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).*

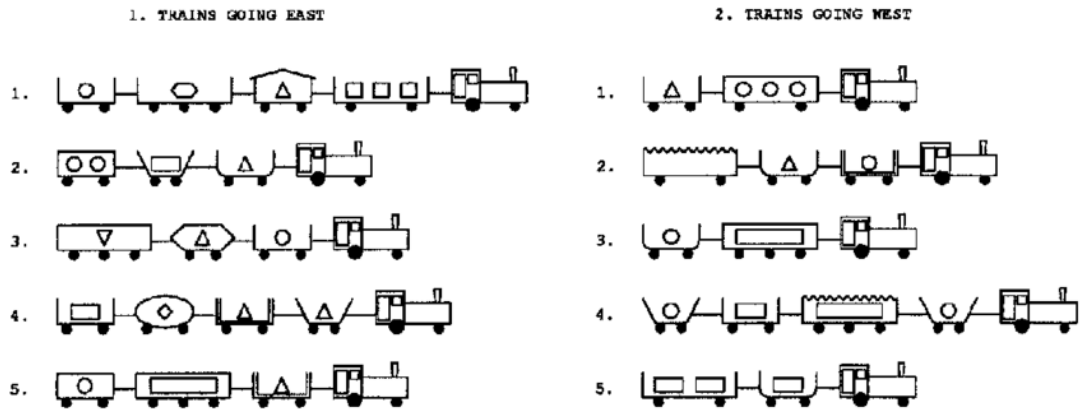

Figure 3.1: Michalski's "trains" problem; adapted from [Mic80, MMPS94].

- *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.

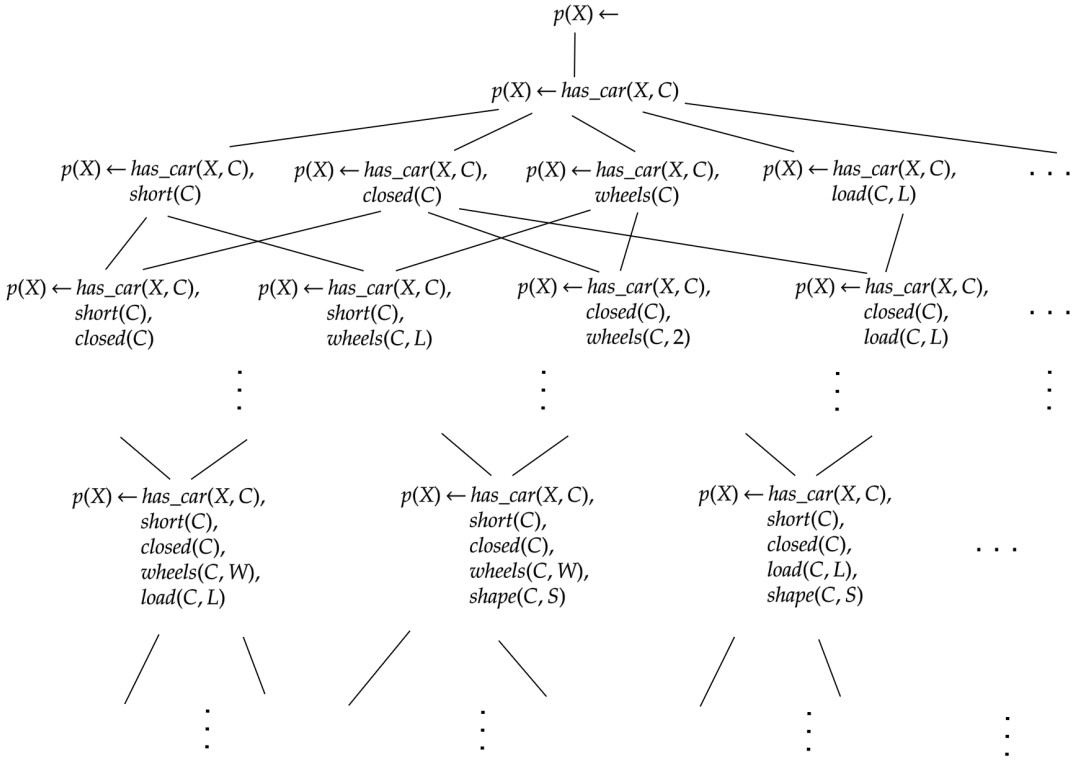

Figure 3.2: A fragment of the subsumption lattice of relational features for the trains problem.

