Journal of Artificial Intelligence Research 62 (2018) 69-100 Submitted 10/17; published 5/18

Lifted Relational Neural Networks: Efficient Learning of Latent Relational Structures

Gustav Šourek SOUREGUS@FEL.CVUT.CZ
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic

Vojtěch Aschenbrenner V@ASCH.CZ
Faculty of Mathematics and Physics
Charles University
Prague, Czech Republic

Filip Železný ZELEZNY@FEL.CVUT.CZ
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic

Steven Schockaert SCHOCKAERTS1@CARDIFF.AC.UK
School of Computer Science & Informatics
Cardiff University
Cardiff, United Kingdom

Ondřej Kuželka ONDREJ.KUZELKA@KULEUVEN.BE
Department of Computer Science
KU Leuven
Leuven, Belgium

Abstract

We propose a method to combine the interpretability and expressive power of first-order logic with the effectiveness of neural network learning. In particular, we introduce a lifted framework in which first-order rules are used to describe the structure of a given problem setting. These rules are then used as a template for constructing a number of neural networks, one for each training and testing example. As the different networks corresponding to different examples share their weights, these weights can be efficiently learned using stochastic gradient descent. Our framework provides a flexible way for implementing and combining a wide variety of modelling constructs. In particular, the use of first-order logic allows for a declarative specification of latent relational structures, which can then be efficiently discovered in a given data set using neural network learning. Experiments on 78 relational learning benchmarks clearly demonstrate the effectiveness of the framework.

1. Introduction

Lifted models, also known as *templated* models, have recently attracted significant attention in areas such as statistical relational learning (Kimmig, Mihalkova, & Getoor, 2015; De Raedt, Kersting, Natarajan, & Poole, 2016). They essentially define patterns from which specific (ground) models can be derived. For example, a Markov logic network (MLN) model (Richardson & Domingos, 2006) may express that *friends of smokers tend to be* *smokers*. Such lifted patterns are typically encoded as weighted first-order formulas. To make predictions, an MLN is combined with a set of facts about specific individuals to define a ground Markov network; e.g. in an MLN about smokers, these facts may include examples of people who smoke and of people who are in a friendship relation. An important advantage of lifted models is that they can make explicit which symmetries exist in a domain, and thus reduce the number of weights that have to be learned. Another advantage is that the first-order formulas could be provided by domain experts, which can offer a convenient way of guiding the learning process, although the formulas could also be learned from data, e.g. through inductive logic programming methods (De Raedt, 2008).

In this paper, we introduce Lifted Relational Neural Networks (LRNNs), a framework that uses lifted models for specifying feed-forward neural networks. Similar to MLNs, LRNNs are represented as sets of weighted first-order rules. Together with a set of relational facts, these weighted rules then define a standard (or "ground") feed-forward neural network. In particular, for each training or testing example, a separate ground neural network is constructed. The structure of this network is obtained from the grounding of the first-order rules w.r.t. the constants that appear in the example. The weights of the ground network are determined by the weights of the first-order rules. Crucially, the weights of different ground neurons that were constructed from the same first-order rule are tied in our framework, similarly to how weights are shared in lifted graphical models and how weights are tied together in convolutional neural networks. As a result, the weights of the first-order rules can be efficiently learned using stochastic gradient descent.

There have been a few other approaches which adapt neural networks for relational learning (Franca, Zaverucha, & Garcez, 2014; Botta, A, & Piola, 1997; Bader & Hitzler, 2005; Ramon & De Raedt, 2000; Uwents, Monfardini, Blockeel, Gori, & Scarselli, 2011). What distinguishes LRNNs from these previous works is that, following the lifted modelling strategy (Kimmig et al., 2015), we construct a different ground network for each example, exploiting each example's particular relational properties. While recursive Neural Tensor Networks (Socher, Perelygin, Wu, Chuang, Manning, Ng, Potts, et al., 2013) follow a similar strategy, they do so in a much more restricted setting. In particular, they use a tree model structure which exactly follows the tree structure of each example, in a way which is reminiscent of Recursive Auto-Associative Memories (Pollack, 1990).

Many approaches in machine learning rely on finding a latent representation of the objects of interest, e.g. by using probabilistic models, matrix factorization, or neural networks. Neural networks and matrix factorization have proven very effective for learning latent representations, but in their standard form, they cannot be used to find latent representations in relational settings. Probabilistic models have also been used to find latent relational representations, but the scalability of such methods is limited by the fact that they need to run expensive expectation maximization (EM) algorithms (Kok & Domingos, 2007). LRNNs allow us to combine the modelling flexibility of probabilistic models with the effectiveness of neural network learning, to efficiently find different kinds of latent relational structures. While there have already been several works that combine propositional or first-order logic with neural networks (Towell, Shavlik, & Noordewier, 1990; Botta et al., 1997; Franca et al., 2014), to the best of our knowledge, none of these existing methods is able to learn weights of latent non-ground relational structures.

The remainder of the paper is organized as follows. The next section briefly recalls some basic notions from first-order logic and neural networks. Section 3 formally introduces LRNNs, and explains how the weights of an LRNN can be learned and how a ground network can be constructed for classifying a given example. Subsequently, Section 4 illustrates some of the modelling constructs that can be encoded using LRNNs. In Section 5 we then provide an overview of related work. Finally, Section 6 presents our experimental evaluation, after which we conclude the paper.

This paper extends an earlier workshop paper (Šourek, Aschenbrenner, Železný, & Kuželka, 2015a). Since the publication of this workshop paper, a number of approaches have been proposed that are somewhat similar in spirit to LRNNs (Rocktäschel & Riedel, 2016; Cohen, 2016). These approaches, and their differences with LRNNs, will be discussed in Section 5.

2. Preliminaries

Firstly, we recall some basic preliminaries from first-order logic, an extension of which is used as the representation formalism for LRNNs, and neural networks, a model of which forms the learning part of the framework.

2.1 First-Order Logic

We consider a function-free first-order logic, in which formulas are formed in the usual way from a set of constants, a set of variables, a set of n-ary predicates for each $n \in \mathbb{N}$, and the propositional connectives $\vee$, $\wedge$ and $\neg$ (Smullyan, 1995). We will not explicitly write quantifiers, but any variables appearing in formulas will implicitly be assumed to be universally quantified. To avoid confusion, constant symbols will be written in lower case (e.g. *alice*) while variables will be written with a capitalized first letter (e.g. *Person*). A *term* is a constant or a variable. An *atom* is an n-ary predicate symbol, for some $n \in \mathbb{N}$, applied to a tuple of n terms (e.g. $\mathit{friends}(X, \mathit{bob})$). A *ground atom* is an atom which only has constants as arguments (e.g. $\mathit{friends}(\mathit{alice}, \mathit{bob})$). A *literal* is an atom or the negation of an atom; a literal is called *positive* if it is an atom and *negative* otherwise. A clause is a disjunction of literals. A clause containing exactly one positive literal is called a *definite clause*. A definite clause is sometimes also referred to as a *rule*, and a set of definite clauses is sometimes called a *logic program*. To help interpretability, a rule $h \vee \neg b_1 \vee \cdots \vee \neg b_k$ will usually be written as $h \leftarrow b_1 \wedge \cdots \wedge b_k$, as is common in the context of logic programming. We refer to the literal h as the *head* of the rule and the conjunction $b_1 \wedge \cdots \wedge b_k$ as the *body*. A clause which consists of a single atom is also called a *fact*.

