B_0 along with first-order relations defining ring-structures and functional-groups used in the construction of QSAR relations, and B_G denote the domain-knowledge consisting of constraints on acceptable molecules (see "Domain-Knowledge" in subsection 6.3.1). Let D_{Tr} denote the data available on inhibitors for JAK1, JAK3 and TYK2; and D_{Te} denote the data available on inhibitors for JAK2 (see "Data" in subsection 6.3.1). Let Δ denote a database of (known) legal molecules. Our methodology is straightforward.

(1) Construct a generator for possible molecules given Δ (the generator in module G1 of Figure 6.3).

(2) For $i = 0, 1$

(a) Let $E_0 = \{(x, y)\}_1^{|D_{Tr}|}$, where x is a molecule in D_{Tr} and y is the activity label obtained based on a threshold θ on the minimum activity for active inhibition;

(b) Let $B_D = B_i$;

(c) Construct a discriminator (for module D in Figure 6.3) using E_0 and the domain-knowledge B_D (see section 6.2);

(d) Sample a set of possible molecules, denoted as N, from the generator constructed in Step (1);.

(e) Let $N' \subseteq N$ be the set of molecules found to be acceptable given the constraints in B_G (that is, N' is a sample from $\hat{D}_{X|B_G}$);

(f) For each acceptable molecule x obtained in Step (2)d above, let y be the label with the highest probability from the distribution $\hat{D}_{Y|X,B}$ constructed by the discriminator in Step (2)c. Let $E = \{(x, y)\}_1^{|N'|}$;

(g) Construct the generator model (for module G2 in Figure 6.3) using $E_0 \cup E$;

(h) Sample a set of molecules, denoted as M_i, from the generator in Step (2)g;

(i) Let $M_i' \subseteq M_i$ be the set of molecules found to be acceptable given the constraints in B_G (that is, M_i' is a sample from $D_{Y|X,B_G}$);

(3) Assess the samples M_0, M_1 obtained in Step (2)h above for possible new inhibitors of the target, using the information in D_{Te}.

The following details are relevant:

- For experiments here Δ is the ChEMBL database, consisting of approximately 1.9 million molecules. The generator also includes legality checks performed by the RDKit package, as described in section 6.2.

- Following [KBBR21], $\theta = 6.0$. That is, all molecules with pIC50 value ≥ 6.0 are taken as "active" inhibitors;

- The discriminator in Step (2)c is a BotGNN. We follow the procedure and parameters described in Chapter 5 to construct BotGNN. We use GraphSAGE [HYL17] for the convolution block in the GNN (Refers to variant 4 in Chapter 5). This is based on the results shown in Chapter 5 for the inclusion of symbolic domain-knowledge for graph-based data, such as molecules.

- The generators in Steps (1) and (2)g are based on the VAE model described earlier. The hyperparameters are as follows: vocabulary length is 100, embedding-dimension is 300, number of highway layers is 2, number of LSTM layers in the encoder is 1 with hidden size 512, and the type is bidirectional, number of LSTM layers in the decoder is 2, each with hidden size 512, dimension of the latent representation ($\mathbf{z}$) is 100.

- To make our generator robust to noise and to be generalised, we also use a word-dropout technique. This technique is identical to the standard practice of dropout in deep learning [SHK+14] except that here the tokens to the decoder are replaced by '*unknown*' tokens with certain probabilities. Here we call it the word-dropout rate and fix it at 0.5.

- The reconstruction loss coefficient is 7. We use cost-annealing [BVV+16] for the KLD-coefficient during training. We use the Adam optimizer [KB15] with learning rate of 0.0001; training batch-size is 256.

- In Step (2)d, $|N| = 30,000$. The B_G provided here results in $|N'| = 18,000$;

- In Step (2)h, $|M_0| = |M_1| = 5000$.

- The acceptable molecules M_0', M_1' after testing for consistency with B_G are assessed along the following two dimensions:

  (a) *Activity*: In the pipeline described in Figure 6.1 assessment of activity would be done by *in vitro* by hit confirmation assays. Here we use a proxy assessment for the result of the assays by using an *in silico* predictor of pIC50 values constructed from the data in D_{Te} on JAK2 inhibitors. The proxy model is constructed by a state-of-the-art activity prediction package (Chemprop [SYS+20]: details of this are provided in section B.3).[1] We are interested in comparing the proportions of generated molecules predicted as "active";

  (b) *Similarity*: we want to assess how similar the molecules generated are to the set of active JAK2 inhibitors in D_{Te}.[2] A widely used measure for this is the Tanimoto (Jaccard) similarity: molecules with Tanimoto similarity > 0.75 are usually taken to be similar. We are interested in the proportion of molecules generated that are similar to known target inhibitors in D_{Te};

[1] Such a model is only possible in the controlled experiment here. In practice, no inhibitors would be available for the target and activity values would have to be obtained by hit assays, or perhaps *in silico* docking calculations.

[2] Again, this is feasible in the controlled experiment here. In practice, we will have no inhibitors for the target, and we will have to perform this assessment on the data available for the target's homologues (D_{Tr}).

Each sample of molecules M_i drawn from the conditional generator can therefore be represented by a pair (a_i, b_i) denoting the values of the proportions in (a) and (b), and (c) above. We will call this pair the "performance summary" of the set M_i.

- We compare performance summaries of sets of molecules in two ways. First, a performance summary $P_i = (a_i, b_i)$ can be compared against the performance summary $P_j = (a_j, b_j)$ in the obvious lexicographic manner. That is, P_i is better than P_j if $[(a_i > a_j)]$ or $[(a_i = a_j) \wedge (b_i > b_j)]$. Secondly, since all the elements of a performance summary are proportions, we are able to assess if the differences in corresponding values are statistically significant. This is done using a straightforward hypothesis test on proportions. Given an estimate p of a proportion of n instances, the distribution of proportions is approximately Normal, with mean p and s.d. $\sigma = \sqrt{\frac{p(1-p)}{n}}$. For testing the hypothesis $p_j < p_i$ at a 95% confidence level the critical value from tables of the standard normal distribution is 1.65. That is, if $p_j < 1.65\sigma$ we will say the difference is statistically significant at the 95% level of confidence.

6.3.3 Results

A summary of the main results obtained is in Figure 6.7. The principal points in this tabulation are these: (1) The performance of the system with $B_D = B_1$ is better than with $B_D = B_0$ or simple random draw of molecules; and (2) The differences in proportions for Activity and Similarity are statistically significant at the 95% confidence level. Taken together, these results suggest that the inclusion of symbolic relations can make a significant difference to the performance of the generation of active molecules.

We turn next to some questions of relevance to these results:

Better Discriminators? A question arises on whether the differences in proportions would be different if we had compared against a different discriminator capable of using $B_D = B_0$. Since B_0 is essentially propositional in nature, any of the usual statistical discriminative approaches could be used. We have found replacing the BotGNN with an MLP with hyper-parameter tuning resulted in significantly worse performance than a BotGNN with $B_D = B_1$. We conjecture that similar results will be obtained with other kinds of statistical models. On the question of whether better discriminators are possible for $B_D = B_1$, we note results in Chapter 5 show BotGNNs performance to be better than techniques based on propositionalisation or a direct use of ILP. Nevertheless, better BotGNN models than the one used here may be possible. For example, we could construct an activity prediction model for the JAK2 homologues using a state-of-the-art predictor like Chemprop. The prediction of this model could be used as an additional molecular property by the BotGNN.

Better Generators? Our generators are simple language models based on variational

Qty.	$B_D = B_1$	$B_D = B_0$	*Random*
$\|M\|$	5000	5000	5000
$\|M'\|$	2058	2160	2877
Act	0.47 (0.01)	0.43 (0.01)	0.34 (0.01)
Sim	0.14 (0.01)	0.11 (0.01)	0.00 (0.00)

Figure 6.7: Summary of system performance. $B_D = B_1$ denotes that the discriminator has access to both propositional and relational domain-knowledge; $B_D = B_0$ denotes that the discriminator has access to propositional domain-knowledge only. *Random* denotes a random draw of molecules from the unconditional molecule generator G1. M denotes the set of molecules drawn (from the conditional generator, or from the unconditional generator for *Random*). The results are compared against the performance of a methodology purely based on Deep Reinforcement Learning [KBBR21]. M' denotes the set of acceptable molecules generated in the sample of M molecules (acceptable molecules satisfy molecular constraints defined on molecular properties). *Act* denotes the proportion of M' that are predicted active (the proxy model predicts an pIC50 ≥ 6.0); *Sim* denotes the proportion of M' that are similar to active target inhibitors (Tanimoto similarity to active JAK2 inhibitors > 0.75). The numbers in parentheses denote the standard deviation in the corresponding estimate.

auto-encoders. Substantial improvements in generative language models (for example, the sequence models based on attention mechanism [DCLT19, RWC+19]) suggest that the generator could be much better. In addition, the rejection-sampling approach we use to discard sample instances that fail constraints in B_G is inherently inefficient, and we suggest that the results here should be treated as a baseline. The modular design of our system-design should allow relatively easy testing of alternatives.

Related to the question of discriminators is the role of ILP in this work. ILP is used to include domain-knowledge in the construction of the BotGNN discriminator. How important was this use of ILP? A quantitative answer is difficult, but we are able to provide indirect, qualitative evidence for the utility of ILP by comparison against a recent result on the same data in [KBBR21]. That work differs from the one here in the following ways: (a) No symbolic domain-knowledge is used in the discrimination step; and (b) Substantially more computation is involved in developing the final generator–the equivalent of module G2 here–through the use of reinforcement learning (RL). The principal concern in [KBBR21] is to generate molecules similar to the active inhibitors for JAK2, and the approach results in 5% of the sampled molecules being similar. The corresponding results here are significantly higher: 14% (with $B_D = B_1$) and 11% ($B_D = B_0$). Both results were obtained with BotGNNs, without requiring the additional episodic training characteristic of RL. Therefore, we believe BotGNNs have played an important role, both in prediction and in easing computation. Since ILP is necessary for the construction of a

BotGNN, their importance to the current system-design follows.[3]

Finally, we consider how samples from the conditional generator can be used to identify potential molecules for synthesis and testing in hit-confirmation assays. We propose a selection based on a combination of (predicted) activity and similarity to the existing inhibitors (when these are unavailable, we would have to rely on models constructed with the target's homologues). Using these measures, there are two surprising subsets of molecules. Molecules in S are those that are similar to JAK2 inhibitors (Tanimoto similarity > 0.75), but have a low predicted activity (substantially lower than 6.0); and molecules in $\overline{S}$ are significantly different to the JAK2 inhibitors (Tanimoto similarity < 0.5), but have a high predicted activity (substantially higher that 6.0).[4] For the sample in this chapter, $S = \emptyset$. However, $\overline{S} \neq \emptyset$ and can provide interesting candidates for novel inhibitors. We exemplify this with a chemical assessment of 3 elements from $\overline{S}$. This is shown in Figure 6.8. Molecule 1562 is identified as a possible candidate for synthesis and hit confirmation.

6.4 Summary

In this chapter, we have shown how a deep neural network capable of including domain-knowledge can be used as part of a larger system designed for a purpose other than simple discrimination. As an example, we have considered the design of a system generating novel molecules for early-stage drug design. We show how a graph-based neural network with domain-knowledge is one component in a modular system design. Specifically, our conclusions from this chapter are as follows: (1) We have constructed a complete end-to-end neural-symbolic system that is capable of generating active molecules that may not be in any existing database; (2) We have demonstrated usage of the system on the classic chemical problem on Janus kinase inhibitors. Importantly, working with a computational chemist, we have shown how the system can be used to discover an active molecule based on entirely new scaffolds; (3) The results reaffirm the conclusions from our Chapter 5 that inclusion of relational domain-knowledge through the use of ILP techniques can significantly improve the performance of deep neural networks. Further, our system design is intentionally modular, to allow "plug-and-play" of discriminators and generators, allowing faster experimentation with better modules.

[3] Could we have directly used ILP for constructing the discriminator? Yes, but we draw attention to evidence from the previous chapter that suggests that BotGNNs result in discriminators that are at least comparable, and possibly better than those from the direct use of ILP.