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.*

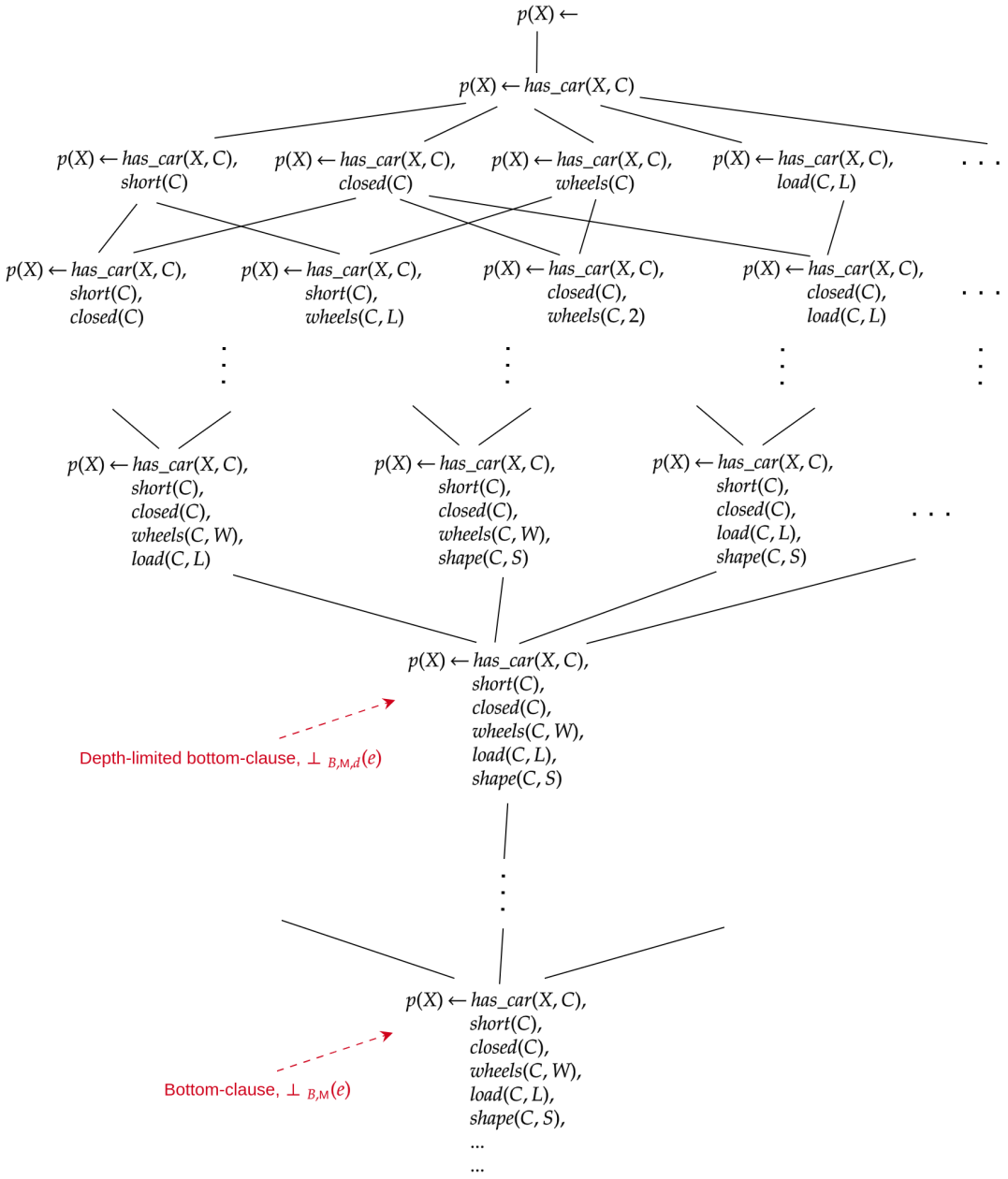

Figure 3.3: The subsumption lattice of relational features for the trains problem. The space is bounded by $p(X) \leftarrow TRUE$ at the top and by the bottom-clause ($\perp_{B,\mathrm{M}}(e)$) at the bottom. The size of the space is bounded by $\mathcal{O}(2^{|\perp_{B,\mathrm{M}}(e)|})$. The relational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