The *Herbrand base* of a set of first-order formulas $\mathcal{P}$ is the set of all ground atoms which can be constructed using the constants and predicates that appear in this set (while respecting the arity of each predicate). A *Herbrand interpretation* of $\mathcal{P}$, also called a *possible world*, is a mapping that assigns a truth value to each element from $\mathcal{P}$'s Herbrand base. We say that a possible world I satisfies a ground atom F, written $I \models F$, if $F \in I$. The satisfaction relation is then generalized to arbitrary ground formulas in the usual way. A set of ground formulas is satisfiable if there exists at least one possible world in which all formulas from the set are true; such a possible world is called a *Herbrand model*. Each set of definite clauses has a unique Herbrand model that is minimal w.r.t. the subset relation, called its least Herbrand model. The least Herbrand model of a finite set of ground definite clauses can be constructed in a finite number of steps using the *immediate-consequence operator* (Van Emden & Kowalski, 1976). This immediate consequence operator is a mapping T_p from Herbrand interpretations to Herbrand interpretations, defined for a set of ground definite clauses $\mathcal{P}$ as $T_p(I) = \{h \,|\, (h \leftarrow b_1 \wedge \cdots \wedge b_k) \in \mathcal{P}, \{b_1, ..., b_k\} \subseteq I\}$. In other words the operator T_p expands the current set of true atoms (i.e. the current Herbrand interpretation I) with their immediate consequences as prescribed by the rules in $\mathcal{P}$.

Now consider a set of non-ground definite clauses $\mathcal{P}$. The *grounding* of a clause α from $\mathcal{P}$ is the set of ground clauses $G(\alpha) = \{\alpha\theta_1, ..., \alpha\theta_n\}$ where $\theta_1, ..., \theta_n$ is the set of all possible substitutions, each mapping the variables occurring in α to constants appearing in $\mathcal{P}$. Note that if α is already ground, its grounding is a singleton. The grounding of $\mathcal{P}$ is given by $G(\mathcal{P}) = \bigcup_{\alpha \in \mathcal{P}} G(\alpha)$. The least Herbrand model of $\mathcal{P}$ is then defined as the least Herbrand model of $G(\mathcal{P})$. In practice, most of the rules in the grounding $G(\mathcal{P})$ will be irrelevant, as their body can never be satisfied. The *restricted grounding* limits the grounding to those rules which are "active", i.e. whose body is satisfied in the least Herbrand model $\mathcal{H}$. It is defined by $G^R(\mathcal{P}) = \{h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta \,|\, (h \leftarrow b_1 \wedge \cdots \wedge b_k) \in \mathcal{P} \text{ and } \{h\theta, b_1\theta, \ldots, b_k\theta\} \subseteq \mathcal{H}\}$.

2.2 Artificial Neural Networks

An artificial neural network (NN) is a biologically inspired mathematical model, consisting of interconnected processing units called *neurons*, each of which is associated with an activation function $g_i \in \mathcal{G}$ from some predefined family of differentiable functions. The neural network then defines a mapping $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ of input space to target space vectors, parameterized by a set of weights $w^l_j \in \mathbb{R}$. Following the pattern of neural interconnections, the mapping f can be seen as a composition of activation functions $g_i \in G$. For feed-forward neural networks, the mapping f typically corresponds to a hierarchical compound of non-linear weighted sums $g_i(\sum_j w^l_j g_j(\sum_k w^{l+1}_k g_k(\ldots)))$, which can be conveniently depicted as a weighted directed acyclic graph of neurons. By adapting the weights $w^i_j \in \mathcal{W}$, the model can be trained to approximate some target function $t : \mathbb{R}^m \mapsto \mathbb{R}^n$. This is typically performed by some sort of gradient descent minimization of a given cost function $cost : \{\mathcal{W}, \mathcal{D}\} \mapsto \mathbb{R}$ capturing the discrepancy between f and t for some set of training samples $(x_d, t(x_d)) \in \mathcal{D}$.

3. Lifted Relational Neural Networks

In this section, we formally introduce the framework of LRNNs. We define the representation language, describe the translation into neural models and how to use them for inference. We then discuss variant semantics with the choice of activation functions and negation, and detail the learning process.

3.1 Definition

A lifted relational neural network (LRNN) $\mathcal{N}$ is a set of weighted definite clauses, i.e. a set of pairs (R_i, w_i) where R_i is a definite clause and $w_i \in \mathbb{R}$. For an LRNN $\mathcal{N}$, we write $\mathcal{N}^*$ to denote the corresponding set of definite clauses, i.e. $\mathcal{N}^* = \{C \,|\, (C, w) \in \mathcal{N}\}$. The grounding $\overline{\mathcal{N}}$ of an LRNN $\mathcal{N}$ is defined in terms of the restricted grounding of $\mathcal{N}^*$.

Specifically, we define $\overline{\mathcal{N}} = \{(C\theta, w) \mid (C, w) \in \mathcal{N}, C\theta \in G^R(\mathcal{N}^*)\}$. As already mentioned in the introduction, LRNNs are seen as templates for creating *ground* neural networks. These networks will (among others) contain a node for each considered ground clause. Since it is clearly beneficial to keep the networks as simple as possible, it is thus important to avoid including any ground clauses that are not relevant (i.e. those that are not active in the least Herbrand model). The restricted grounding of $\mathcal{N}^*$ contains exactly those clauses that are relevant (i.e. those that are active in the Herbrand model).

Example 1 *Let the LRNN $\mathcal{N}$ be defined as follows:*

$$\begin{aligned}\mathcal{N} =\{&(\mathit{mother}(C, M) \leftarrow \mathit{parent}(C, M) \wedge \mathit{female}(M), 1),\\ &(\mathit{father}(C, F) \leftarrow \mathit{parent}(C, F) \wedge \mathit{male}(F), 2),\\ &(\mathit{female}(\mathit{alice}), 1), (\mathit{parent}(\mathit{bob}, \mathit{alice}), 1), (\mathit{parent}(\mathit{eve}, \mathit{alice}), 1)\}.\end{aligned}$$

The grounding $\mathcal{N}$ is then given by:

$$\begin{aligned}\overline{\mathcal{N}} =\{&(\mathit{mother}(\mathit{bob}, \mathit{alice}) \leftarrow \mathit{parent}(\mathit{bob}, \mathit{alice}) \wedge \mathit{female}(\mathit{alice}), 1),\\ &(\mathit{mother}(\mathit{eve}, \mathit{alice}) \leftarrow \mathit{parent}(\mathit{eve}, \mathit{alice}) \wedge \mathit{female}(\mathit{alice}), 1),\\ &(\mathit{female}(\mathit{alice}), 1), (\mathit{parent}(\mathit{bob}, \mathit{alice}), 1), (\mathit{parent}(\mathit{eve}, \mathit{alice}), 1)\}.\end{aligned}$$

Note that $\overline{\mathcal{N}}$ does not contain the predicates male/1 or father/2 as they do not appear in least Herbrand model of $\mathcal{N}$.

The neural network corresponding to an LRNN $\mathcal{N}$ is constructed as follows.