[4] A good reason to consider dissimilar molecules is that it allows us to explore more diverse molecules.

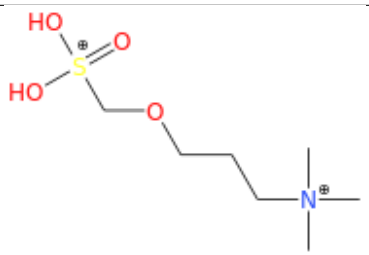
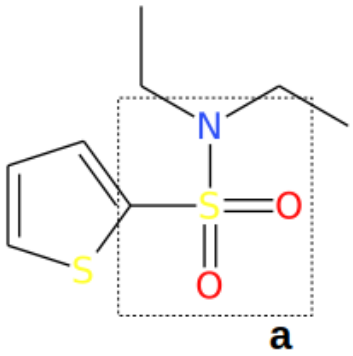
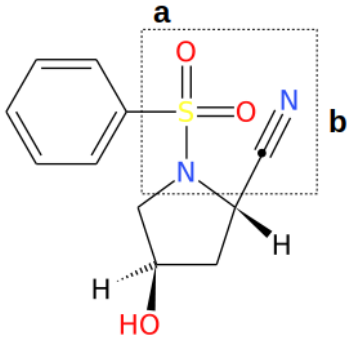
ID	Structure	Descriptors	Assessment
551		$Act = 9.12$ $Sim = 0.15$	This molecule has very low similarity to known JAK2 inhibitors. Also none of the groups specific to JAK2 could be identified by the substructure search. Discard this molecule.
1548		$Act = 9.04$ $Sim = 0.22$	This molecule has very low similarity to known JAK2 inhibitors. Also none of the groups specific to JAK2 could be identified by the substructure search. However, the sulfonamide group commonly found in JAK family inhibitors was found to be present (highlighted).
1562		$Act = 9.49$ $Sim = 0.32$	Despite low similarity to existing JAK2 inhibitors, 1562 had one JAK2-selective subgroup and a group common to JAK inhibitors, indicating potential to act as JAK family inhibitor, but the selectivity to JAK2 cannot be confirmed. Possibly interesting new scaffold (highlighted) and worth pursuing further.

Figure 6.8: A chemical assessment of possible new JAK2 inhibitors. The molecules are from the sample of molecules from the conditional generator, that are predicted to have high JAK2 activity, and are significantly dissimilar to known inhibitors. The assessment is done by a computational chemist[†]. The assessment uses structural features and functional groups identified for the JAK2 site in the literature [KBBR21, DS13, DYCFY14].

[†]Dr. Arijit Roy, TCS Innovation Labs, Hyderabad.

Chapter 7

Conclusions and Future Work

Let us return again to the "Domain-Knowledge Grand Challenge" [STN+20], first introduced in Chapter 1 as follows:

> "ML and AI are generally domain-agnostic... Off-the-shelf practice treats [each of these] datasets in the same way and ignores domain knowledge that extends far beyond the raw data itself—such as physical laws, available forward simulations, and established invariances and symmetries—that is readily available... Improving our ability to systematically incorporate diverse forms of domain knowledge can impact every aspect of AI ..."

This dissertation has been about addressing the challenge of inclusion of complex symbolic domain-knowledge into some kinds of deep neural networks to analyse relational data. The domain-knowledge in all cases "extends the data" used by the deep neural networks, either by non-uniform sampling of relational features; or by inclusion of relational information in a simplified form; or by inclusion of all the relational information through the use of techniques developed in Inductive Logic Programming (ILP). In all cases, we find empirical evidence supporting the principal conjecture investigated in the dissertation, namely:

- Inclusion of domain-knowledge can significantly improve the performance of a deep neural network.

Here we reiterate the main contributions and findings, and then sketch a broad outline of future research directions.

7.1 Summary of the Dissertation

7.1.1 The Main Contributions

The principal contributions made in this dissertation are as follows:

Concepts. Hide-and-seek sampling for relational features; a simplified technique for treating domain-relations as hyperedges for inclusion in graph-based neural networks (GNNs); a general technique for the inclusion of relational domain-knowledge into GNNs through the use of mode-directed inverse entailment;

Implementations. Techniques that combine neural networks and symbolic representations resulting in Deep Relational Machines (DRMs), Vertex-Enriched Graph Neural Networks (VEGNNs), Bottom-Graph Neural Networks (BotGNNs); and a modular end-to-end neuro-symbolic system for generation of novel molecules for drug-design; and

Applications. Large-scale empirical testing, using nearly 75 datasets in the broad area of drug discovery and with domain-knowledge containing nearly 100 relations and over 200,000 relational data instances; and large-scale generation of molecules outside the space of compounds in known chemical databases.

7.1.2 The Main Findings

Some of the findings obtained during the course of these contributions are as follows:

DRMs. DRMs equipped with hide-and-seek sampling of relational features are simple, yet powerful. The predictive performance of a DRM increases with increase in number of relational features. However, constructing a reasonably powerful predictive model with DRM can be computationally expensive, often requiring a large number of logically expressive relational features.

VEGNNs. VEGNNs learn effectively using domain-relations represented as hyperedges. However, the vertex-enrichment technique results in loss of domain information, that is, enriched vertex labels do not convey information such as a vertex is a member of two different hyperedges of the same type. Therefore, this technique results simplified inclusion of domain-knowledge into GNNs.

BotGNNs. BotGNN is a general technique for complete inclusion of relational information into GNNs. BotGNNs are better than BCP-based MLPs. DRMs with hide-and-seek sampling can perform better than BotGNNs, but only with significant computational effort (several 1000s of input features, which can require sampling and evaluating many more features). BotGNNs are better than VEGNNs in terms of predictive performance and the amount of domain-information they can include into a GNN. BotGNNs appear to be at least as good as optimised ILP and probably better if no parameter optimisation is performed for the ILP engine (as is often the case in practice).

Novel Molecule Generation. A molecule generation system that has two deep generative models and a BotGNN acting as discriminator is found to benefit from the inclusion of symbolic domain-knowledge. The system is able to generate a diverse set of molecules, with novel scaffolds that can act as inhibitors for a well-studied target protein.

7.2 Challenges and Future Work

It is possible to consider representing domain-knowledge not as logical or numeric constraints, but through statements in natural language. Recent rapid progress in the area of language models, for example, the models based on attention [VSP+17, BMR+20] raises the possibility of incorporating domain-knowledge through conversations. While precision of these formal representations may continue to be needed for the purpose of construction of deep models with domain-knowledge, the flexibility of natural language may be especially useful in communicating commonsense knowledge to day-to-day machine assistants that need an informal knowledge of the world [TVdM18, ZKK+21]. Progress in this is being made (see, for example, https://allenai.org/aristo), but there is much more that needs to be done to make the language models required accessible to everyday machinery.

In this dissertation, we have not been concerned with how the domain-knowledge to be acquired from domain experts. Instead, our focus has been on the methods of their inclusion into DNNs. The work has further been restricted to inclusion by changing the input data. As we described in Chapter 2, domain-knowledge can also be about the loss function, the structure or parameters of the deep neural network. How any of these forms of domain-knowledge are to be acquired remains an important question to be addressed.

As this dissertation is being written, the field of deep neural networks is proceeding at a rapid pace. We have developed principled ways in which domain-knowledge can be included in just two kinds of neural networks, albeit with wide applicability (MLPs and GNNs). Do these same techniques apply to other kinds of DNNs and will the results be similarly positive? This remains to be studied.

The empirical results in the dissertation are all from molecular datasets, which we have used as classic representatives of relational data. However there are clearly many other problems where data are much more diverse in their relational structure; and where the tasks may not be "object-centred", but may involve relations between multiple relational objects (link-prediction tasks are an example). BotGNNs are not restricted to object-centred relations, since the graph it constructs is a representation of relations that hold between any number of objects. The power of BotGNNs as a deep neural-symbolic network has thus not been fully explored by the experimental work here.

Finally, we note that this dissertation has been focussed entirely on using domain-

knowledge to improve the predictive performance of a deep neural network. Developing a mapping of internal representations of the deep-network's model to concepts provided as domain-concepts will be necessary for acceptable explanations for the model's predictions. One route for developing trust in the model comes through understanding of how decisions are made by the model, and what are the determining factors in these decisions. An important requirement of machine-models in workflows with humans-in-the-loop is that the models are human-understandable. Domain-knowledge can be used in two different ways to assist this. First, it can constrain the kinds of models that are deemed understandable. Secondly, it can provide concepts that are meaningful for use in a model. The role of domain-knowledge in constructing explanations for deep neural network models has not been addressed in this dissertation. However, the development of explanatory deep neural network models that identify true causal connections based on concepts provided as domain-knowledge remains an open question.

7.3 Closing Remarks

The overarching position in this dissertation is "Domain-Knowledge Matters". This position apparently runs against a school of machine-learning that takes the view that all necessary concepts can be discovered automatically from low-level data. However, this is not a true reflection of our position. It is indeed important for a machine-learning approach to be able to discover new concepts, especially if it is used as a part of an autonomous agent. However, for the use of machine-learning systems as tools for decision-support, our position is that it is inefficient, and altogether unfair, not to provide a machine-learning system all the information that may be relevant to the construction of good models. In this dissertation we have sought to develop concepts and implementations that allow us to explore this position more fully by combining the expressiveness of logical forms domain-knowledge with the predictive power of deep neural networks. We hope the results obtained provide encouragement to the development of neural-symbolic machine-learning for prediction and explanation.

Appendix A

Background

This appendix presents some background for the research presented in this dissertation. We first elaborate on the conceptual details of some standard deep neural network architectures that are extensively used in our research. Then, we provide some brief conceptual details on Inductive Logic Programming (ILP).

A.1 Deep Neural Networks

Deep neural networks (or DNNs) are artificial neural networks that consists of multiple layers between an input layer (that takes features describing a data-instance as input) and an output (that computes the prediction given the input). A DNN is defined by a structure and a set of parameters. The structure of a DNN refers to the organisation of various layers in the DNN and the interconnections among various layers. Each connection in a DNN associates a connection strength (a real-value) called the synaptic weight or weight. There are also weights of a different kind called biases. The set of all the weights and biases is called the parameters of a DNN. DNNs can model complex non-linear relationships in the data by expressing a data-instance as a layered composition of primitive features [STE13]. The hidden layers in a DNN are responsible for constructing a transformed representation for the data-instance suitable for the given problem.

Figure A.1 shows a diagrammatic representation of the brief process of learning a DNN model given data. A learner $\mathcal{L}$ takes as input the data D, a DNN model structure π relevant for learning D, the parameters $\boldsymbol{\theta}$ of the model corresponding to the structure π, an adequate loss function L for the problem, and constructs a DNN model M. Learning refers to finding a suitable set of parameters $\boldsymbol{\theta}$ for the DNN such that the model correctly represents the data D. This involves a procedure that iteratively updates the parameters in $\boldsymbol{\theta}$ given the data D by minimising the loss function L.

Data D consists of pairs of data-instances and their target outputs. That is, D consits of $(\mathbf{x}, y)$ pairs, where $\mathbf{x}$ represents a data-instance and y represents a category

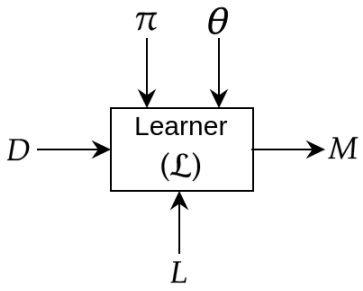


Figure A.1: Construction of a DNN model from data (Based on Figure 2.2; reproduced here for readability and completeness).

(for classification problems) or a real-value (for regression problems). $\mathbf{x}$ can be: a vector or tensor of properties describing a data instance, an image, a graph, or a sentence. For the applications studied in this dissertation, y is usually a class-label, although this is not necessary. Depending on what $\mathbf{x}$ is, there are different kinds of DNNs. For examples, if $\mathbf{x}$ is a vector (or in general, a tensor) there are multilayer perceptrons (MLPs), if $\mathbf{x}$ is an image there are convolutional neural networks (CNNs), if $\mathbf{x}$ is graph, there are graph neural networks (GNNs), etc.