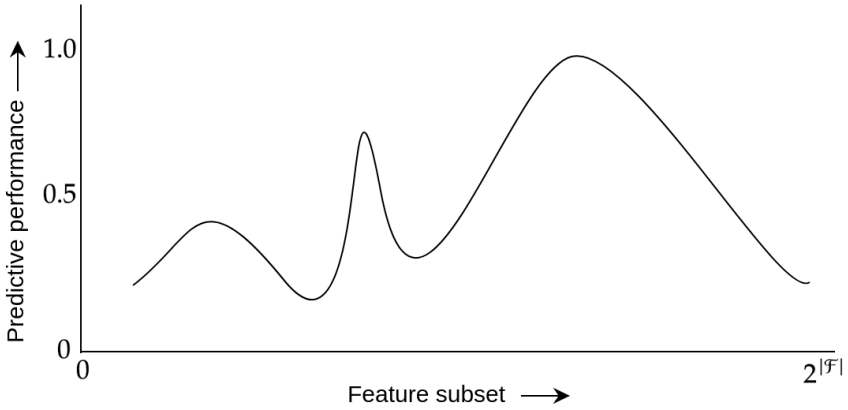

Figure 3.4: Redrawn and adapted from [JRS08]. Identifying the best subset of relational features for constructing a DRM. The X-axis enumerates the different subsets of relational features that can be constructed by an ILP engine ($\mathcal{F}$ denotes the set of all possible relational features that 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