3.1.1 Neurons

The neurons that appear in the network correspond to logical constructs, such as atoms, facts and rules. Specifically, the network contains the following types of neurons:

- For each ground atom h occurring in $\overline{\mathcal{N}}$, there is a neuron A_h, called an *atom neuron*.
- For each ground fact $(h, w) \in \overline{\mathcal{N}}$, there is a neuron $F_{(h,w)}$, called a *fact neuron*.
- For every ground rule $(c\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w) \in \overline{\mathcal{N}}$, there is a neuron $R_{(c\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w)}$, called a *rule neuron*.
- For every (possibly non-ground) rule $(c \leftarrow b_1 \wedge \cdots \wedge b_k, w) \in \mathcal{N}$ and every grounding $h = c\theta$ of c that occurs in $\mathcal{H}$, there is a neuron $Agg^h_{(c \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$, called an *aggregation neuron*.

3.1.2 Weights and Connections

We now describe how the different neurons are connected, and how their outputs are defined. Intuitively, the neural network computes for each ground atom h a truth value, which is encoded by the output of the atom neuron A_h. To obtain these truth values, the network

Type of neuron	Notation	Output
Atom neuron	A_h	$g_\vee(o(F_{(h,w)}), o(Agg^h_{\alpha_1}), ..., o(Agg^h_{\alpha_m}))$
Fact neuron	$F_{(h,w)}$	w
Rule neuron	$R_{(c\theta \leftarrow b_1\theta \wedge ... \wedge b_k\theta, w)}$	$g_\wedge(o(A_{b_1\theta}), ..., o(A_{b_1\theta}))$
Aggregation neuron	$Agg^h_{(c \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$	$w \cdot g_*(o(R_{\alpha_1}), ..., o(R_{\alpha_m}))$

Table 1: Overview of the process of constructing a neural network from a given LRNN $\mathcal{N}$.

propagates values in a way which closely mimics the immediate consequence operator[1]. In particular, when using the immediate consequence operator, there are two ways in which h can become true: if h corresponds to a fact, or if h is the head of a rule whose body is already satisfied. Similarly, the inputs of the atom neuron A_h consist of the fact neurons of the form $F_{(h,w)}$ and aggregation neurons of the form $Agg^h_{(c \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$. The output of an atom neuron with inputs $i_1, ..., i_m$ is given by $g_\vee(i_1, ..., i_m)$, where $g_\vee$ is an activation function that maps the inputs to a real-valued output. Different choices are possible for $g_\vee$, as will be discussed in detail in Section 3.2.

A fact neuron $F_{(h,w)}$ has no input and has the value w as its output. The output of the aggregation neuron $Agg^h_{(c \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$ intuitively expresses how strongly h can be derived using the rule $c \leftarrow b_1 \wedge \cdots \wedge b_k$. Note that there can be several groundings of the rule that have the atom h in the head, when the body of the rule contains variables that do not appear in the head. This is why we need to distinguish rule neurons, which correspond to individual groundings of rules, from aggregation neurons, which group together all groundings of a rule that have the same head. Specifically, the inputs of the aggregation neuron $Agg^h_{(c \leftarrow b_1 \wedge \cdots \wedge b_k)}$ are all rule neurons $R_{(c\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w)}$ for which $c\theta = h$. The output of this aggregation neuron is given by $w \cdot g_*(i_1, ..., i_m)$, where $i_1, ..., i_m$ are its inputs, g_* is an activation function, and w is the weight of the corresponding rule. Note that while we will assume throughout this paper that the weight of a rule and the value $g_*(i_1, ..., i_m)$ are combined using multiplication, in principle other combination operators are also possible. The rule neuron $R_{(c\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w)}$ intuitively needs to fire if the atoms $b_1\theta, ..., b_k\theta$ are all true. Accordingly, its inputs $i_1, ..., i_k$ are given by the atom neurons $A_{b_1\theta}, ..., A_{b_k\theta}$, and its output is $g_\wedge(i_1, ..., i_k)$, with $g_\wedge$ being a third type of activation function. An overview of the different types of neurons and their interconnections is shown in Table 1, where we write $o(N)$ to denote the output of neuron N.

Throughout this paper, we will only consider LRNNs whose associated ground neural network is feed-forward. It is easy to see that this is the case when the rules in $\mathcal{N}^*$ are free from cycles, i.e. when there exists a strict ordering $\prec$ of the predicates such that for each predicate p_2 occurring in the body of a rule with predicate p_1 in the head, it holds that $p_1 \prec p_2$. Note, however, that there are many cases where the logic program $\mathcal{N}^*$ associated with an LRNN contains cycles, but where the resulting ground neural network does not. For instance, rules defining directed paths in acyclic graphs would not lead to directed cycles in the resulting ground neural networks, despite the fact that the rules in the LRNN itself

[1] In fact, it is possible to precisely characterize the behavior of the neural network by using extensions of the immediate consequence operator for multi-valued logics (Achs & Kiss, 1995; Damásio & Pereira, 2001b).

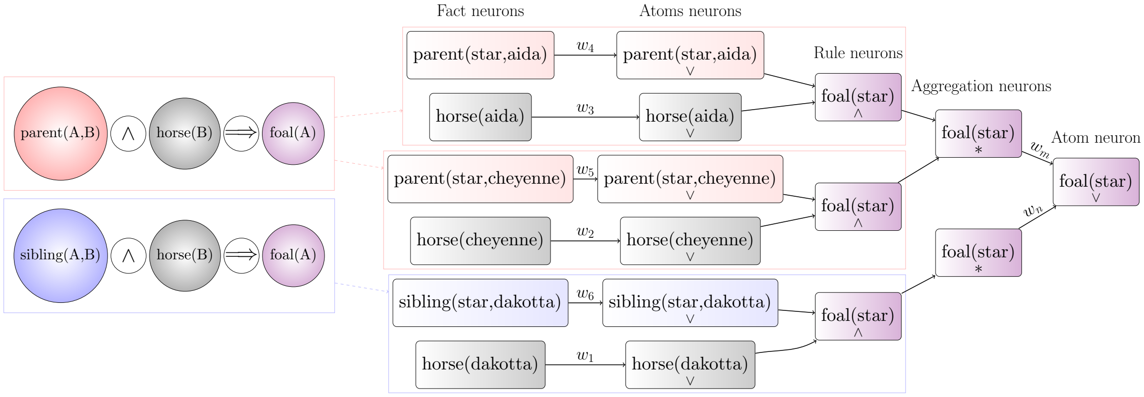

Figure 1: Left: the LRNN $\mathcal{N}$ from Example 2, omitting the ground facts. Right: the corresponding ground neural network. Atom neurons are denoted by "$\vee$", rule neurons by "$\wedge$" and aggregation neurons by "$*$", the remaining neurons are fact neurons.

would clearly be recursive. In general, however, when grounding an LRNN with cycles, we may end up with recurrent neural networks, which are more difficult to train.

Example 2 *Let us consider the following LRNN:*

$$\begin{aligned}\mathcal{N} = \{&(foal(A) \leftarrow parent(A,B) \wedge horse(B), w_m), (foal(A) \leftarrow sibling(A,B) \wedge horse(B), w_n),\\ &(horse(dakotta), w_1), (horse(cheyenne), w_2), (horse(aida), w_3),\\ &(parent(star, aida), w_4), (parent(star, cheyenne), w_5), (sibling(star, dakotta), w_6)\}.\end{aligned}$$

Figure 1 shows the LRNN $\mathcal{N}$ from the example, together with its ground neural network. To visualize the creation of the neural network, colors are used to denote unique predicate signatures, while rectangles group the neurons that have been derived from the same ground rule.