Before describing what $\mathbf{x}$ is in this dissertation, we first summarise standard deep neural network architectures that are used in our research. We use two kinds of deep neural networks: (1) deep fully-connected feed-forward neural network, called Multilayer Perceptrons or MLPs, and (2) graph neural networks or GNNs.

Multilayer Perceptrons (MLPs)

MLPs are a class of deep neural network architectures consisting of multiple layers of neurons, each fully connected to those in the preceding layer (from which they receive input) and to those in the succeeding layer (which they, in turn, influence). The layer of neurons corresponding directly with the input features is called the input layer (or simply, "inputs") and the layer corresponding to the output(s) is called the output layer. The layers of neurons in between the inputs and the output layer are called hidden layers. A simple box diagram of a L-layered MLP structure is shown in Fig. A.2.

We now discuss some fundamental computations involved in an MLP. We start with a set of notations: Let $\mathbf{x} \in \mathbb{R}^d$ denote a data instance with d-features $x_1, \ldots, x_d$. This forms the input to an MLP. Let $m^{(\ell)}$ denote the size (number of neurons) of any layer ℓ. Clearly, $m^0 = d$. Let $\mathbf{z}^{(\ell)} \in \mathbb{R}^{m^{(\ell)}}$ denote the inputs and $\sigma^{(\ell)}$ be the activation for the hidden layer ℓ. Then, $\mathbf{h}^{(\ell)} \in \mathbb{R}^{m^{(\ell)}}$. The parameters of any layer ℓ is denoted by $\mathbf{W}^{(\ell)} \in \mathbb{R}^{m^{(\ell-1)} \times m^{(\ell)}}$. Let the bias parameters at layer ℓ be denoted by $\mathbf{b} \in \mathbb{R}^{m^{(\ell)}}$. The forward propagation in MLP consists of the following computations:

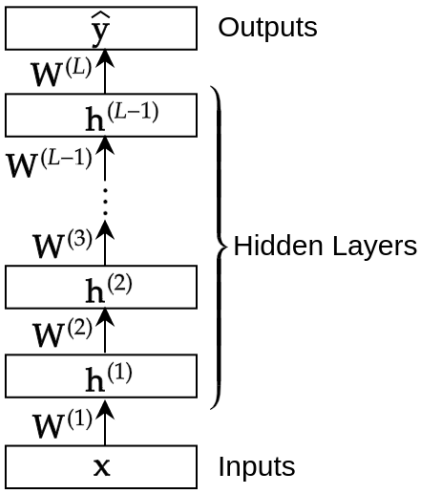


Figure A.2: Representing MLP with layers as boxes. No importance to be given to the width of the boxes. The *depth* of the MLP is L. $\mathbf{h}$ denotes a vector of hidden layer activations (also called hidden representation) and $\hat{\mathbf{y}}$ denotes the outputs. Superscript (ℓ) represents the layer index. The arrows show propagation of information (activations) from one layer to another. $\mathbf{W}^{(\ell)}$ denotes the parameters (a matrix of synaptic weights) at layer ℓ.

For layer $\ell = 0, \ldots, L-1$:

$$\mathbf{z}^{(\ell)} = \mathbf{h}^{(\ell-1)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)} \tag{A.1}$$

$$\mathbf{h}^{(\ell)} = \sigma^{(\ell)}(\mathbf{z}^{(\ell)}) \tag{A.2}$$

Clearly, $\mathbf{h}^{(0)} = \mathbf{x}$. For the output layer, the computations are then:

$$\mathbf{z}^{(L)} = \mathbf{h}^{(L-1)}\mathbf{W}^{(L)} + \mathbf{b}^{(L)} \tag{A.3}$$

$$\hat{\mathbf{y}} = \sigma^{(L)}(\mathbf{z}^{(L)}) \tag{A.4}$$

In the above computations, the activation function $\sigma^{(\ell)}$ is applied element-wise. It decides the degree of activation of a neuron based on the net input it receives. Below, we provide some common activation functions:

1. Linear activation: It results in an affine transformation of the input.

$$\sigma(z) = \alpha z \tag{A.5}$$

   where $\alpha \in \mathbb{R}$ is some constant.

2. Sigmoid or logistic activation: It squashes the input to a value in the range $[0, 1]$.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{A.6}$$

3. Hyperbolic tangent (tanh) activation: It squashes the input to a value in the range $[-1, 1]$.

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{A.7}$$

4. Rectified Linear Unit (ReLU) activation: A simple and popular non-linear function for deep networks.

$$\sigma(z) = \max(0, z) \tag{A.8}$$

5. Variants of ReLU:

   (a) Leaky ReLU:

   $$\sigma(z) = \max(0.01z, z) \tag{A.9}$$

   (b) Parametric ReLU:

   $$\sigma(z) = \max(\alpha z, z) \tag{A.10}$$

   where $\alpha \in \mathbb{R}$ is a learnable parameter.

Learning of an MLP refers to the update of the model parameters $\mathbf{W}$s given some labelled data in the form of $(\mathbf{x}, y)$ pairs. This process is referred to as "training". Training in a deep neural network is usually done using the popular *backpropagation* procedure [RHW86]. Backpropagation updates the parameters (synaptic weights and sometimes, other hyperparameters) using the chain-rule of derivative calculus for propagating the gradients of the loss from output layer towards the inputs.

Graph Neural Networks (GNNs)

MLPs described above can learn from data-instances that are described using numeric feature-vectors and cannot be adopted directly for graph-structured data where each data-instance is a graph. MLPs would require that the graph instances to be converted to fixed-length numeric vectors which can then be used as inputs. There is a class of DNNs that are suitable for learning directly from graph-structured data, called graph neural networks (GNNs). GNNs, however, do involve a structure similar to MLPs within their structure. We will discuss some implementational aspects of GNNs here. For a more complete technical overview, the reader is referred to [Ham20].

In Chapter 4, we discussed that GNNs, in their implementations, involve 3 procedures: (a) AGGREGATE: For every vertex, this procedure aggregates the information from neighboring vertices; and (b) COMBINE: This procedure updates the label of the vertex by combining its present label with its neighbors'; and (c) READOUT: This procedure constructs a vectorised representation of the entire graph. In mathematical forms, these procedures are described as follows:

For a graph G, at some iteration k, the labelling of a vertex v (denoted by h_v) is updated as

$$a_v^{(k)} = \mathsf{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right),$$
$$h_v^{(k)} = \mathsf{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

where, $\mathcal{N}(v)$ denotes the set of vertices adjacent to v. Initially (at $k = 0$), $h_v^{(0)} = X_v$.

The vector representation of the entire graph G is obtained in the final iteration ($k = K$) as

$$h_G = \mathsf{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right)$$

In practice, AGGREGATE and COMBINE procedures are implemented using graph convolution and pooling operations. The READOUT procedure is usually implemented using a global or hierarchical pooling operation [XHLJ19]. Variants of GNNs result from modifications to these 3 procedures: AGGREGATE, COMBINE and READOUT.

Implementation of AGGREGATE-COMBINE

There are several variants of GNNs of which some GNN variants are quite popular due to their successes in many different real-world problems. We provide some brief notes on their implementation, primarily, focusing on how the graph convolution operation is implemented in them. We focus mainly on the following variants of graph convolutions that are used in the research conducted in this dissertation: (1) GCN: spectral graph convolution [KW17], (2) k-GNN: multistage graph convolution [MRF+19], (3) GAT: graph convolution with attention [VCC+18], (4) GraphSAGE: simple-and-aggregate graph convolution [HYL17], and (5) ARMA: graph convolution with auto-regressive moving average [BGLA21].

Variant 1: GCN

Based on the spectral-based graph convolution as proposed by [KW17], this graph convolution uses a layer-wise (or iteration-wise) propagation rule for a graph with N vertices as:

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\Theta^{(k-1)}\right) \tag{A.11}$$

where, $H^{(k)} \in \mathbb{R}^{N\times D}$ denotes the matrix of vertex representations of length D, $\tilde{A} = A + I$ is the adjacency matrix representing an undirected graph G with added self-connections, $A \in \mathbb{R}^{N\times N}$ is the graph adjacency matrix, I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\Theta^{(k-1)}$ is the iteration-specific trainable parameter matrix, $\sigma(\cdot)$ denotes the activation function e.g. $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{X}$ is the matrix of feature-vectors of the vertices, where each vertex i is associated with a feature-vector X_i.

Variant 2: k-GNN

This graph convolution passes messages (vertex feature-vectors) directly between subgraph structures inside a graph [MRF+19]. At iteration k, the feature representation of a vertex is computed by using

$$h_u^{(k)} = \sigma\left(h_u^{(k-1)} \cdot \Theta_1^{(k)} + \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} \cdot \Theta_2^{(k)}\right) \tag{A.12}$$

where, h_u^k denotes the vertex-representation of a vertex u at iteration k, $\mathcal{N}$ denotes the neighborhood function, σ is a non-linear transfer function applied component wise to the function argument, Θs are the layer-specific learnable parameters of the network.

Variant 3: GAT

This variant is based on aggregating information from neighbours with attention. This approach is popularly known as Graph Attention Network (GAT: [VCC+18]). This network assumes that the contributions of neighboring vertices to the central vertex are not pre-determined which is the case in the Graph Convolutional Network [KW17]. This adopts attention mechanisms to learn the relative weights between two connected vertices. The graph convolutional operation at iteration k is thereby defined as:

$$h_u^{(k)} = \sigma\left(\sum_{v \in \mathcal{N}(u) \cup u} \alpha_{uv}^{(k)} \Theta^{(k)} h_u^{(k-1)}\right) \tag{A.13}$$

where, h_u^k denotes the vertex-representation of a vertex u at iteration k; $h_u^{(0)} = X_u$ (the initial feature-vector associated with a vertex u). The connective strength between the vertex u and its neighbor vertex v is called attention weight, which is defined as

$$\alpha_{uv}^{(k)} = \text{softmax}\left(\text{LeakyReLU}\left(a^{\intercal}\left[\Theta^{(k)} h_u^{(k-1)} \parallel \Theta^{(k)} h_v^{(k-1)}\right]\right)\right) \tag{A.14}$$

where, a is the set of learnable parameters of a single layer feed-forward neural network, $||$ denotes the concatenation operation.

Variant 4: GraphSAGE

This graph convolution is based on inductive representation learning on large graphs, as proposed in [HYL17]. The convolution technique here is used to generate low-dimensional vector representations for vertices by learning how to aggregate feature information from the neighbourhood of the vertices. It adopts two steps: First, it samples a neighbourhood vertices of a vertex; Second, aggregate the feature-information from these sampled

vertices. GraphSAGE is used to found to be very useful for graphs with vertices associated with rich feature-vectors. The following is an iterative update of the vertex representations in a graph:

$$h_u^{(k)} = \sigma\left(h_u^{(k-1)} \cdot \Theta_1^{(k)} + \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} \cdot \Theta_2^{(k)}\right) \tag{A.15}$$

where, h_u^k denotes the vertex-representation of a vertex u at iteration k, σ is a non-linear transfer function applied component wise to the function argument, $\mathcal{N}$ denotes the neighborhood function, Θs are the layer-specific learnable parameters of the network.

Variant 5: ARMA

This graph convolution is inspired by the auto-regressive moving average (ARMA) filters that are considered to be more robust than polynomial filters [BGLA21]. The ARMA graph convolutional operation is defined as:

$$\mathbf{H}^{(k)} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{H}_m^{(K)} \tag{A.16}$$

where, $\mathbf{H}^k$ denotes the vertex-representation matrix at iteration k, M is the number of parallel stacks, K is the number of layers; and $\mathbf{H}_m^{(K)}$ is recursively defined as

$$\mathbf{H}_m^{(k+1)} = \sigma\left(\hat{L}\mathbf{H}_m^{(k)}\Theta_2^{(k)} + \mathbf{H}^{(0)}\Theta_2^{(k)}\right) \tag{A.17}$$

where, σ is a non-linear transfer function, $\hat{L} = I - L$ is the modified Laplacian. The Θ parameters are learnable parameters.