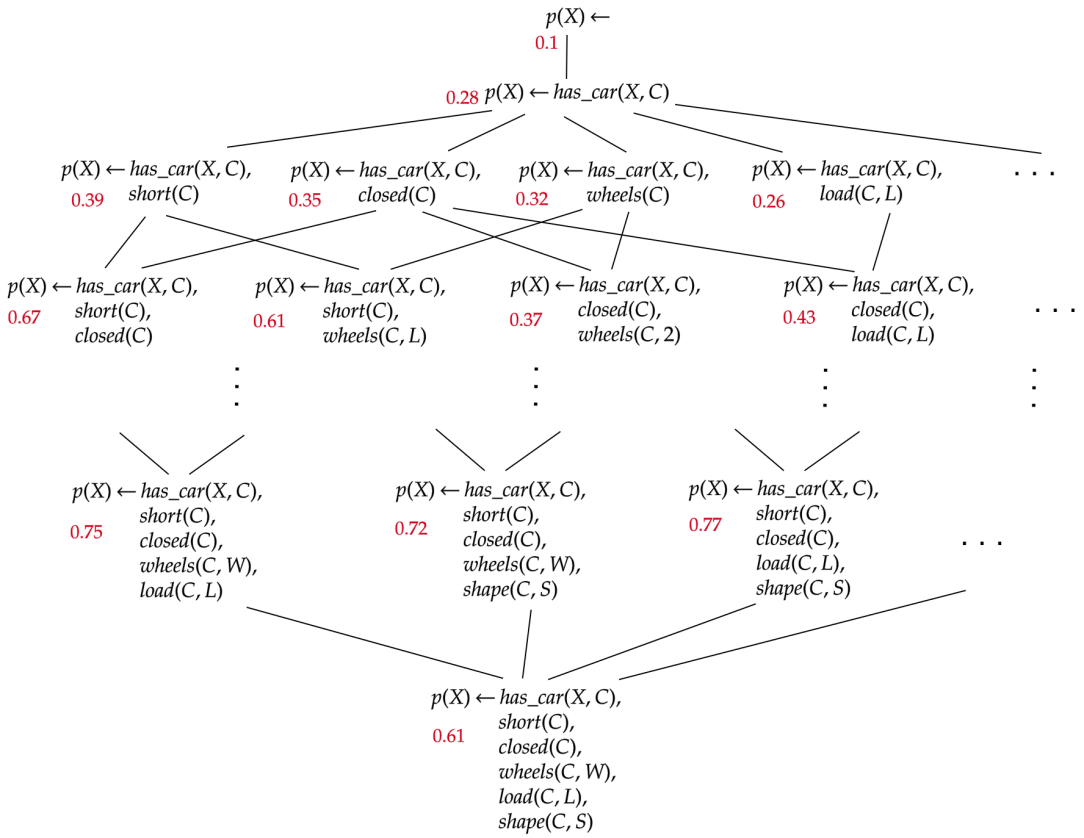

Figure 3.5: The subsumption lattice of relational features for the trains problem. Each feature is associated with a utility score (shown in red colour). Our proposed utility-based sampling strategy selects features from this space.

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.

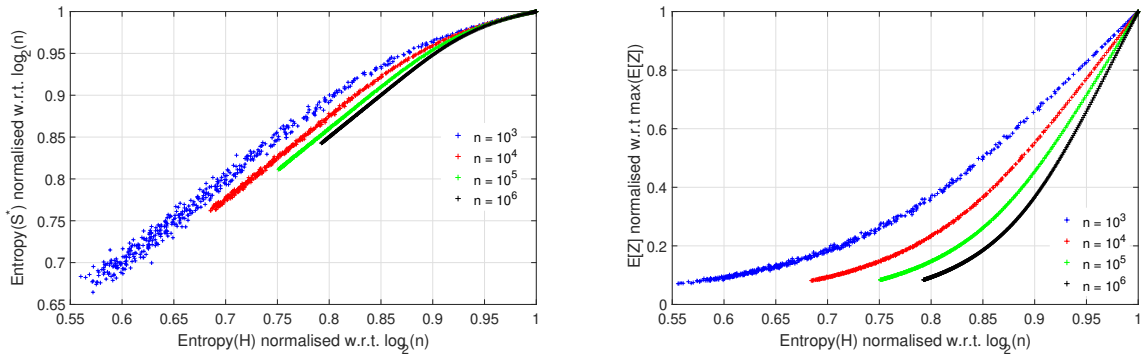

Figure 3.6: Known Hider Distribution: (Left) Entropy of the hider distribution vs. Entropy of the seeker distribution, (Right) Entropy of the hider distribution vs. Expected number of misses by the seeker.

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.

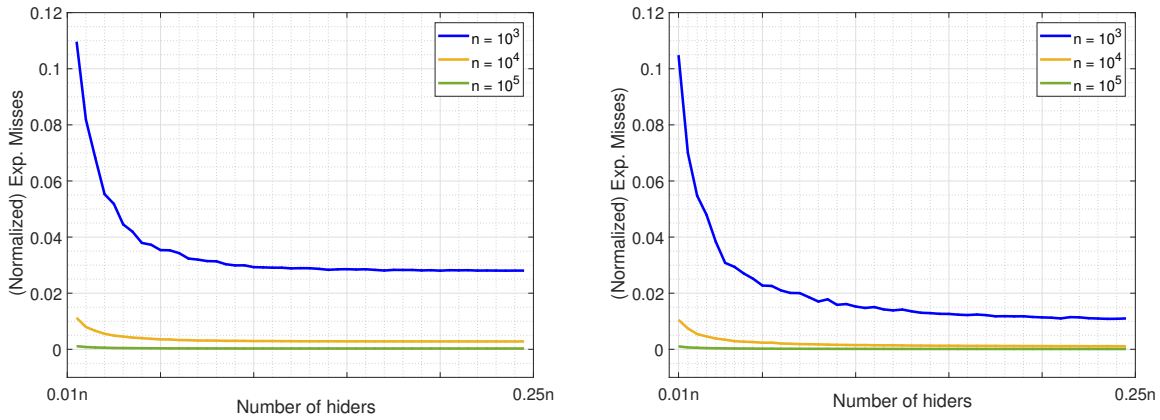

Figure 3.7: Unknown hider distribution, with more than 1 hider: (Left) $p = 0.1$, (Right) $p = 0.25$. That is, the proportion of boxes in the H's partition of the step-approximation is known to be 10% and 25% of n. The number of balls is varied from 1% of n to 25% of n (X-axis). The expected number of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