3.1.3 LRNNs as Neural Network Templates

To use LRNNs in practice, we typically start from a set of rules $\mathcal{P}$ and some labelled examples. The labelled examples are used to learn weights for the general rules in $\mathcal{P}$, as will be explained in Section 3.4. This process leads to a trained LRNN $\mathcal{N}$ which contains weighted rules, but does not contain any ground facts. Each time we want to use this LRNN, we then first add a set of weighted ground facts $\mathcal{M}$ that describe the specific example about which we want to make a prediction. In this way, for each prediction we make, a different ground neural network is constructed, with a potentially very different structure. This process makes LRNNs similar in spirit to lifted graphical models, and rather different from normal neural network frameworks.

Example 3 *We consider an LRNN $\mathcal{N}$ containing rules for predicting the toxicity of molecules, based on conformations of the atoms contained in them and their bonds. For example, it* *may be beneficial to assign atoms to some latent groups. Assuming we want to consider two latent groups, this can be accomplished using the following rules:*

$$\begin{array}{llll} w_{o_1} : gr_1(X) \leftarrow O(X) & w_{n_1} : gr_1(X) \leftarrow N(X) & \ldots & w_{h_1} : gr_1(X) \leftarrow H(X) \\ w_{o_2} : gr_2(X) \leftarrow O(X) & w_{n_2} : gr_2(X) \leftarrow N(X) & \ldots & w_{h_2} : gr_2(X) \leftarrow H(X) \end{array}$$

The weight w_{o_1}, for instance, represents the degree to which oxygen atoms (O) belong to the first latent group. This membership degree is learned based on training data, i.e. all we need to specify is that we want to consider two latent groups as the basis for making predictions and that none of the atoms $O, N, ..., H$ is excluded a priori from belonging to these groups. From the latent groups of atoms, we can now construct relational patterns, such as small chains, trees or general graphlets. For instance, the following rule describes a small relational pattern where an atom from gr_1 is connected to an atom from gr_2 through a bond (represented by the predicate b):

$$w_{f_1} : toxic \leftarrow gr_1(A) \wedge b(A, B) \wedge gr_2(B) \tag{1}$$

In general, there would be different rules with toxic in the head, each encoding a different relational pattern in the body. Which of these relational patterns is actually predictive of toxicity is then again learned from training data. For example, if (1) was found to be predictive, then w_{f_1} would receive a high value after training; otherwise, it might be set as 0.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two sets of (weighted) facts, each describing a given molecule, i.e. the particular conformations of specific atoms and bonds. To use the LRNN $\mathcal{N}$ for predicting the toxicity of these molecules (after its weights have been trained), we construct the ground networks of $\overline{\mathcal{N} \cup \mathcal{M}_1}$ and $\overline{\mathcal{N} \cup \mathcal{M}_2}$, and for both of the resulting ground neural networks we compute the output of an atom neuron corresponding to the literal toxic. The ground neural networks for two example molecules are displayed in Fig. 2. As can be seen from this figure, the neural networks for the two cases are different in both size and structure, which is a result of the fact that $\mathcal{N}^* \cup \mathcal{M}_1^*$ and $\mathcal{N}^* \cup \mathcal{M}_2^*$ have different Herbrand models.

3.2 Activation Functions

The behavior of an LRNN crucially depends on how the activation functions $g_\wedge$, $g_\vee$ and g_* are chosen. Intuitively, $g_\wedge$ should behave like a conjunction, in the sense that the atom neuron for h should only receive a high weight from the rule neuron for $h \leftarrow b_1 \wedge \cdots \wedge b_k$ if the atom neurons for $b_1, ..., b_k$ all have a high output value. Similarly, $g_\vee$ and g_* should intuitively behave like disjunctions; the reason why we need two types of disjunctive activation functions will become clear below. One possibility for choosing the activation functions is to draw inspiration from the field of fuzzy logic, where extensions of logical connectives to real-valued arguments have been widely studied (Klement, Mesiar, & Pap, 1997). In particular, if all input weights are between 0 and 1, such fuzzy logic connectives could straightforwardly be used. For example, in accordance with Gödel logic, we could choose the activation functions as follows:

$$\begin{aligned} g_\wedge(b_1, ..., b_k) &= \min(b_1, ..., b_k) \\ g_*(b_1, ..., b_m) = g_\vee(b_1, ..., b_m) &= \max(b_1, ..., b_k) \end{aligned}$$