Graph Pooling

In addition to the graph convolution methods mentioned above, graph pooling is a method that applies down-sampling to graphs. This operation allows to obtain refined graph representations at each layer. Like in convolutional neural networks, a (graph-)pooling operation follows a (graph-)convolution operation. The primary aim of including a graph pooling operation after each graph convolution is that this operation can reduce the graph representation while ideally preserving important structural information.

In the research conducted in this dissertation, we use a popular structural-attention based graph pooling method [LLK19]. This method uses the graph convolution defined in Equation (A.11) to obtain a self-attention score as given in Equation (A.18) with the trainable parameter replaced by $\Theta_{att} \in \mathbb{R}^{N \times 1}$, which is a set of trainable parameters in

the pooling layer.

$$Z = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{X}\Theta_{att}\right) \tag{A.18}$$

Here, $\sigma(\cdot)$ is the activation function e.g. tanh.

Implementation of READOUT

The graph-convolution and graph-pooling operations described in the preceding subsection allows an iterative construction of vertex-representations. To deal with the problem of graph classification (as is the case in this dissertation), we need to represent an input graph as a "flattened" fixed-length feature-vector that can then be used with a standard fully-connected multilayer neural network (e.g. Multilayer Perceptron) to produce a class-label. To construct this graph-representation (mostly, a dense real-valued feature-vector, also called a *graph-embedding*), we use hierarchical graph-pooling method proposed by [CVJ+18].

The hierarchical pooling method is implemented with two operations: (a) global average pooling, that averages all the learnt vertex representations in the final (readout) layer; (b) augmenting the representation obtained in (a) with the representation obtained using global max pooling, that seek to obtained the most relevant information and could strengthen the graph-representation. The term "hierarchical" refers to the fact that the above two operations (a) and (b) are carried out after each "convolution-pooling" operation in the GNN. The final graph representation is an aggregate of all the layer-wise representations by taking their sum. The output graph after each convolution-pooling block can be represented by a concatenation of the global average pool representation and the global max pool representation as

$$H_G^{(k)} = \text{avg}(\mathbf{H}^{(k)}) \,||\, \max(\mathbf{H}^{(k)}) \tag{A.19}$$

where, H_G^k denotes the graph-representation at iteration k; $\mathbf{H}^{(k)}$ denotes the matrix of vertex-representations after convolution-pool operations at iteration k as mathematically described in the preceding subsection; avg and max denote the average and max operations, which are computed as follows:

$$\text{avg}(\mathbf{H}^{(k)}) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{H}_i^{(k)} \tag{A.20}$$

$$\max(\mathbf{H}^{(k)}) = \max_{i=1}^{N}\mathbf{H}_i^{(k)} \tag{A.21}$$

Here, $\mathbf{H}_i^k$ denotes the representation for the ith vertex of the graph; N is the number of nodes in the graph.

The final fixed-length representation after iteration K for the whole input graph is

then computed by the element-wise sum, denoted as $\oplus$, of these intermediate graph-representations in Equation (A.19):

$$H_G^{(K)} = \oplus_{k=1}^{K} H_G^{(k)} \tag{A.22}$$

A.2 Inductive Logic Programming (ILP)

Here we provide some conceptual details on ILP. By no means, this section is intended to provide a complete technical overview of ILP; for which the reader could refer to [Mug91, Md94].

ILP is a symbolic machine learning method that construct models from data and background knowledge. It is a formal framework for symbolic machine learning and provides practical algorithms for inductively learning relational descriptions for data (in the form of programs in first-order logic) from training examples and background knowledge, both uniformly represented in relational form (first-order logic). We describe the learning problem in ILP with an illustration of the classic east-west train classification problem, proposed by Michalski [Mic80, MMPS94], which we have discussed extensively in Chapter 3.

Example A.1. *Michalski's east-west train classification problem has the following setting:*

- *Consider there are 10 trains, 5 going east and 5 going west as re-shown in the figure below. Each train can consist of more than one cars.*

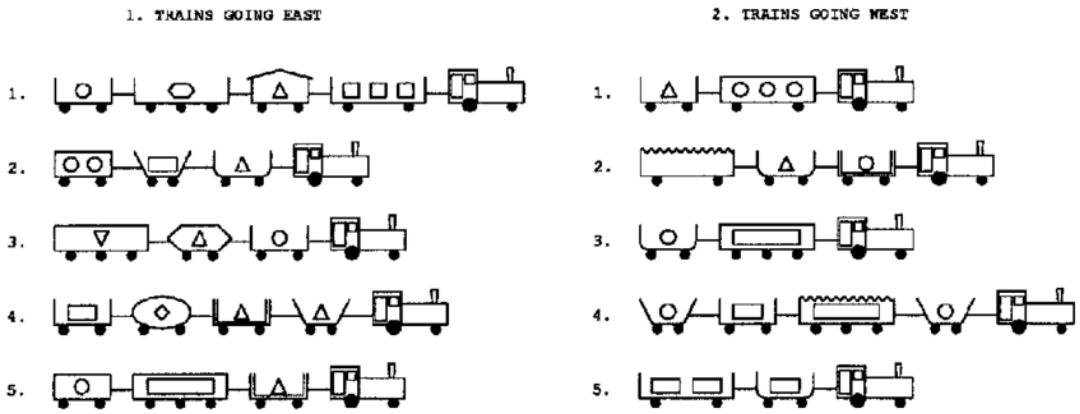


Figure A.3: Michalski's trains problem; adapted from [Mic80, MMPS94].

- *In a relational representation these trains are represented as facts:*

  Positive examples, E^+**:** $eastbound(east1)$, ..., $eastbound(east5)$;

  Negative examples, E^-**:** $eastbound(west6)$, ..., $eastbound(west10)$.

- *Each train comprises a set of locomotive pulling wagons; whether a particular train is travelling towards the east or towards the west is determined by some properties of that train.*

- *The learning task here is to determine what governs which kinds of trains are Eastbound and which kinds are Westbound.*

- *The following background knowledge about each wagon (or car) in the train are available: which train it is part of, its shape, how many wheels it has, whether it is open (i.e. has no roof) or closed, whether it is long or short, the shape of the things the car is loaded with. In addition, for each pair of connected wagons, knowledge of which one is in front of the other can be extracted.*

- *Let us assume the following background knowledge for the cars in train east*1*:*

  $short(car12)$, $short(car14)$, $long(car11)$, $long(car13)$,
  $closed(car12)$, $open(car11)$, $open(car13)$, $open(car14)$,
  $in\_front(car11, car12)$, $in\_front(car12, car13)$, $in\_front(car13, car14)$,
  $shape(car11, rectangle)$, $shape(car12, rectangle)$,
  $shape(car13, rectangle)$, $shape(car14, rectangle)$,
  $load(car11, rectangle, 3)$, $load(car12, triangle, 1)$,
  $load(car13, hexagon, 1)$, $load(car14, circle, 1)$,
  $wheels(car11, 2)$, $wheels(car12, 2)$, $wheels(car13, 3)$, $wheels(car14, 2)$,
  $has\_car(east1, car11)$, $has\_car(east1, car12)$,
  $has\_car(east1, car13)$, $has\_car(east1, car14)$.

  *Here cars are uniquely identified by constants of the form carxy, where x is number of the train to which the car belongs and y is the position of the car in that train. For example, car*14 *refers to the fourth car behind the locomotive in the first train.*

- *Then an ILP system could generate the following hypothesis:*

$$eastbound(X) \leftarrow has\_car(X, Y), short(Y), closed(Y)$$

  meaning, a train is eastbound if it has a car which is both short and closed.

The hypothesis about an eastbound train is simply a relational description. The core technique developed within ILP to constrain the search for such relational descriptions for data is mode-directed inverse entailment (MDIE), which was introduced by Muggleton in [Mug95]. More detailed description is provided in Chapter 5. For completeness, we briefly describe MDIE and the related concepts below.

Mode-Directed Inverse Entailment (MDIE)

Mode-directed Inverse Entailment (MDIE [Mug95]) is a technique for constraining the search for explanations for data in Inductive Logic Programming (ILP). Given a relational data instance e, background knowledge B, a set of modes M, a depth-limit d, and language restriction $\mathcal{L}$, MDIE identifies a most-specific logical formula $\bot_{B,\mathrm{M},d}(e)$ that contains all the relational information in B that is related to e.

In MDIE, Background knowledge B is given as a logic theory in the form of Horn clauses; data instances are provided as a set of positive and negative examples, $E = E^+ \wedge E^-$. Based on MDIE, a correct hypothesis H is a conjunction of definite clauses $H = D_1 \wedge D_2 \wedge \ldots$ which satisfies the following logical requirements:

- Prior necessity ($B \not\models E^+$): It forbids any generation of a hypothesis as long as the positive examples are explainable without it. More clearly, this requirement checks that at least one positive example cannot be explained by the background knowledge B alone.
- Posterior Sufficienty ($B \wedge H \models E^+$): It requires any generated hypothesis h to explain all positive examples E^+.
- Prior satisfybility ($B \wedge H \not\models \square$): This is a weak consistency requirement that forbids generation of any hypothesis h that contradicts the background knowledge B.
- Posterior satisfybility ($B \wedge H \wedge E^- \not\models \square$): This is a strong consistency requirement that forbids generation of any hypothesis that contradicts the negative examples (if given).

Here $\models$ denotes logical consequence and $\square$ denotes a contradiction. MDIE implementations attempt to find the most-probable H, given B and the data E^+, E^-. The key concept used in [Mug95] is to constrain the identification of the D_j using a *most-specific clause*. Let us look at the following example (re-used here from Chapter 5):

Example A.2. *In the following, capitalised letters like X, Y denote variables. Let B:*

$$parent(X,Y) \leftarrow father(X,Y)$$
$$parent(X,Y) \leftarrow mother(X,Y)$$
$$mother(jane, alice) \leftarrow$$

e:

$$gparent(henry, john) \leftarrow father(henry, jane), mother(jane, john)$$

The most-specific clause for the example $gparent(henry, john)$, denoted by $\bot_{B,\mathrm{M},d}(e)$ is:

$$gparent(henry, john) \leftarrow father(henry, jane), mother(jane, john), mother(jane, alice), parent(henry, jane), parent(jane, john), parent(jane, alice)$$

Language Restrictions

In this section we intuitively describe what "mode" means in ILP. Given a set of examples and background knowledge, the space of possible hypotheses in ILP tends to be very large [Mug91, Rae10]. To reduce the search space is to be more specific about how the predicates in the hypothesis (Horn) clauses will look like. There are two possible ways this can be achieved: (1) by limiting the number of existentially quantified variables allowed in the learned clauses; and (2) by explicitly specifying what the learned hypothesis will look like, in terms of restrictions on both their head and their body. In particular, for each predicate in the body of a hypothesis clause, it is possible to specify whether an argument in the predicate is to be a ground term, a new variable or a variable given in the head. This is done in some popular ILP systems such as Aleph [Sri01]. This is called a language restriction in ILP and it drastically reduces the search time by confining the search to be carried out in a limited search space. In implementations, this is carried out by providing a set of mode declarations [Mug95]. There are two kinds of mode declarations:

modeh declaration This is a mode declaration that dictates what the head of the hypothesis clauses will look like.

modeb declaration This is a mode declaration that stipulates the format of the predicates in the body of the clauses.

Let us look the following examples:

Example A.3. *For our grandparent example above, the following are some mode declarations:*

$$\boxed{\begin{array}{l} modeh(gparent(+person, -person)) \\ modeb(father(+person, -person)) \\ modeb(mother(+person, -person)) \\ modeb(parent(+person, -person)) \end{array}}$$

The first mode declaration, modeh is for the head of a hypothesis clause that stipulates that the head literal will have a predicate name gparent and that the first argument will take in a given person name (specified by '+', called input) and second argument will return a person name (specified by '-', called output). This means that the learned predicate will take in the name of a person and return its grand parent, as required. Similar description applies to the body literals of the clause, where there can be three kinds of body literals; father, mother, parent.