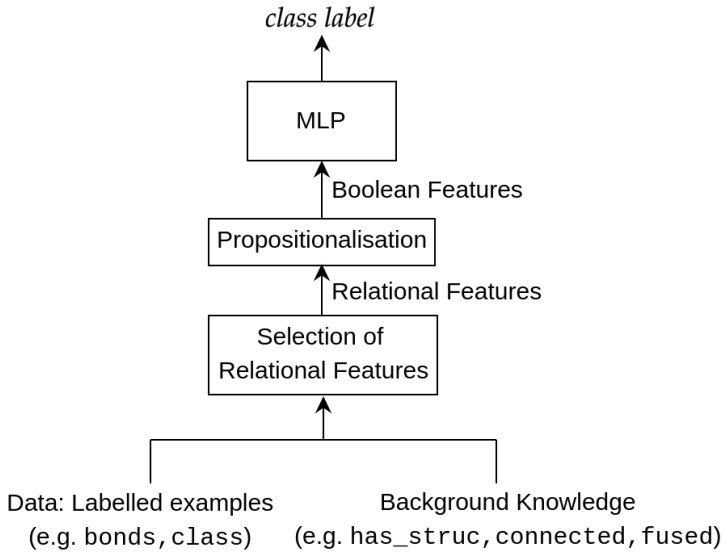

Figure 3.8: Diagrammatic Representation of a Deep Relational Machine (DRM). The examples shown at the bottom are the predicates in data and background knowledge. The selection of relational features includes the feature construction and sampling steps.

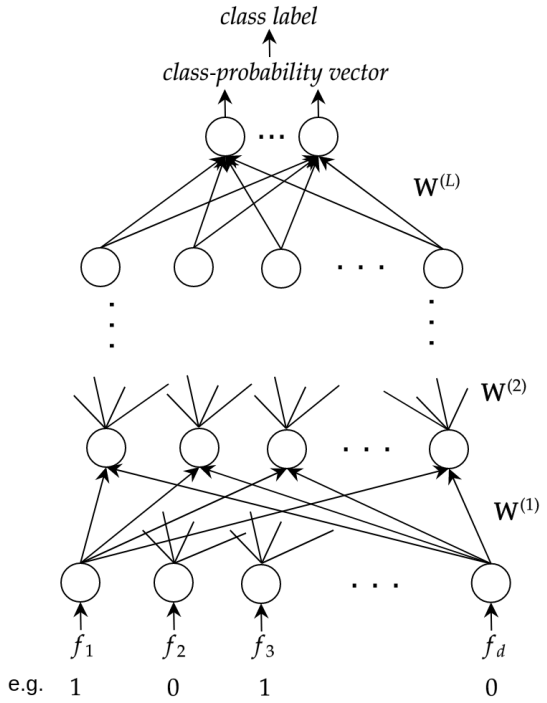

Figure 3.9: Diagrammatic Representation of Constructing a DRM using relational features and propositionalisation. The inputs to an MLP represent a Boolean-valued feature vector obtained by propositionalisation of the relational features $f_1, \ldots, f_d$. The parameters of the MLP are denoted as: $\mathbf{W}^{(\ell)}$, 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.

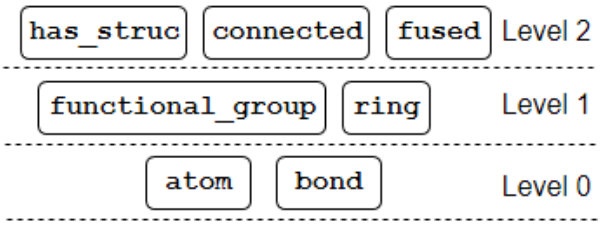

Figure 3.11: Levels of organisation of the background knowledge. Level 0 corresponds to the standard atom and bond information for the molecular compounds; Level 1 refers to the existence of various functional groups and ring structures; Level 2 knowledge is inferred further from Level 0 and 1.