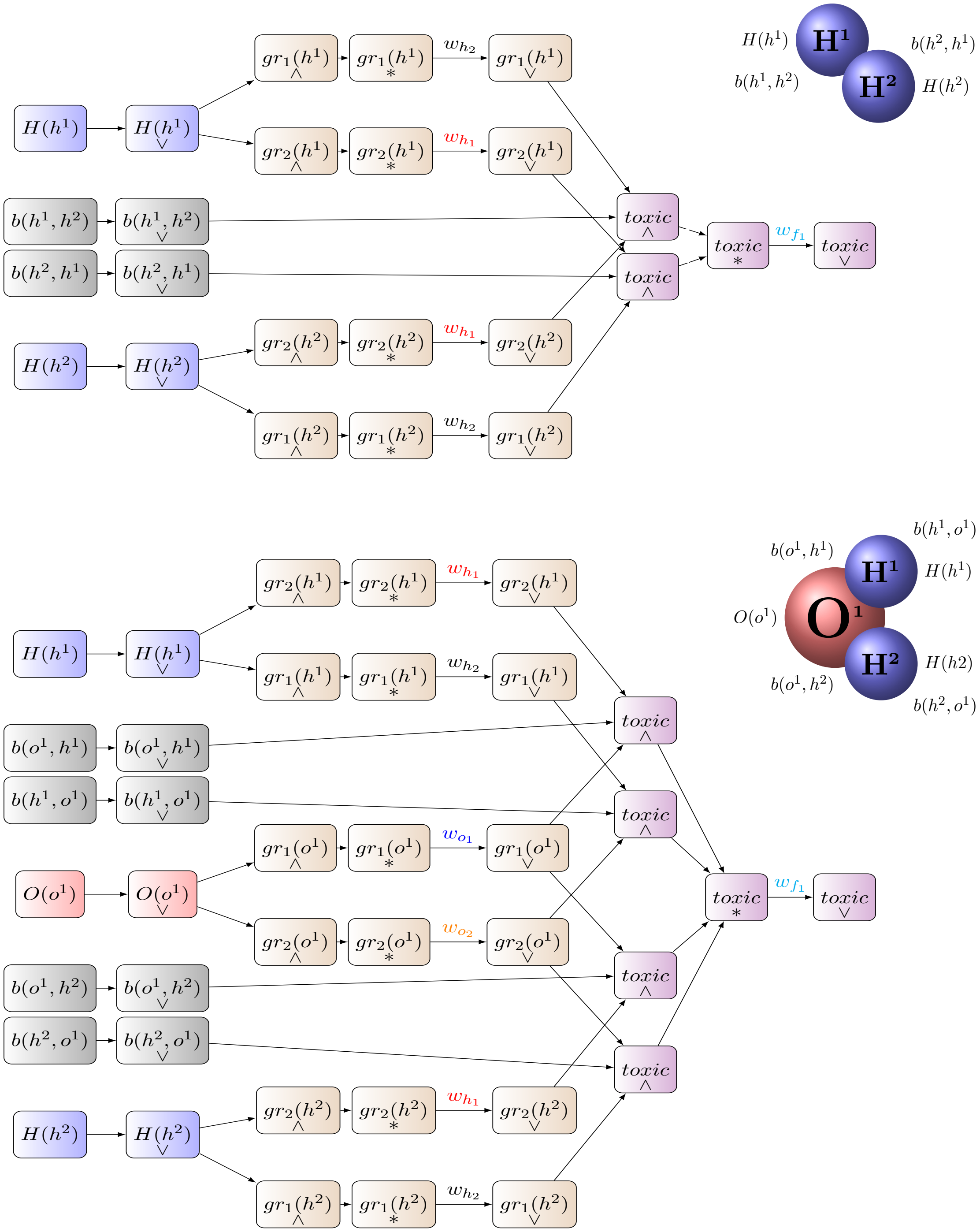

Figure 2: The two neural networks $\overline{\mathcal{N} \cup \mathcal{M}_1}$ and $\overline{\mathcal{N} \cup \mathcal{M}_2}$ grounded from the single LRNN $\mathcal{N}$ and the two example molecules from Example 3. Atom neurons are denoted by "$\vee$", rule neurons by "$\wedge$" and aggregation neurons by "$*$", the remaining neurons are fact neurons.

Alternatively, using the connectives from product logic, we obtain:

$$g_\wedge(b_1, ..., b_k) = b_1 \cdot ... \cdot b_k$$
$$g_*(b_1, ..., b_m) = g_\vee(b_1, ..., b_m) = 1 - (1 - b_1) \cdot ... \cdot (1 - b_k)$$

Another popular alternative are the Łukasiewicz connectives:

$$g_\wedge(b_1, ..., b_k) = \max(b_1 + ... + b_k - k + 1, 0)$$
$$g_\vee(b_1, ..., b_m) = \min(b_1 + ... + b_m, 1)$$
$$g_*(b_1, ..., b_m) = \max(b_1, ..., b_k)$$

Note that $g_\vee$ and g_* in this case correspond to the two types of disjunctions that are used in Łukasiewicz logic. An advantage of using fuzzy logic connectives is that LRNNs can then be seen as fuzzy logic programs. In particular, the predictions of an LRNN $\mathcal{N}$ are then precisely given by the truth degrees of the corresponding atoms in the least Herbrand model of $\mathcal{N}$, viewed as a fuzzy logic program. However, using these fuzzy logic connectives also has two important drawbacks. First, gradient-based learning is considerably less effective with such operations, compared to e.g. sigmoidal activation functions. Second, it would be useful to consider parametrized families of activation functions, such that a specific activation function could be chosen based on training data. The latter issue could in principle be addressed by using continuous families of fuzzy logic connectives, such as the Frank family of t-norms, which is parametrized by a real value, and has the three aforementioned examples of fuzzy logic conjunctions as special cases. However, as this would further complicate gradient-based learning, in this paper we will focus on more standard activation functions, and sigmoidal functions in particular.

Specifically, we will consider two classes of activation functions, which will be useful in slightly different types of applications. The first class is defined as follows.

Definition 1 (Max-Sigmoid Activation Functions) *The Max-Sigmoid (MS) collection of activation functions are defined as:*

$$g_\wedge(b_1, \ldots, b_k) = sigm\left(a \cdot \left(\sum_{i=1}^{k} b_i - k + 1 + b_0\right)\right)$$
$$g_*(b_1, \ldots, b_m) = \max_i b_i$$
$$g_\vee(b_1, \ldots, b_k) = sigm\left(a \cdot \left(\sum_{i=1}^{k} b_i + b_0\right)\right)$$

where $a, b_0 \in \mathbb{R}$ are parameters.

The parameters a and b_0 are typically fixed, although they could also be determined using training data (see Section 3.4). When learned from training data, they help to ensure that we select an activation function that is appropriate for the domain being modelled. When the inputs are between 0 and 1, the parameters a and b_0 in the definition of $g_\wedge$

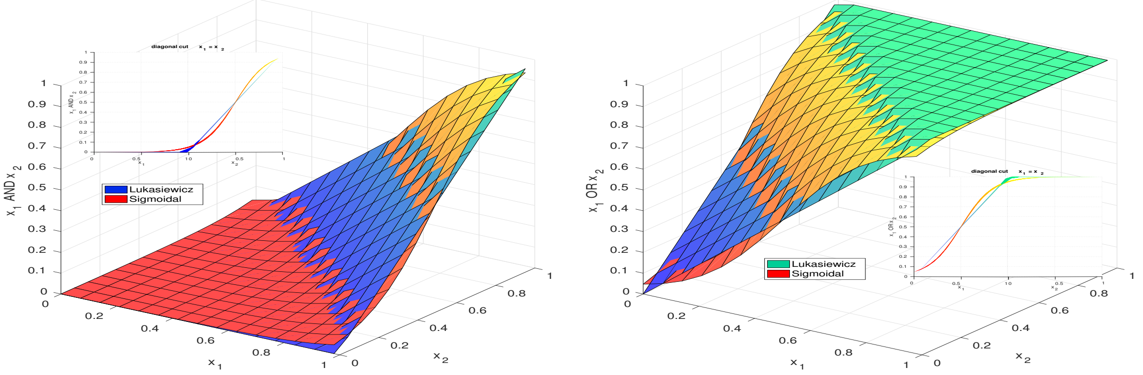

Figure 3: A crude approximation of Łukasiewicz conjunction (left) and disjunction (right) by respective sigmoidal activation functions for the use in LRNNs. A diagonal view of $x_1 = x_2$ is embedded to detail the level of approximation of each operator.

and $g_\vee$ can be chosen such that these activation functions approximate the Łukasiewicz connectives. To illustrate this, the functions $\mathit{sigm}\left(a \cdot (b_1 + \cdots + b_k - k + 1 + b_0)\right)$ and $a \cdot \mathit{sigm}\left(a \cdot (b_1 + \cdots + b_k + b_0)\right)$, with $b_0 = -0.5$ and $a = 6$, are compared with the Łukasiewicz conjunction and disjunction, respectively, in Figure 3. The g_* activation function groups different groundings of the same rule with the same head. If we think of the bodies of these rules as patterns, choosing g_* as the maximum means that we are simply looking for the best match. This view is illustrated in the following example.