Example A.4. *There can be mode declaration with argument in the predicate being specified with '#'. Let us consider the following two mode declarations:*

$$\boxed{\begin{array}{l} modeh(class(+flower, \#category)) \\ modeb(colour(+flower, \#colourname)) \end{array}}$$

The modeh declaration here stipulates that the predicate name class will take a given flower (specified by '+') and the second argument will return a ground instance (specified by '#') of the type category. Similar description applies to the modeb declaration.

In addition to the mode declarations, the language restriction requires few more parameters such as the 'depth-limit', denoted by d, and the number of literals in the body of a clause. Informally, depth refers to the depth of a term (arguments) in the literals (predicates) appearing the body of a hypothesis clause. More details on this parameter is provided in Chapter 5. The language restriction is often referred to as the 'depth-limited mode language' in ILP.

Some ILP Learning Systems

There are several implementations of ILP learning systems. Probably one of the oldest is FOIL [Qui90], and the newest is Popper [CM21]. There are close to 20 implementations of ILP learning systems (as per [CD20, CDEM22]) in the last 3 decades that are based on considerably different techniques. It is difficult to provide details on these learning systems in this dissertation, and therefore, we direct the reader to some relevant discussions on some of these systems in the following survey: [CD20]. In this section, we briefly focus first on Progol [Mug95], an ILP system that is based on MDIE as discussed earlier. Progol is a precursors to one of the most popular ILP system, Aleph [Sri01]. In general, the technique of MDIE and the Aleph system are extensively used in this dissertation.

Progol

Progol is one of the most important ILP systems that has inspired development of many other ILP systems, including Aleph [Sri01]. Progol combines the technique of MDIE with general-to-specific search through a refinement graph representing the discrete hypothesis space. The space is bounded by an empty clause (at the top of the refinement graph) and a most-specific clause (at the bottom of the refinement graph) that entails a positive example, given a set of mode-declarations and background knowledge. A simple diagrammatic representation of the bounded search space in Progol is shown in Figure A.4. The search of an optimal hypothesis is performed by A*-search, over clauses which subsume the most-specific clause in the refinement graph.

For more details on Progol, the reader is directed to: [Mug95] and [Mug96]. A more concise description is available at [Rob97]. Next we describe a successor of Progol, called Aleph, in more detail.

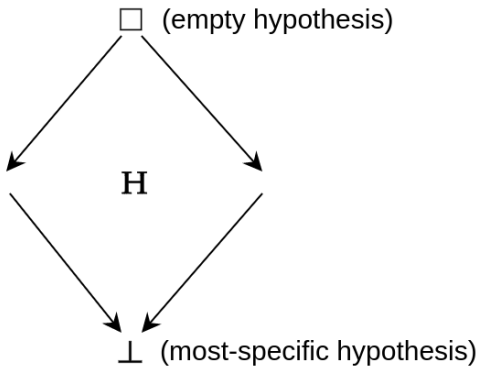


Figure A.4: Bounded search space in Progol.

Aleph

Aleph [Sri01] is based on the method of *inverse entailment* [Mug95] and uses a bottom-up learning mechanism where it restricts the hypothesis space by constructing a bottom-clause from the data and background-knowledge. Despite being at least 30 years old since its inception, Aleph remains to be the one of the most popularly used ILP system in relational learning research. The primary reason of its popularity could be its ease-of-use nature and the software is accompanied by a detailed user-manual [Sri01]. Aleph is written in Prolog, and its implementation is influenced by Progol [Mug95]. The learning in Aleph is based on the learning from entailment.

To construct a hypothesis, Aleph starts with an empty hypothesis and employs the following steps:

(1) Select a positive example to generalise. If none exists, stop and return the current hypothesis; otherwise proceed to the next step.

(2) Construct the most specific clause (the depth-limited bottom-clause) that is consistent with the mode declaration and entails the example.

(3) Search for a clause more general than the bottom-clause and has the best *score*.

(4) Add the clause to the hypothesis and remove all the positive examples covered by it. Return to Step 1.

Next we elaborate on the approaches to steps 2 and 3 below.

Step 2 The purpose of constructing a bottom-clause for an example is to bound the search in Step. (3) above. In general, a bottom-clause could be of infinite length (infinite cardinality). Aleph uses a depth-limited mode language to restrict them. During the search space, Aleph only considers the clauses that are generalisations of the bottom-clause, all of which entail the example.

Step 3 Aleph starts the search from the most general hypothesis and specialises it (by adding literals from the bottom clause) until it finds the best hypothesis. This results in a discrete space of clauses forming a search lattice. Figure A.5 shows the hypothesis space for the grandparent example. Aleph evaluates each clause in the lattice and assigns a score based on how well the training set is described by the clause. The default evaluation measure is the coverage as $P - N$, where P and N are the numbers of positive and negative examples, respectively, entailed by the clause. The users could also provide other predefined measures for search. Aleph tries to specialise a clause by adding literals to the body of the clause, which it selects from the bottom-clause or by instantiating variables. Each specialisation of a clause is called refinement. If Aleph finds the best clause that satisfies some specified constraint on the evaluation score, it adds it to the hypothesis, and remove all the positive examples covered by the new hypothesis, and returns to Step (1).

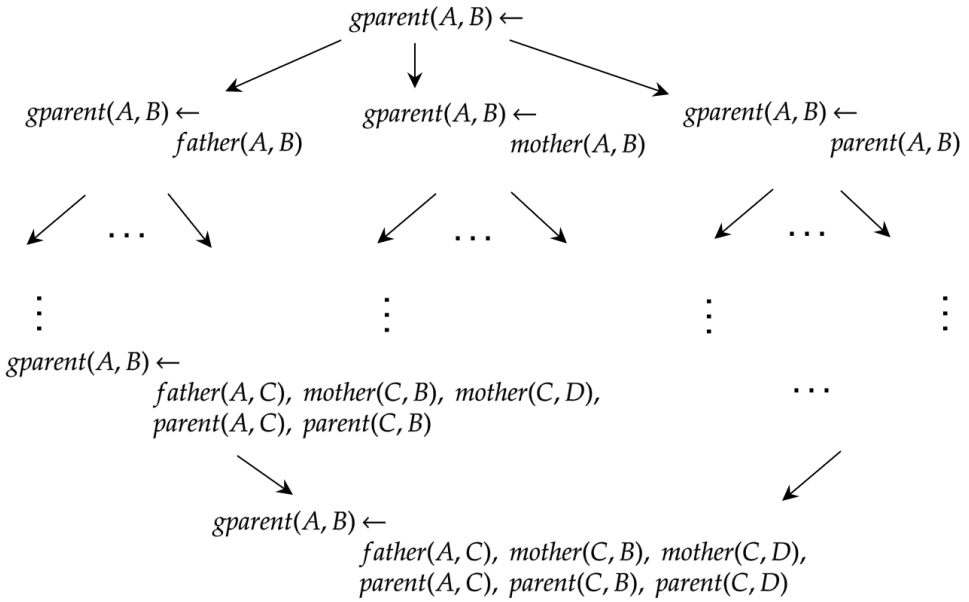


Figure A.5: A fragment of the hypothesis space in Aleph for the grandparent example, bounded by the most general hypothesis (at the top) and the most specific hypothesis (at the bottom).

Aleph offers several functionalities such as: constraint learning, mode learning, abductive learning, feature construction. In Chapter 3, the feature construction functionality of Aleph is extensively used. More details on Aleph can be found in [Sri01].

Appendix B

Additional Experimental Details

B.1 Details relevant to Chapter 3

Bottom-Clause Propositionalisation (BCP [FZG14]) constructs propositions using the most-specific clauses returned by the ILP system Aleph given the background-knowledge B, modes M and depth-limit d. In BCP, each data instance is represented using a Boolean vector of 0's and 1's, depending on the value of propositions (constructed manually or automatically) for the data instance (the value of the i^{th} dimension is 0 if the i^{th} proposition is false for the data-instance and 1 otherwise). The propositions represent the relational features constructed from the literals of a bottom-clause in ILP. For the construction of these Boolean features using BCP, we use the code available at [Jan20]. The resulting dataset is used to construct an MLP model. Our construction of MLP using BCP features is based on the following setup:

- The MLP is implemented using Tensorflow-Keras [C+15].
- The number of layers in MLP is tuned using a validation-based approach. The parameter grid for number of hidden layers is: $\{1, 2, 3, 4\}$.
- Each layer has fixed number of neurons: 10.
- The dropout rate is 0.5. We apply dropout [SHK+14] after every layer in the network except the output layer.
- The activation function used in each hidden layer is `relu`.
- The training is carried out using the Adam optimiser [KB15] with learning rate 0.001.
- Additionally, we use early-stopping [Pre98] with a patience period of 50 to control over-fitting during training.

B.2 Details relevant to Chapter 5

Mode-Declarations

We use the ILP engine, Aleph [Sri01] to construct the most-specific clause for a relational data instance given background-knowledge, mode specifications and a depth. The mode-language used for our main experiments in the chapter is given below:

```
:- modeb(*,bond(+mol,-atomid,-atomid,#atomtype,#atomtype,#bondtype)).
:- modeb(*,has_struc(+mol,-atomids,-length,#structype)).
:- modeb(*,connected(+mol,+atomids,+atomids)).
:- modeb(*,fused(+mol,+atomids,+atomids)).
```

The '#'-ed arguments in the mode declaration refers to type, that is, #`atomtype` refers to the type of atom, #`bondtype` refers to the type of bond, and #`structype` refers to the type of the structure (functional group or ring) associated with the molecule.

Experiments with ILP Benchmarks

The seven datasets are taken from [SKB03]. These datasets are some of the most popular benchmark datasets to evaluate various techniques within ILP studies. For the construction of BotGNNs, the following details are relevant:

- There is background knowledge available for each dataset.
- There are 10 splits for each dataset. Therefore, for each test-split we construct BotGNNs (all 5 variants), using 8 of rest splits as training set and the remaining 1 split as a validation set.
- Since these datasets are small (few hundreds of data instances), we could manage to perform some hyperparameter tuning for construction of our BotGNNs. The parameter grids for this are: $m : \{8, 16, 32, 64, 128\}$, batch-size: $\{16, 32\}$, and learning rate: $\{0.0001, 0.0005, 0.001\}$.
- Other details are same as described in the main BotGNN experiments.
- We report the test accuracy from the best performing BotGNN variant.

B.3 Details relevant to Chapter 6

A proxy for the results of hit confirmation assays is constructed using the assay results available for the target. This allows us to approximate the results of such assays on molecules for which experimental activity is not available. Of course, such a model is only

possible within the controlled experimental design we have adopted, in which information on target inhibition is deliberately not used when constructing the discriminator in D and generator in G2. In practice, if such target-inhibition information is not available, then a proxy model would have to be constructed by other means (for example, using the activity of inhibitors of homologues).

We use the state-of-the-art chemical activity prediction package Chemprop.[1] We train a Chemprop model using the data consisting of JAK2 inhibitors and their pIC50 values. The parameter settings used are: `class-balance` = TRUE, and `epochs` = 100 (all other parameters were set to their default values within Chemprop). Chemprop partitions the data into 80% for training, 10% validation and 10% for test. Chemprop allows the construction of both classification and regression models. The performance of both kinds of models are tabulated below:

Partition	Classification (AUC)	Regression (RMSE)
Valid	0.9472	0.6515
Test	0.8972	0.6424

The classification model is more robust, since pIC50 values are on a log-scale. We use the classification model for obtaining the results in Chapter 6 (see Figure 6.7), and we use the prediction of pIC50 values from the regression model as a proxy for the results of the hit-confirmation assays.

[1]It is likely that a BotGNN with access to the information in B_D along with the Chemprop prediction would result in a better proxy model. We do not explore this here.

Bibliography

[AA+15] Martín Abadi, Ashish Agarwal, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ADL+06] Howard Y Ando, Luc Dehaspe, Walter Luyten, Elke Van Craenenbroeck, Henk Vandecasteele, and Luc Van Meervelt. Discovering h-bonding rules in crystals with inductive logic programming. *Molecular pharmaceutics*, 3(6):665–674, 2006.

[ADRDS+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[Aiz99] Igor N Aizenberg. Neural networks based on multi-valued and universal binary neurons: theory, application to image processing and recognition. In *International Conference on Computational Intelligence*, pages 306–316. Springer, 1999.