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

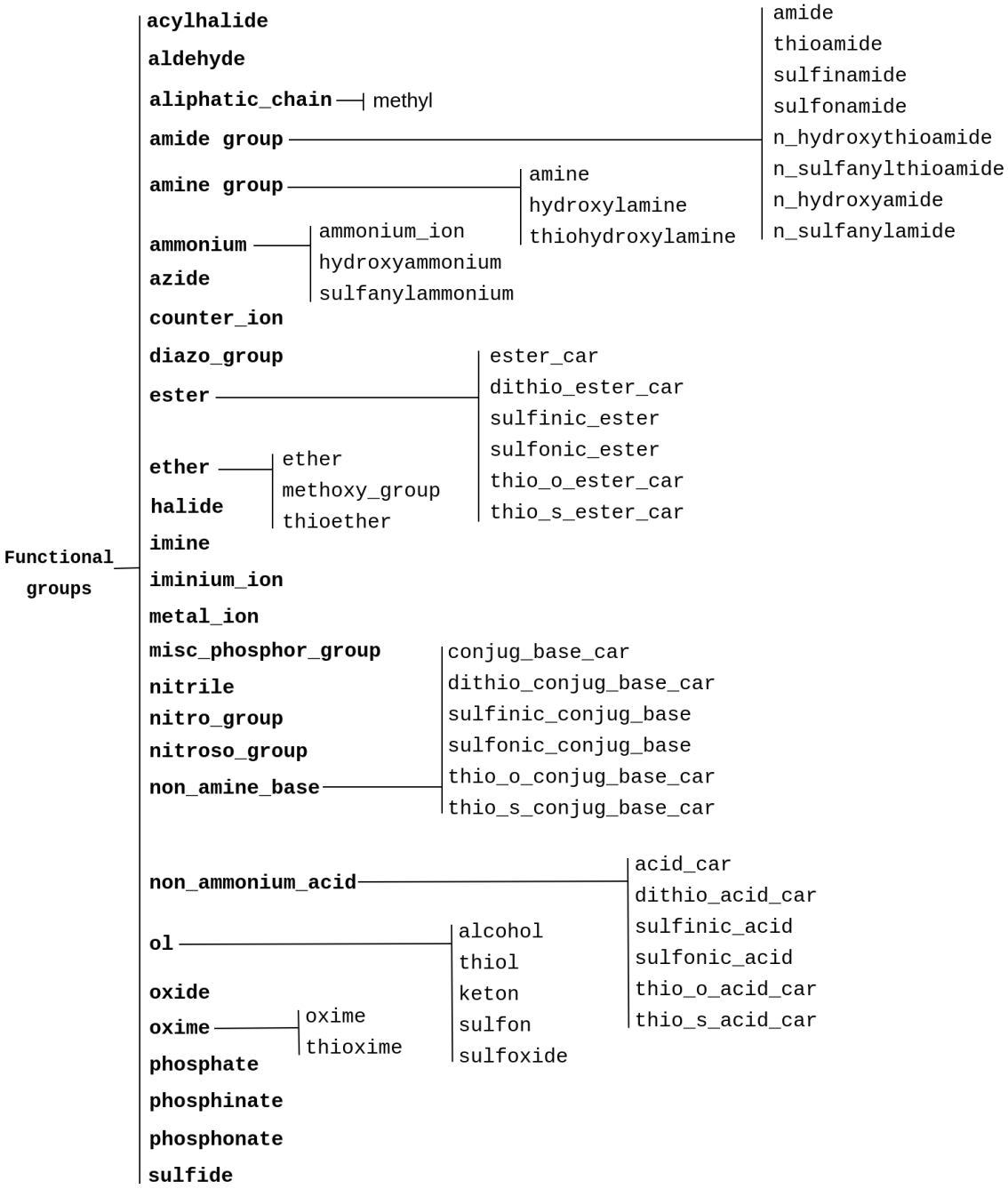

Figure 3.12: Hierarchy of various functional groups in the background knowledge.