Example 4 *Let us consider the following LRNN with the activation functions from the Max-Sigmoid family:*

$$\begin{aligned}\mathcal{N} = \{&(\mathit{hasBrightEdge} \leftarrow \mathit{isBright}(E), 1),\\ &(\mathit{isBright}(E) \leftarrow \mathit{edge}(E, U, V) \wedge \mathit{bright}(U) \wedge \mathit{bright}(V), 1),\\ &(\mathit{bright}(U) \leftarrow \mathit{yellow}(U), 2), (\mathit{bright}(U) \leftarrow \mathit{red}(U), 1), (\mathit{bright}(U) \leftarrow \mathit{blue}(U), 0.5)\}.\end{aligned}$$

Note that hasBrightEdge is a predicate of arity 0. Let us also consider a set $\mathcal{G}$ describing a specific graph with colored vertices.

$$\begin{aligned}\mathcal{G} = \{&(\mathit{edge}(e_1, v_1, v_2), 1), (\mathit{edge}(e_2, v_2, v_3), 1), (\mathit{edge}(e_3, v_3, v_4), 1), (\mathit{edge}(e_4, v_4, v_1), 1),\\ &(\mathit{red}(v_1), 1), (\mathit{blue}(v_2), 1), (\mathit{yellow}(v_3), 1), (\mathit{yellow}(v_4), 1)\}\end{aligned}$$

The output of the atom neuron $A_{\mathit{hasBrightEdge}}$ only depends on the "brightest edge", which in this case is e_3. This is because the aggregation function g_* is $g_*(b_1, \ldots, b_m) = \max_i b_i$. Note that any colored graph that contains an edge connecting two yellow vertices would lead to the same output. Similar kinds of LRNNs could be useful in practice, for instance, to detect molecules which contain a substructure that is similar to a prescribed pattern. Instead of colors, we would then model physico-chemical properties of atoms (e.g. partial charge) and instead of graph edges we would describe bonds in molecules.

In the next example, we illustrate how using a sigmoidal function for $g_\vee$ leads us intuitively to accumulate the evidence coming from different rules with the same head.

Example 5 *Consider the following LRNN with the activation functions from the Max-Sigmoid family:*

$$\begin{aligned}\mathcal{N} =&\{(\mathit{highPressure}(X) \leftarrow \mathit{stressed}(X), 1), (\mathit{highPressure}(X) \leftarrow \mathit{obese}(X), 1),\\ &(\mathit{highPressure}(X) \leftarrow \mathit{exercises}(X), -1)\}\end{aligned}$$

and the set of weighted facts $\mathcal{P} = \{(\mathit{stressed}(\mathit{alice}), 1), (\mathit{obese}(\mathit{alice}), 1), (\mathit{stressed}(\mathit{bob}), 1), (\mathit{exercises}(\mathit{bob}), 1)\}$. *Outputs of aggregation neurons for rules with the same head are combined using the activation function* $g_\vee$. *In this particular example, if we construct the ground LRNN of* $\overline{\mathcal{N} \cup \mathcal{P}}$ *then the output of the atom neuron* $A_{\mathit{highPressure}(\mathit{alice})}$ *will be higher than the output of the atom neuron* $A_{\mathit{highPressure}(\mathit{bob})}$, *because alice is stressed and obese whereas bob is just stressed and exercises.*

We now give an example of a scenario where the Max-Sigmoid class of activation functions is not appropriate.

Example 6 *Let us consider the following simple LRNN for predicting which individuals are infected with the flu:*

$$\mathcal{N} = \{(\mathit{hasFlu}(A) \leftarrow \mathit{friends}(A, B) \wedge \mathit{hasFluDiagnosed}(B), 1)\}$$

and a set of weighted ground facts $\mathcal{P}$ *about a group of people and their friendships. If we constructed the ground neural networks of* $\overline{\mathcal{N} \cup \mathcal{P}}$ *using the activation functions from the Max-Sigmoid family then the prediction of whether an individual has the flu would be entirely based on the existence of at least one person who was already diagnosed with the flu. It would obviously be more meaningful to base the predictions on the fraction of one's friends who were diagnosed with the flu.*

The next definition introduces a family of activation functions that are appropriate for situations such as the one in the previous example.

Definition 2 (Avg-Sigmoid Activation Functions) *The Avg-Sigmoid (AS) collection of activation functions are defined as:*

$$\begin{aligned}g_\wedge(b_1, \dots, b_k) &= \mathit{sigm}\left(a \cdot \left(\sum_{i=1}^{k} b_i - k + b_0\right)\right)\\ g_*(b_1, \dots, b_m) &= \frac{1}{m}\sum_{i=1}^{m} b_i\\ g_\vee(b_1, \dots, b_k) &= \mathit{sigm}\left(a \cdot \left(\sum_{i=1}^{k} b_i + b_0\right)\right)\end{aligned}$$

An advantage of the Avg-Sigmoid family over the Max-Sigmoid family is that the functions from the Avg-Sigmoid family are everywhere differentiable, which simplifies learning. Of course, a wide variety of other families of activation functions can be used for LRNN learning, but in this paper we will limit ourselves to the two considered families.

Finally note that, in principle, a different activation function could be used for every neuron. For example, we may have some rules that are aimed at detecting the existence of a pattern, suggesting the use of a maximum, while for other rules we may want to accumulate the evidence provided by different rules, suggesting the use of a summation or average. We could also consider a setting where the different activation functions are learnable, e.g. by tuning the value of the parameter b_0. Note, however, that the latter could also be simulated in the basic framework, by adding an additional atom to each rule body and learning the weight of that atom instead.

3.3 Negation As Failure

There is a close connection between LRNNs, on the one hand, and (multi-valued extensions of) logic programs, on the other hand. However, from a logic programming point of view, the syntax of LRNNs is quite limited, as negation is not considered. In particular, *negation-as-failure* is a central notion in most logic programming frameworks (Lloyd, 2012). The idea is to use rules such as $a \leftarrow b \wedge \textbf{not } c$, which intuitively encodes that we can derive a if b is true, unless we can prove that c is true (i.e. c is an explicit exception to the default rule "if b then normally a"). In general, logic programs with negation-as-failure may no longer have a unique least Herbrand model. This effectively makes these logic programs non-deterministic, and thus unsuitable as a basis for LRNNs in general.

A logic program with negation-as-failure $\mathcal{P}$ is called stratified if there exists a partition $\mathcal{P} = \mathcal{P}_1 \cup ... \cup \mathcal{P}_n$ such that for each rule $a \leftarrow b_1 \wedge ... \wedge b_k \wedge \textbf{not } b_{k+1} \wedge ... \wedge \textbf{not } b_{k+l}$ in $\mathcal{P}_i$ it holds that the predicates used in the atoms $b_{k+1}, ..., b_{k+l}$ do not occur in the head of any of the rules in $\mathcal{P}_i \cup ... \cup \mathcal{P}_n$ (Rondogiannis & Wadge, 2005). It is easy to verify that stratified logic programs have a unique least Herbrand model. It turns out that we can easily generalize LRNNs to cases where the corresponding logic program $\mathcal{N}^*$ is a stratified logic program with negation-as-failure. To this end, we define the grounding of such an LRNN $\mathcal{N}$ as

$$\overline{\mathcal{N}} = \{(h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta \wedge \textbf{not } b_{k+1}\theta \wedge \ldots, \wedge\textbf{not } b_{k+l}\theta, w) : \{h\theta, b_1\theta, \ldots, b_k\theta\} \subseteq \mathcal{H} \text{ and }$$
$$(h \leftarrow b_1 \wedge \cdots \wedge b_k \wedge \textbf{not } b_{k+1} \wedge \textbf{not } b_{k+l}, w) \in \mathcal{N}\}$$

where $\mathcal{H}$ is the least Herbrand model of $\mathcal{N}^*$. Note that we do not put any constraints on the containment of the atoms $b_{k+1}\theta \wedge \cdots \wedge b_{k+l}\theta$ in the Herbrand model $\mathcal{H}$. In a classical logic programming setting, we can omit the rule $h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta \wedge \textbf{not } b_{k+1} \wedge \textbf{not } b_{k+l}$ if one of the atoms $b_{k+1}, ..., b_{k+l}$ is in $\mathcal{H}$. However, in accordance with most multi-valued extensions of negation-as-failure, in LRNNs the atom neuron corresponding to **not** b_i may have a non-zero output even if b_i has a non-zero output. Intuitively, if we can only derive that b_i is partially true, a rule with **not** b_i in the body will still partially fire.