[ARD05] Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 18–26, Arlington, Virginia, USA, 2005. AUAI Press.

[BDBC+10] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

[BDR+20] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020.

[BGB+17] Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

[BGC17] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[BGKP21] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The modern mathematics of deep learning. *arXiv preprint arXiv:2105.04026*, 2021.

[BGLA21] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[BHB+18] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[BKBR21] Navneet Bung, Sowmya Ramaswamy Krishnan, Gopalakrishnan Bulusu, and Arijit Roy. De novo design of new chemical entities for sars-cov-2 using artificial intelligence. *Future Medicinal Chemistry*, 13(6):575–585, 2021.

[BLPL07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[Blu92] Avrim Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.

[BMO+21] Jannis Born, Matteo Manica, Ali Oskooei, Joris Cadow, Greta Markert, and María Rodríguez Martínez. Paccmannrl: De novo generation of hit-like anticancer molecules from transcriptomic data via reinforcement learning. *Iscience*, 24(4):102269, 2021.

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sas-

try, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[BPZ97] Igor I Baskin, Vladimir A Palyulin, and Nikolai S Zefirov. A neural device for searching direct correlations between structures and properties of chemical compounds. *Journal of chemical information and computer sciences*, 37(4):715–721, 1997.

[BVV+16] Samuel R. Bowman, L. Vilnis, Oriol Vinyals, Andrew M. Dai, R. Józefowicz, and S. Bengio. Generating sentences from a continuous space. In *CoNLL*, 2016.

[BW91] Wray L. Buntine and A. Weigend. Bayesian back-propagation. *Complex Syst.*, 5, 1991.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CD20] Andrew Cropper and Sebastijan Dumančić. Inductive logic programming at 30: a new introduction. *arXiv preprint arXiv:2008.07912*, 2020.

[CDEM22] Andrew Cropper, Sebastijan Dumančić, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Machine Learning*, 111(1):147–172, Jan 2022.

[CDM20] Andrew Cropper, Sebastijan Dumančić, and Stephen H. Muggleton. Turning 30: New ideas in inductive logic programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4833–4839, 2020.

[CHBP02] Anthony Coates, Yanmin Hu, Richard Bax, and Clive Page. The future challenges facing the development of new antimicrobial drugs. *Nature reviews Drug discovery*, 1(11):895–910, 2002.

[CL14] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic logic and mechanical theorem proving*. Academic press, 2014.

[CLZ+19] Qibin Chen, Junyang Lin, Yichang Zhang, Ming Ding, Yukuo Cen, Hongxia Yang, and Jie Tang. Towards knowledge-based recommender dialog system. *arXiv preprint arXiv:1908.05391*, 2019.

[CM21] Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, 110(4):801–856, 2021.

[CS82] Brian Cohen and Claude Sammut. Object recognition and concept learning with confucius. *Pattern Recognition*, 15(4):309–316, 1982.

[CVJ+18] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *ArXiv*, abs/1811.01287, 2018.

[CWPZ18] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2018.

[CYK+18] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder for english. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, 2018.

[CYM20] William Cohen, Fan Yang, and Kathryn Rivard Mazaitis. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *Journal of Artificial Intelligence Research*, 67:285–325, 2020.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186, 2019.

[DDS+09] 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 conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Dec86] Rina Dechter. Learning while searching in constraint-satisfaction-problems. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 178–185. Morgan Kaufmann, 1986.

[dGL20] Artur d'Avila Garcez and Luis C. Lamb. Neurosymbolic ai: The 3rd wave, 2020.

[DGS17] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

[DQW+21] Tianjian Dong, Qi Qi, Jingyu Wang, Alex X Liu, Haifeng Sun, Zirui Zhuang, and Jianxin Liao. Generative adversarial network-based transfer reinforcement learning for routing with prior knowledge. *IEEE Transactions on Network and Service Management*, 2021.

[DRG17] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 920–923. IEEE, 2017.

[DRKT07] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[DRR16] T. Demeester, Tim Rocktäschel, and S. Riedel. Lifted rule injection for relation embeddings. *ArXiv*, abs/1606.08359, 2016.

[DS13] Brian W Dymock and Cheng Shang See. Inhibitors of jak2 and jak3: an update on the patent literature 2010–2012. *Expert opinion on therapeutic patents*, 23(4):449–501, 2013.

[DSW+20] Chunning Du, Haifeng Sun, Jingyu Wang, Qi Qi, and Jianxin Liao. Adversarial and domain-aware bert for cross-domain sentiment analysis. In *Proceedings of the 58th annual meeting of the Association for Computational Linguistics*, pages 4019–4028, 2020.

[DYCFY14] Brian W Dymock, Eugene Guorong Yang, Yuyi Chu-Farseeva, and Lianbin Yao. Selective jak inhibitors. *Future medicinal chemistry*, 6(12):1439–1471, 2014.

[EG18] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

[EMM+18] Kevin Ellis, Lucas Morales, Mathias Sabl Meyer, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping domain-specific languages for neurally-guided bayesian program learning. In *Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction*, 2018.

[ES09] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):1–11, 2009.

[EWN+20] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020.

[FBDC+19] M. Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin T. Vechev. Dl2: Training and querying neural networks with logic. In *ICML*, 2019.

[FCDRDG14] Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. kLog: A language for logical and relational learning with kernels. *Artificial Intelligence*, 217:117–143, 2014.

[FGU+21] Hossein Rajaby Faghihi, Quan Guo, Andrzej Uszok, Aliakbar Nafar, Elaheh Raisi, and Parisa Kordjamshidi. Domiknows: A library for integration of symbolic domain knowledge in deep learning. *arXiv preprint arXiv:2108.12370*, 2021.

[FL19] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[FO93] Justin Fletcher and Zoran Obradovic. Combining prior symbolic knowledge and constructive neural network learning. *Connection Science*, 5(3-4):365–375, 1993.

[Fog08] Agner Fog. Sampling methods for wallenius' and fisher's noncentral hypergeometric distributions. *Communications in Statistics—Simulation and Computation®*, 37(2):241–257, 2008.

[FSK12] Tanveer A Faruquie, Ashwin Srinivasan, and Ross D King. Topic models with relational features for drug design. In *International conference on inductive logic programming*, pages 45–57. Springer, 2012.

[Fu93] L. M. Fu. Knowledge-based connectionism for revising domain theories. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(1):173–182, 1993.

[Fu95] Li Min Fu. Introduction to knowledge-based neural networks. *Knowledge-Based Systems*, 1995.

[FZG14] Manoel VM França, Gerson Zaverucha, and Artur S d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[FZG15] Manoel Vitor Macedo França, Gerson Zaverucha, and ASD Garcez. Neural relational learning through semi-propositionalization of bottom clauses. In *AAAI Spring Symposium Series*, 2015.

[GBG12] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

[GC10] Kurt De Grave and Fabrizio Costa. Molecular graph augmentation with rings and functional groups. *Journal of chemical information and modeling*, 50(9):1660–1668, 2010.

[GC21] Victor Guimarães and Vítor Santos Costa. Neurallog: a neural logic language. *arXiv preprint arXiv:2105.01442*, 2021.

[GFS21] Manas Gaur, Keyur Faldu, and Amit Sheth. Semantics of the black-box: Can knowledge graphs help make deep learning systems more interpretable and explainable? *IEEE Internet Computing*, 25(1):51–59, 2021.

[GHN+17] Anna Gaulton, Anne Hersey, Michał Nowotka, A Patricia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2017.

[GMLS20] Francesca Grisoni, Michael Moret, Robin Lingwood, and Gisbert Schneider. Bidirectional molecule generation with recurrent neural networks. *Journal of chemical information and modeling*, 60(3):1175–1183, 2020.

[GMS05] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[Gre21] Daria Grechishnikova. Transformer neural network for protein-specific de novo drug generation as a machine translation problem. *Scientific reports*, 11(1):1–13, 2021.

[GRS+21] Manas Gaur, Kaushik Roy, Aditya Sharma, Biplav Srivastava, and Amit Sheth. " who can help me?": Knowledge infused matching of support seekers and support providers during covid-19 on reddit. *arXiv preprint arXiv:2105.06398*, 2021.

[GSR+17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[GWW+16] 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, pages 192–202, 2016.

[GZ99] Artur S Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.

[Hai10] William N. Hait. Anticancer drug development: the grand challenges. *Nature Reviews Drug Discovery*, 9(4):253–254, Apr 2010.

[Háj13] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

[Ham20] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[HBC+20] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320*, 2020.

[HBZ+18] William L. Hamilton, P. Bajaj, M. Zitnik, Dan Jurafsky, and J. Leskovec. Embedding logical queries on knowledge graphs. In *NeurIPS*, 2018.

[HDFN95] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[Heb49] Donald Olding Hebb. *The organisation of behaviour: a neuropsychological theory*. Science Editions New York, 1949.

[HH21] Hiroshi Honda and Masafumi Hagiwara. Analogical reasoning with deep learning-based symbolic processing. *IEEE Access*, 9:121859–121870, 2021.

[HKBG21] Nicholas Hoernle, Rafael Michael Karampatsis, Vaishak Belle, and Kobi Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. *arXiv preprint arXiv:2111.01564*, 2021.

[HMK07] David Heckerman, Chris Meek, and Daphne Koller. Probabilistic entity-relationship models, prms, and plate models. *Introduction to statistical relational learning*, pages 201–238, 2007.

[HML+16] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. *ArXiv*, abs/1603.06318, 2016.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[HZJ07] Yu-Chi Ho, Qian-Chuan Zhao, and Qing-Shan Jia. *Ordinal Optimization: Soft Optimization for Hard Problems*. Springer, 2007.

[Jan20] Vince Jankovics. vakker/cilp. https://github.com/vakker/CILP, 2020.

[JRS08] Sachindra Joshi, Ganesh Ramakrishnan, and Ashwin Srinivasan. Feature construction using theory-guided sampling and randomised search. In *International Conference on Inductive Logic Programming*, pages 140–157. Springer, 2008.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[KBBR21] Sowmya Ramaswamy Krishnan, Navneet Bung, Gopalakrishnan Bulusu, and Arijit Roy. Accelerating de novo drug design against novel proteins using deep learning. *Journal of Chemical Information and Modeling*, 61(2):621–630, 2021.

[KGC17] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[KGS19] Ugur Kursuncu, Manas Gaur, and Amit Sheth. Knowledge infused learning (k-il): Towards deep incorporation of knowledge in deep learning. *arXiv preprint arXiv:1912.00512*, 2019.

[Kit16] Hiroaki Kitano. Artificial intelligence to win the nobel prize and beyond: Creating the engine for scientific discovery. *AI magazine*, 37(1):39–49, 2016.

[KKM+16] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de.

[KLF01] Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization Approaches to Relational Data Mining*, pages 262–291. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[KMSS96] Ross D King, Stephen H Muggleton, Ashwin Srinivasan, and MJ Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[KT07] Eyal Krupka and Naftali Tishby. Incorporating prior knowledge on features into learning. In *AISTATS*, 2007.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[KWJ+04] Ross D King, Kenneth E Whelan, Ffion M Jones, Philip GK Reiser, Christopher H Bryant, Stephen H Muggleton, Douglas B Kell, and Stephen G Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, 2004.

[L+06] Greg Landrum et al. Rdkit: Open-source cheminformatics. https://www.rdkit.org/docs/index.html, 2006.

[Lav90] Nada Lavrac. *Principles of knowledge acquisition in expert systems*. PhD thesis, Ph. D. thesis, Faculty of Technical Sciences, University of Maribor, 1990.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LDG91] Nada Lavrač, Sašo Džeroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with linus. In *European Working Session on Learning*, pages 265–281. Springer, 1991.

[LFJ+18] Lei Li, Min Feng, Lianwen Jin, Shenjin Chen, Lihong Ma, and Jiakai Gao. Domain knowledge embedding regularization neural networks for workload prediction and analysis in cloud computing. *J. Inf. Technol. Res.*, 11(4):137–154, October 2018.

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[Lip16] Zachary C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[LLK19] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743, 2019.

[Llo12] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[Lod13] Huma Lodhi. Deep relational machines. In *International Conference on Neural Information Processing*, pages 212–219. Springer, 2013.