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-

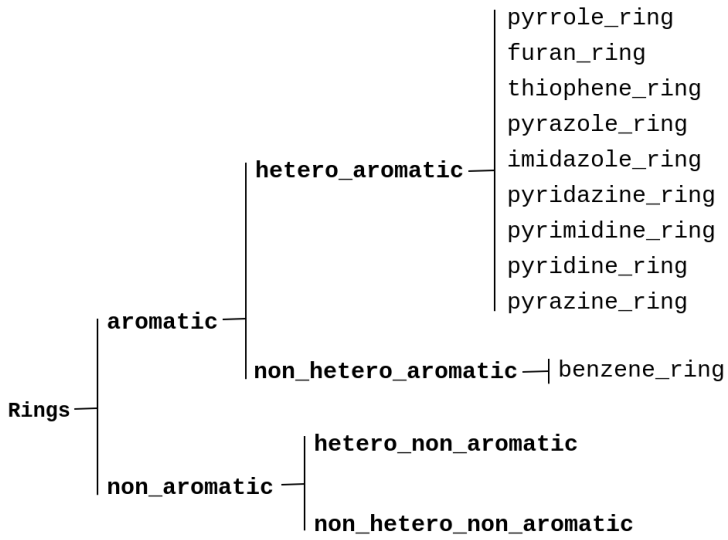

Figure 3.13: Hierarchy of various ring structures in the background knowledge.

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

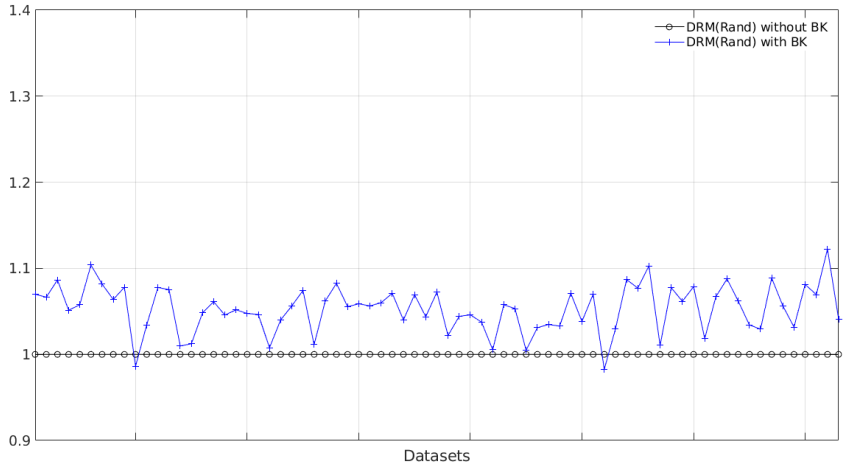

Figure 3.14: Improvements in predictive performance of DRMs, when provided with domain-knowledge through propositionalisation of relational features constructed using simple random sampling strategy by an ILP engine. The average number of relational features across the datasets is roughly 3800. Here 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,