To construct the ground network of an LRNN with negation-as-failure, we consider an additional type of neuron called *negation neuron.* For every ground atom **not** a that occurs in one of the rules of the grounding $\overline{\mathcal{N}}$, we add one such negation neuron $\textbf{Not}_a$, which has the atom neuron for a as input, if it exists (i.e. if $a \in \mathcal{H}$), and the constant 0 otherwise. This negation neuron is then added as one of the inputs of the corresponding rule neuron. As with the other types of neurons, the activation function $g_{\mathbf{not}}$ associated with a negation neuron can be chosen in different ways.

Example 7 *Let us consider an LRNN $\mathcal{N}$ consisting of the rule*

$$w_1 : \mathit{flies}(X) \leftarrow \mathit{bird}(X) \wedge \mathbf{not}\ \mathit{antarctic}(X)$$

and the fact bird(tweety). The grounding $\overline{\mathcal{N}}$ contains the rule

$$\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{bird}(\mathit{tweety}) \wedge \mathbf{not}\ \mathit{antarctic}(\mathit{tweety}).$$

Therefore there will be a neuron $\mathbf{Not}_{\mathit{antarctic}(\mathit{tweety})}$ in the ground network. The input to this neuron is the constant 0 in this case because antarctic(tweety) is not in the least Herbrand model of the corresponding logic program $\mathcal{N}^*$. Assuming that the activation function of negation neurons is $1-x$, the output of this neuron will be 1. The rule neuron $R_{\mathit{flies}(\mathit{tweety})\leftarrow \mathit{bird}(\mathit{tweety})\wedge \mathbf{not}\ \mathit{antarctic}(\mathit{tweety})}$ then has two inputs, the neuron $\mathbf{Not}_{\mathit{antarctic}(\mathit{tweety})}$ and the atom neuron $A_{\mathit{bird}(\mathit{tweety})}$. The rest of the network is constructed as for normal LRNNs, following the rules described in Section 3.1.

In LRNNs where all the outputs are between 0 and 1, a natural choice for this activation function, which is in line with the semantics of several existing multi-valued logic programming frameworks (Damásio & Pereira, 2001a; Blondeel, Schockaert, Vermeir, & De Cock, 2013), is $g_{\mathbf{not}}(x) = 1 - x$. In other contexts, where positive and negative values are associated with positive and negative support, respectively, the choice $g_{\mathbf{not}}(x) = -x$ could be used.

3.4 Weight Learning

We consider an LRNN $\mathcal{N}$ whose weights we want to train. To this end, we assume that we also have access to a list of training examples $\mathcal{E} = (\mathcal{E}^1, \dots, \mathcal{E}^m)$, where each $\mathcal{E}^j$ is an LRNN. In applications, these LRNNs encoding training examples would typically only contain ground facts. In such cases, each training example intuitively describes some relational structure using a set of weighted atoms (e.g. as displayed in Fig. 2). We also assume that we are given a list $\mathcal{Q} = (\{(q_1^1, t_1^1), \dots, (q_{k_1}^1, t_{k_1}^1)\}, \dots, \{(q_1^m, t_1^m), \dots, (q_{k_m}^m, t_{k_m}^m)\})$ where each q_i^j is a ground atom, which we call a *training query atom*, and t_i^j is its *target value*. For a query atom q_i^j, let y_i^j denote the output of the atom neuron $A_{q_i^j}$ in the ground neural network of $\overline{\mathcal{N} \cup \mathcal{E}^j}$. The goal of the learning process is to find the weights w_h of the rules (and possibly facts) in $\mathcal{N}$ for which the loss J on the training query atoms $J(\mathcal{Q}) = \sum_{j=1}^{m} \sum_{i=1}^{k_j} \mathit{cost}(y_i^j, t_i^j)$ is minimized, where *cost* is some predefined loss function that measures the discrepancy between the output of the neurons corresponding to the training query atoms and their desired target values.

Example 8 *Let us again consider the LRNN $\mathcal{N}$ from Example 3 and the sets of facts $\mathcal{M}_1$ and $\mathcal{M}_2$ describing the two molecules (shown in Figure 2). We recall that $\mathcal{N}$ contains* *the rule for toxicity of molecules w_1 : toxic $\leftarrow gr_1(A) \wedge b(A,B) \wedge gr_2(B)$ where $b(.,.)$ is a predicate for representing atomic bonds in the molecules, and the predicates $gr_1(.)$ and $gr_2(.)$ are defined using the rules listed in Example 3. Let us suppose that we want to learn the weights of this theory based on the knowledge that $\mathcal{M}_1$ is toxic and $\mathcal{M}_2$ is not. In other words, the list of training examples in this case is given by $\mathcal{E} = (\mathcal{M}_1, \mathcal{M}_2)$. We also need to specify the corresponding list of training query atoms $\mathcal{Q}$, which in this case is given by $\mathcal{Q} = \{\{toxic, 1\}, \{toxic, 0\}\}$. The weight learning task is to optimize the weights so as to minimize the discrepancy between the toxicity according to the training query atoms and the toxicity predicted by the LRNNs $\mathcal{N} \cup \mathcal{M}_1$ and $\mathcal{N} \cup \mathcal{M}_2$, where the predicted toxicity is the output of the atom neuron A_{toxic}.*

Similarly to conventional neural networks, weight adaptation is performed by gradient descent steps:

$$w_h \leftarrow w_h - \gamma \frac{\partial J(\mathcal{Q})}{\partial w_h}$$

where $\gamma > 0$ is some given learning rate. Different from conventional neural networks, in the case of LRNNs, we end up with different ground networks for the different learning examples $\mathcal{E}^j$. This is not problematic, however, because the weights for all the ground neural networks $\overline{\mathcal{N} \cup \mathcal{E}^j}$ are fully specified in the LRNN $\mathcal{N}$.

It is possible to learn the weights of an LRNN using conventional stochastic gradient descent, based on gradients computed by backpropagation, except that the increments for the shared weights must be accumulated. This can be seen from the following reasoning. Let $J(u_1, u_2, \ldots, u_n)$ be a loss function of a given ground neural network. To encode which of these n weights are shared, we can consider a function $g : R^m \rightarrow R^n$, where $R = \{w_1, ..., w_m\}$ is the set of distinct weight variables ($m \leq n$). For a given vector $v \in R^m$, the vector $g(v)$ is obtained by copying the values of the shared weights. For instance, if $n = 3$, $R = \{w_1, w_2\}$ and the first two weights are shared, then $g(w_1, w_2) = (w_1, w_1, w_2)$. It follows from elementary multivariate calculus that $\frac{\partial}{\partial w_i}(J \circ g) = \nabla J \cdot \frac{\partial g}{\partial w_i}$, where $\cdot$ denotes scalar product (noting that both ∇J and $\frac{\partial g}{\partial w_i}$ are n-dimensional vectors). Since $\frac{\partial g}{\partial w_i}$ is a vector which has 1 at position j iff the j-th weight is assigned to the shared weight w_i, it follows that $\frac{\partial}{\partial w_i}(J \circ g)$ can be computed as the sum of the components of ∇J that correspond to the positions j to which w_i is assigned. In other words, this means that in order to compute the partial derivative w.r.t. a shared weight w_i, we can compute the gradient using standard backpropagation without assuming any weight sharing and sum the individual terms corresponding to the given shared weight.