[LS20] Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 2020.

[LSR20] Nada Lavrac, Blaz Skrlj, and Marko Robnik-Sikonja. Propositionalization and embeddings: two sides of the same coin. *Mach. Learn.*, 109(7):1465–1507, 2020.

[LWM18] Xuan Liu, Xiaoguang Wang, and Stan Matwin. Improving the interpretability of deep neural networks with knowledge distillation. *arXiv preprint arXiv:1812.10924*, 2018.

[LZZ21] Xing Luo, Dongxiao Zhang, and Xu Zhu. Deep learning based forecasting of photovoltaic power generation by incorporating domain knowledge. *Energy*, 225:120240, 2021.

[Mar18] Gary Marcus. Deep Learning: A Critical Appraisal. *arXiv*, jan 2018.

[Mar20] Gary Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

[MB88] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Machine Learning Proceedings 1988*, pages 339–352. Elsevier, 1988.

[MCP+21] Kenneth Marino, Xinlei Chen, Devi Parikh, Abhinav Gupta, and Marcus Rohrbach. Krisp: Integrating implicit and symbolic knowledge for open-domain knowledge-based vqa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14111–14121, 2021.

[Md94] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994. Special Issue: Ten Years of Logic Programming.

[MDK+18] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31:3749–3759, 2018.

[MDRP+12] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. Ilp turns 20. *Machine learning*, 86(1):3–23, 2012.

[Mic73] Ryszard S. Michalski. Discovering classification rules using variable-valued logic system VL1. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*, pages 162–172. William Kaufmann, 1973.

[Mic80] Ryszard S Michalski. Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(4):349–361, 1980.

[MIM+19] Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. Incorporating Prior Domain Knowledge into Deep Neural Networks. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, 2019.

[MMBC21] Omar Mahmood, Elman Mansimov, Richard Bonneau, and Kyunghyun Cho. Masked graph modeling for molecule generation. *Nature communications*, 12(1):1–12, 2021.

[MMPS94] Donald Michie, Stephen Muggleton, David Page, and Ashwin Srinivasan. To the international computing community: A new east-west challenge. *Distributed email document available from https://www.doc.ic.ac.uk/~shm/Papers/ml-chall.pdf*, 1994.

[MOHU03] Kenneth A Marx, Philip O'Neil, Patrick Hoffman, and ML Ujwal. Data mining the nci cancer cell line compound gi50 values: identifying quinone subtypes effective against melanoma and leukemia cell classes. *Journal of chemical information and computer sciences*, 43(5):1652–1667, 2003.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MR19] Kit-Kay Mak and Pichika. Mallikarjuna Rao. Artificial intelligence in drug development: present status and future prospects. *Drug Discovery Today*, 24(3):773–780, 2019.

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[MS98] Eric McCreath and Arun Sharma. LIME: A System for Learning Relations. In *International Conference on Algorithmic Learning Theory*, pages 336–374. Springer, 1998.

[Mug87] Stephen Muggleton. Duce, an oracle-based approach to constructive induction. In *IJCAI*, pages 287–292. Citeseer, 1987.

[Mug91] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[Mug95] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[Mug96] Stephen Muggleton. Learning from positive data. In *International conference on inductive logic programming*, pages 358–376. Springer, 1996.

[MW+97] Alan D McNaught, Andrew Wilkinson, et al. *Compendium of chemical terminology*, volume 1669. Blackwell Science Oxford, 1997.

[MZB+21] Fanhe Ma, Faen Zhang, Shenglan Ben, Shuxin Qin, Pengcheng Zhou, Changsheng Zhou, and Fengyi Xu. Monotonic neural network: combining

deep learning with domain knowledge for chiller plants energy optimization. *arXiv preprint arXiv:2106.06143*, 2021.

[Nea95] Radford M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, CAN, 1995. AAINN02676.

[Nil91] Nils J Nilsson. Logic and artificial intelligence. *Artificial intelligence*, 47(1-3):31–56, 1991.

[OOD+21] Ivan Olier, Oghenejokpeme I Orhobor, Tirtharaj Dash, Andy M Davis, Larisa N Soldatova, Joaquin Vanschoren, and Ross D King. Transformational machine learning: Learning how to learn from many related scientific problems. *Proceedings of the National Academy of Sciences*, 118(49), 2021.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[PIT18] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science Advances*, 4(7):eaap7885, 2018.

[PKD+19] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. *Estimating Node Importance in Knowledge Graphs Using Graph Neural Networks*, page 596–606. Association for Computing Machinery, New York, NY, USA, 2019.

[Plo70] Gordon D Plotkin. A note on inductive generalization. *Machine intelligence*, 5(1):153–163, 1970.

[Plo72] Gordon Plotkin. Automatic methods of inductive inference. *PhD Thesis, The University of Edinburgh*, 1972.

[Pre98] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[PSS20] Hemant Purohit, Valerie L Shalin, and Amit P Sheth. Knowledge graphs to empower humanity-inspired ai systems. *IEEE Internet Computing*, 24(4):48–54, 2020.

[PW80] Fernando CN Pereira and David HD Warren. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial intelligence*, 13(3):231–278, 1980.

[Qui90] J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[Rae10] Luc De Raedt. *Inductive Logic Programming*, pages 529–537. Springer US, Boston, MA, 2010.

[RBSR14] Tim Rocktäschel, Matko Bosnjak, Sameer Singh, and Sebastian Riedel. Low-dimensional embeddings of logic. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 45–49, Baltimore, MD, June 2014. Association for Computational Linguistics.

[RDMM20] Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4943–4950. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Survey track.

[RGL+20] Ryan Riegel, Alexander G. Gray, Francois P. S. Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, and Santosh K. Srivastava. Logical neural networks. *CoRR*, abs/2006.13155, 2020.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[RJBS07] Ganesh Ramakrishnan, Sachindra Joshi, Sreeram Balakrishnan, and Ashwin Srinivasan. Using ilp to construct features for information extraction from semi-structured text. In *International Conference on Inductive Logic Programming*, pages 211–224. Springer, 2007.

[Rob97] Sam Roberts. An introduction to progol. *Department of Computer Science, University of York*, 244, 1997.

[Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[RP20] Alan Ramponi and Barbara Plank. Neural unsupervised domain adaptation in nlp—a survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6838–6855, 2020.

[RS22] Davor Runje and Sharath M Shankaranarayana. Constrained monotonic neural networks. *arXiv preprint arXiv:2205.11775*, 2022.

[RSR15] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, Denver, Colorado, May–June 2015. Association for Computational Linguistics.

[RSSB05] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural networks*, 18(8):1093–1110, 2005.

[Ruc91] William H. Ruckle. A discrete search game. In *Theory and Decision Library*, pages 29–43. Springer Netherlands, 1991.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SAZ+18] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[SB86] Claude Sammut and Ranan B Banerji. Learning concepts by asking questions. *Machine learning: An artificial intelligence approach*, 2:167–192, 1986.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[SdCRG21] Prithviraj Sen, Breno W. S. R. de Carvalho, Ryan Riegel, and Alexander G. Gray. Neuro-symbolic inductive logic programming with logical neural networks. *CoRR*, abs/2112.03324, 2021.

[SFK+19] Niclas Stahl, Goran Falkman, Alexander Karlsson, Gunnar Mathiason, and Jonas Bostrom. Deep reinforcement learning for multiparameter optimization in de novo drug design. *Journal of Chemical Information and Modeling*, 59(7):621–630, 2019.

[SG16] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.

[SGKW19] A. Sheth, M. Gaur, U. Kursuncu, and R. Wickramarachchi. Shades of knowledge-infused learning for enhancing deep learning. *IEEE Internet Computing*, 23(6):54–63, 2019.

[SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[SGT+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SK99] Ashwin Srinivasan and Ross D King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.

[SKB03] Ashwin Srinivasan, Ross D King, and Michael E Bain. An empirical study of the use of relevance information in inductive logic programming. *Journal of Machine Learning Research*, 4(Jul):369–383, 2003.

[SKB+18] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[SKTW17] Marwin H.S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1):120–131, 2017.

[SLM20] Mattia Silvestri, Michele Lombardi, and Michela Milano. Injecting domain knowledge in neural networks: a controlled experiment on a constrained problem. *arXiv preprint arXiv:2002.10742*, 2020.

[Šou20] Gustav Šourek. *Deep Learning with Relational Logic Representations*. PhD thesis, Czech Technical University in Prague, 2020.

[SPG19] Amit Sheth, Swati Padhee, and Amelie Gyrard. Knowledge graphs and knowledge networks: the story in brief. *IEEE Internet Computing*, 23(4):67–75, 2019.

[SR11] Ashwin Srinivasan and Ganesh Ramakrishnan. Parameter screening and optimisation for ilp using designed experiments. *Journal of Machine Learning Research*, 12(2), 2011.

[Sri99a] Ashwin Srinivasan. A study of two probabilistic methods for searching large spaces with ilp, 1999.

[Sri99b] Ashwin Srinivasan. A study of two sampling methods for analyzing large datasets with ilp. *Data Mining and Knowledge Discovery*, 3(1):95–123, Mar 1999.

[Sri01] Ashwin Srinivasan. The Aleph Manual. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html, 2001.

[SS97] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[SSJ+09] Lucia Specia, Ashwin Srinivasan, Sachindra Joshi, Ganesh Ramakrishnan, and Maria das Graças Volpe Nunes. An investigation into feature construction to assist word sense disambiguation. *Machine Learning*, 76(1):109–136, 2009.

[SSR12] Amrita Saha, Ashwin Srinivasan, and Ganesh Ramakrishnan. What kinds of relational features are useful for statistical learning? In *International Conference on Inductive Logic Programming*, pages 209–224. Springer, 2012.

[SSRN06] Lucia Specia, Ashwin Srinivasan, Ganesh Ramakrishnan, and Maria das Graças Volpe Nunes. Word sense disambiguation using inductive logic programming. In *International Conference on Inductive Logic Programming*, pages 409–423. Springer, 2006.

[Stå21] Niclas Ståhl. *Integrating domain knowledge into deep learning: Increasing model performance through human expertise*. PhD thesis, Högskolan i Skövde, 2021.

[STE13] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling,

Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[STN+20] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. Ai for science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[Sto76] Lawrence D Stone. *Theory of optimal search*, volume 118. Elsevier, 1976.

[SWP+20] Petra Schneider, W Patrick Walters, Alleyn T Plowright, Norman Sieroka, Jennifer Listgarten, Robert A Goodnow, Jasmin Fisher, Johanna M Jansen, José S Duca, Thomas S Rush, et al. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(5):353–364, 2020.

[SYS+20] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

[ŠŽK21] Gustav Šourek, Filip Železnỳ, and Ondřej Kuželka. Beyond graph neural networks with lifted relational neural networks. *Machine Learning*, pages 1–44, 2021.

[TA18] Naoya Takeishi and Kosuke Akimoto. Knowledge-based distant regularization in learning probabilistic models. *arXiv preprint arXiv:1806.11332*, 2018.

[Tan97] Ah Hwee Tan. Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transactions on Neural Networks*, 1997.

[THDS15] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015.

[THM21] Efthymia Tsamoura, Timothy Hospedales, and Loizos Michael. Neural-symbolic integration: A compositional perspective. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6):5051–5060, May 2021.

[TS93] Geoffrey G Towell and Jude W Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine learning*, 13(1):71–101, 1993.

[TS94] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1-2):119–165, 1994.

[TSN90] Geofrey G Towell, Jude W Shavlik, and Michiel O Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the eighth National conference on Artificial intelligence*, volume 861866. Boston, MA, 1990.

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[TVdM18] Niket Tandon, Aparna S. Varde, and Gerard de Melo. Commonsense knowledge in machine intelligence. *SIGMOD Rec.*, 46:49–52, 2018.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[VCVD02] E. Van Craenenbroeck, H. Vandecasteele, and L. Dehaspe. Dmax's functional group and ring library. https://dtai.cs.kuleuven.be/software/dmax/, 2002.

[vRMB+21] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, et al. Informed machine learning-a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[VSBV17] Lovekesh Vig, Ashwin Srinivasan, Michael Bain, and Ankit Verma. An investigation into the role of domain-knowledge on the use of embeddings. In Nicolas Lachiche and Christel Vrain, editors, *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*, volume 10759 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2017.