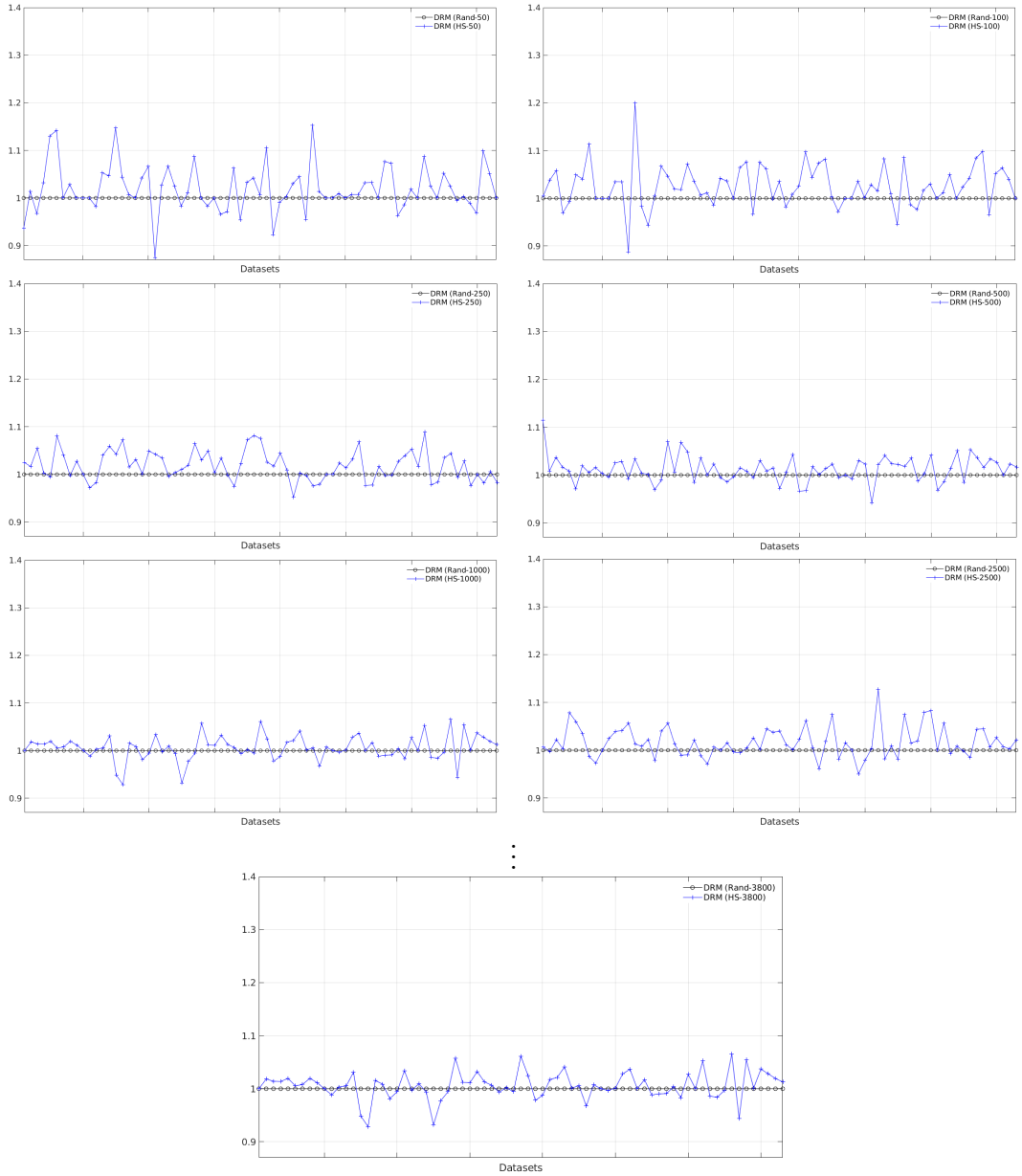

Figure 3.16: Qualitative comparison of predictive performance of DRMs (Hide-and-Seek:"HS" vs Random:"Rand") with different number of relational features: $\{50, 100, 250, 500, 1000, 2500, 3800\}$; The number 3800 is to match the average number of features sampled using simple random sampling. Here X-axis 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.

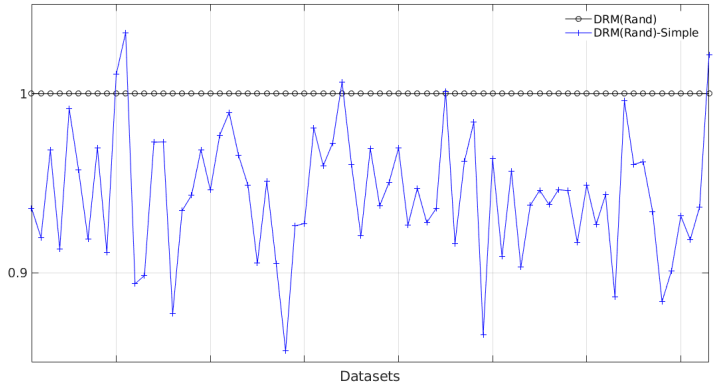

Figure 3.19: Degradation of DRM performance when expressivity of features is decreased from an unrestricted class to the class of relational features obtained using simple features as discussed in [MS98].

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.