[VSKB10] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[WBS+15] Kevin Williams, Elizabeth Bilsland, Andrew Sparkes, Wayne Aubrey, Michael Young, Larisa N Soldatova, Kurt De Grave, Jan Ramon, Michaela De Clare, Worachart Sirawaraporn, et al. Cheaper faster drug development validated by the repositioning of drugs against neglected tropical diseases. *Journal of the Royal society Interface*, 12(104):20141289, 2015.

[WD18] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[Wei88] David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[Wil89] Andrew F Wilks. Two putative protein-tyrosine kinases identified by application of the polymerase chain reaction. *Proceedings of the National Academy of Sciences*, 86(5):1603–1607, 1989.

[WMMR21] Thomas Winters, G. Marra, Robin Manhaeve, and L. D. Raedt. Deepstochlog: Neural stochastic logic programming. *ArXiv*, abs/2106.12574, 2021.

[WPC+20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[WZX+19] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The World Wide Web Conference*, WWW '19, page 3307–3313, New York, NY, USA, 2019. Association for Computing Machinery.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[XXK+19] Yaqi Xie, Ziwei Xu, Mohan S. Kankanhalli, Kuldeep S. Meel, and Harold Soh. Embedding symbolic knowledge into deep networks. In *Advances in Neural Information Processing Systems*, 2019.

[XZF+18] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van Den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *35th International Conference on Machine Learning, ICML 2018*, 2018.

[YLD+21] Shweta Yadav, Usha Lokala, Raminta Daniulaityte, Krishnaprasad Thirunarayan, Francois Lamy, and Amit Sheth. "when they say weed causes depression, but it's your fav antidepressant": Knowledge-aware attention framework for relationship extraction. *PloS one*, 16(3):e0248299, 2021.

[YN13] Shuo Yang and Sriraam Natarajan. Knowledge intensive learning: Combining qualitative constraints with causal independence for parameter learning in probabilistic models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 580–595. Springer, 2013.

[YZQ+19] Changchang Yin, Rongjian Zhao, Buyue Qian, Xin Lv, and Ping Zhang. Domain knowledge guided deep learning with electronic health records. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 738–747. IEEE, 2019.

[ZCH+20] 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.

[ZKK+21] Jieyu Zhao, Daniel Khashabi, Tushar Khot, Ashish Sabharwal, and Kai-Wei Chang. Ethical-advice taker: Do language models understand natural language interventions? In *Findings of the Association for Computational Linguistics: ACL-IJCNLP*, pages 4158–4164, 2021.

[ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[ZQD+20] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[ZWQ+19] Zirui Zhuang, Jingyu Wang, Qi Qi, Haifeng Sun, and Jianxin Liao. Toward greater intelligence in route planning: A graph-aware deep learning approach. *IEEE Systems Journal*, 14(2):1658–1669, 2019.

[ZYZZ18] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 2018.

List of Publications

The work carried out in this dissertation have appeared in the following peer-reviewed publications in reverse chronological order:

1. T. Dash, S. Chitlangia, A. Ahuja, A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks", *Nature Scientific Reports*, 2022.
   URL: https://doi.org/10.1038/s41598-021-04590-0

2. T. Dash, A. Srinivasan, L. Vig, A. Roy, "Using domain-knowledge to assist lead discovery in early-stage drug design", *International Conference on Inductive Logic Programming*, 2021.
   URL: https://doi.org/10.1007/978-3-030-97454-1_6

3. T. Dash, A. Srinivasan, A. Baskar, "Inclusion of domain-knowledge into GNNs using mode-directed inverse entailment", *Machine Learning*, 2021.
   URL: https://doi.org/10.1007/s10994-021-06090-8

4. T. Dash, A. Srinivasan, L. Vig, "Incorporating symbolic domain knowledge into graph neural networks", *Machine Learning*, 2021.
   URL: https://doi.org/10.1007/s10994-021-05966-z

5. T. Dash, A. Srinivasan, R.S. Joshi, A. Baskar, "Discrete stochastic search and its application to feature-selection for deep relational machines", *International Conference on Artificial Neural Networks*, 2019.
   URL: https://doi.org/10.1007/978-3-030-30484-3_3

6. T. Dash, A. Srinivasan, L. Vig, O.I. Orhobor, R.D. King, "Large-scale assessment of deep relational machines", *International Conference on Inductive Logic Programming*, 2018.
   URL: https://doi.org/10.1007/978-3-319-99960-9_2
   (⋆Winner of the Best Student Paper Award)

The author was involved in several other publications during his PhD. These publications do not constitute any content of this dissertation, but each publication has served as a motivation for the problem investigated in this dissertation. A non-exhaustive list of peer-reviewed publications is provided below.

1. G. Chhablani, A. Sharma, H. Pandey, T. Dash, "Superpixel-based Knowledge Infusion in Deep Neural Networks for Image Classification", *ACM Southeast Regional Conference*, 2022.
   URL: https://doi.org/10.1145/3476883.3520216
   (*Winner of the Best Short Paper Award)

2. A. Sonwane, G. Shroff, L. Vig, A. Srinivasan, T. Dash, "Solving Visual Analogies Using Neural Algorithmic Reasoning", *AAAI Student Abstract and Poster Program*, 2022.
   URL: https://arxiv.org/abs/2111.10361

3. I. Olier, O.I. Orhobor, T. Dash, A.M. Davis, L.N. Soldatova, J. Vanschoren, R.D. King, "Transformational machine learning: Learning how to learn from many related scientific problems", *Proceedings of the National Academy of Sciences of the U.S.A.*, 2021.
   URL: https://doi.org/10.1073/pnas.2108013118

4. S. Chitlangia, A. Sonwane, T. Dash, L. Vig, A. Srinivasan, G. Shroff, "Using Program Synthesis and Inductive Logic Programming to solve Bongard Problems", *International Workshop on Approaches and Applications of Inductive Programming*, 2021.
   URL: https://lr2020.iit.demokritos.gr/online/IJCLR_2021_paper_21.pdf

5. H. Shah, A. Vaswani, T. Dash, R. Hebbalaguppe, A. Srinivasan, "Empirical Study of Data-Free Iterative Knowledge Distillation", *International Conference on Artificial Neural Networks*, 2021.
   URL: https://doi.org/10.1007/978-3-030-86365-4_44

6. S. Krishnan, R. Khincha, L. Vig, T. Dash, A. Srinivasan, "A Case Study of Transfer of Lesion-Knowledge", *MICCAI Workshop on Medical Image Learning with Less Labels and Imperfect Data*, 2020.
   URL: https://doi.org/10.1007/978-3-030-61166-8_15

7. K. Mahajan, M. Sharma, L. Vig, R. Khincha, S. Krishnan, A. Niranjan, T. Dash, A. Srinivasan, G. Shroff, "CovidDiagnosis: Deep Diagnosis of Covid-19 Patients using Chest X-rays", *MICCAI Workshop on Thoracic Image Analysis*, 2020.
   URL: https://doi.org/10.1007/978-3-030-62469-9_6

8. S. Yalburgi, T. Dash, R. Hebbalaguppe, S. Hegde, A. Srinivasan, "An Empirical Study of Iterative Knowledge Distillation for Neural Network Compression", *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020.
URL: https://www.esann.org/.../proceedings/2020/ES2020-205.pdf

9. T. Dash, S.N. Dambekodi, P.N. Reddy, A. Abraham, "Adversarial neural networks for playing hide-and-search board game Scotland Yard", *Neural Computing and Applications*, 2018.
URL: https://doi.org/10.1007/s00521-018-3701-0

10. A. Saboo, A. Sharma, T. Dash, "GASOM: Genetic Algorithm Assisted Architecture Learning in Self Organizing Maps", *International conference on Neural Information Processing*, 2017.
URL: https://doi.org/10.1007/978-3-319-70087-8_25

11. P.P. Pai, T. Dash, S. Mondal, "Sequence-based discrimination of protein-RNA interacting residues using a probabilistic approach", *Journal of Theoretical Biology*, 2017.
URL: https://doi.org/10.1016/j.jtbi.2017.01.040

Brief Biography of the Candidate

Tirtharaj Dash started his PhD in Machine Learning in January 2017 in the Department of Computer Science and Information Systems at BITS Pilani, Goa Campus, under the supervision of Senior Professor Ashwin Srinivasan. He received a Master of Technology (M.Tech) degree in Computer Science and Engineering, with one year thesis, from VSSUT, Burla in the year 2014 and a Bachelor of Technology (B.Tech) degree in Information Technology from NIST Berhampur in the year 2012. He was the topper of his batch, both in his M.Tech. and B.Tech, and the university and institute awarded him the Silver Medals. In August 2015, Tirtharaj joined the Department of Computer Science and Information Systems at BITS Pilani, Goa Campus, as an Assistant Professor (Grade-II). In June 2020, he was inducted to the Anuradha and Prashanth Palakurthi Centre for Artificial Intelligence Research (APPCAIR), BITS Pilani, Goa Campus. Before joining BITS, he worked as an Assistant Professor in the School of Computer Science at NIST Berhampur for over a year, from 2014–2015. He also worked as an IASc-INSA-NASI Summer Research Fellow at ISI Kolkata in 2015. His research areas of interest are Deep Learning, Neuro-Symbolic Learning, Graph Representation Learning and Machine Learning. He is a regular member of the ACM.

Brief Biography of the Supervisor

Ashwin Srinivasan received his PhD from the School of Electrical Engineering and Computer Science at the University of New South Wales, Australia, in 1991. His dissertation examined the use of defeasible logic for the photo-interpretation of remotely sensed data and investigated the comparative advantage of this representation over methods like Multivariate Gaussian Analysis and Dempster Shafer Theory. During the latter half of 1990, he developed a non-monotonic logic-based system for interpreting chemical pathology data. This was awarded the Pacific Diagnostic's Prize and recommended for use in all hospitals in the state of New South Wales. In 1991, Ashwin joined the ILP group at the Turing Institute, Scotland and—with S. Muggleton (now at Imperial College, London)—worked on the application of Algorithmic Information Theory to noise-detection and non-monotonic learning in ILP. From 1993, Ashwin was a member of the Oxford University Computing Laboratory, where he was involved in pioneering applications of ILP systems to difficult real-world problems in molecular biology and chemistry. From 1998–2000 he was the Nuffield Trust Research Fellow in Medical Mathematics and a Research Fellow of Green College, Oxford. In 2001, he was appointed to a University Lecturership in Computation at Oxford and a Fellowship in Computation at St Peter's College. Prior to this, he has also been a member of Wolfson College, Oxford. In 2003, he moved to the IBM Research – India, as a Research Staff Member. In 2009, he was awarded a Ramanujan Fellowship by the Department of Science and Technology of the Government of India. In 2010, he took up the post of Professor at the newly formed South Asian University and became the founder Dean of the Faculty of Mathematics and Computer Science. He moved to the IIIT-D in Sept. 2012 and in Jan 2015 to BITS Pilani, Goa Campus. He was also a Visiting Professor at the Computing Laboratory, University of Oxford, and is a Visiting Professorial Fellow at the School of Computer Science and Engineering, University of New South Wales.

Brief Biography of the Co-supervisor

Sukanta Mondal works in the field of computational biology and bioinformatics to address various challenging questions in bio-molecular science. He received his PhD degree from Indian Institute of Science in 2007, under the supervision of Prof. Ramakumar S, for thesis work titled "Contributions to venominformatics: sequence-structure-function studies of toxins from marine cone snails. Application of order-statistics filters for detecting membrane-spanning helices". After his doctoral studies, Dr. Mondal moved to National Institutes of Biomedical Innovation, Health and Nutrition (NIBIOHN), Japan, for pursuing post-doctoral research on "Development of an international pharmaceutical innovation value chain for *in silico* drug discovery" under the supervision of Prof. Kenji Mizuguchi. Gathering rich experience at both national and international levels, he initiated his Annotate Biomolecules Computationally (ABC) group in the year 2012, which currently has various doctoral, graduate and undergraduate students working on protein functional annotation.