The complete weight learning algorithm then works as follows. First, the given LRNN $\mathcal{N}$ is grounded w.r.t. every example $\mathcal{E}^j$, leading to a set of ground neural networks with shared weights; information about the origin of each weight is explicitly stored, such that the respective weights in the template can be updated correctly. The algorithm then iterates over the ground networks in a random order. Each time, it computes the gradient of the loss function for the current example given the current weights in the template, and updates the weights accordingly. It continues iterating these steps, following the standard stochastic gradient descent procedure. To reduce the risk of getting stuck in local optima, we can also employ a restart strategy for this algorithm, restarting the search with randomly initialized weights after a given number of SGD epochs has been reached. There are many restart sequences that we can use, e.g. Luby's universal strategy (Luby, Sinclair, & Zuckerman, 1993).

For the Max-Sigmoid family of activation functions, learning is complicated by the fact that the max operator introduces non-differentiable points to the optimization problem. As a consequence, some weights, corresponding to ground rules which never contribute to the output value (because they never give maximal output), may never be updated by SGD because the respective partial derivatives of the loss function are zero. This may then lead to poor solutions. The restart strategies mentioned above help to partially alleviate this problem.

A note on structure learning For many types of lifted models, the learning process is separated in two steps, called structure learning and weight learning. In the case of MLNs, for instance, the aim of structure learning consists in determining relevant first-order formulas, whose associated weights are then determined in the subsequent weight learning step. While LRNNs could, in principle, be used in a similar way, one of the main strengths of LRNNs is the fact that they can learn predictive latent relational structures in an efficient way. When we use LRNNs for this purpose, as we do in this paper, the first-order rules are completely generic. Their purpose is then to encode what types of latent structures we want to find, rather than to encode domain knowledge. All domain knowledge is then obtained through weight learning. This way of using LRNNs is illustrated in detail in the next section.

4. Illustrative Examples of LRNN Modeling Constructs

In this section we describe several modelling constructs that can straightforwardly be encoded using LRNNs, but which would be difficult or impossible to implement in frameworks such as CILP++ (Franca et al., 2014), which also combines logic and neural networks. The considered modelling constructs correspond to different kinds of latent relational structures that can be effectively learned using LRNNs. Frameworks such as CILP++ do not allow simultaneous learning of target and auxiliary predicates, and are thus not well-suited for learning latent structures. Instead, they rely on propositionalization (Krogel, Rawles, Železný, Flach, Lavrač, & Wrobel, 2003) and are thus only capable of learning latent features over the corresponding propositionalized representations. While somewhat similar modelling constructs can, in principle, be used in MLNs and in probabilistic logic programming systems such as Problog (De Raedt, Kimmig, & Toivonen, 2007), they would require EM algorithms which repeatedly need to perform computationally expensive probabilistic inference.

4.1 Implicit Soft Clustering

In many learning tasks, it has been observed that good results can be obtained by generating (soft) clusters of objects. The idea is then to make predictions based on the membership degrees of a given object in these clusters, rather than trying to make predictions directly from the features describing the object itself. As an example, let us consider the problem of predicting adverse effects of drugs. For this problem, the use of auxiliary clusters of similar drugs has been found to lead to significant improvements in predictive accuracy (Davis, Costa, Berg, Page, Peissig, & Caldwell, 2012). However, existing methods to generate these auxiliary clusters rely on a form of greedy discrete clustering, which can often be too crisp. Using LRNNs, on the other hand, we can simply define predicates that represent these *soft* clusters, and then automatically train the corresponding weights, which represent the membership degrees. This is illustrated in the following example.

Example 9 *Let us suppose that, similarly to the work of Davis et al. (2012), we have temporal data about patients, the drugs they took, and the time instants when changes in their health occurred. We want to predict adverse effects of drugs or their combinations. Let us also assume that we have a set of general rules like:*

$$\begin{aligned}
w_1^{(1)} : \mathit{effect}(P, AE, T2) &\leftarrow && \mathit{took}(P, D1, T1) \wedge \mathit{period}(T1, T2, T) \wedge \mathit{shortPeriod}(T) \wedge \\
& && \wedge \mathit{took}(P, D2, T2) \wedge \mathit{drugGroup1}(D1) \wedge \mathit{drugGroup2}(D2) \wedge \\
& && \wedge \mathit{effectGroup1}(AE) \\
& && \dots \\
w_1^{(2)} : \mathit{effectGroup1}(E) &\leftarrow && \mathit{headache}(E) \\
w_2^{(2)} : \mathit{effectGroup1}(E) &\leftarrow && \mathit{sneezing}(E) \\
& \dots
\end{aligned}$$

Here, effect(Patient, AdverseEffect, Time) is a predicate whose meaning is that the patient Patient had the adverse effect AdverseEffect at time Time. Similarly, took(Patient, Drug, Time) states that the patient Patient took the drug Drug at time Time, Period$(T1, T2, T)$ *is true when* $T = T2 - T1$, *i.e. it expresses that* T *is the length of the time period from* $T1$ *to* $T2$. *For simplicity, we assume that time is discretized (e.g. it may be enough to measure the time in days for certain types of adverse effects), which means that we may represent these predicates extensively as facts. The predicate shortPeriod is a predicate which defines what "short period" means in the given context; here we assume that it is specified by an expert*[2]. *Finally, the predicates drugGroup1, . . . , drugGroupN and effectGroup1, . . . , effectGroupM are latent predicates which will be learnt by the LRNN. For convenience, we use the superscript to index the latent predicates and subscript to index the rules defining the latent predicates, starting from* 1 *for each of the predicates.*

Intuitively, the adverse effects that happen to have high soft membership *in the same clusters of effects, defined by the effect-group predicates, are effects that tend to frequently occur together. The groups of drugs are supposed to represent similar drugs. The intuition behind the rules for the predicate effect/*3 *is as follows. Using these rules, we want to be able to capture the adverse effects which result from interactions of certain drugs that were taken by the patient in a short time interval. Let us assume that we already have definitions of the effectGroup/*1 *and drugGroup/*1 *in accordance with the described intuition. Then each of the rules essentially says that if a person took a drug from group 1 and shortly after a drug from group 2 then the patient will get (all the) adverse effects from the adverse-effect group 1. Note that the weights of these rules can also be negative, encoding the fact that the drugs that were taken actually prevent the adverse effects from the corresponding group.*

Using the Avg-Sigmoid family of activation functions, weight learning in this LRNN can implicitly discover clusters of drugs which interact adversely with other clusters of drugs,

[2] The predicate *shortPeriod* could be learned using techniques analogical to those described later for the predicate *Similar*(X, Y) in Section 6.2.

clusters of adverse effects corresponding to these combinations of drugs, and an appropriate definition for the predicate shortPeriod. To obtain these weights, we only require examples consisting of ground facts describing patients, the drugs they have taken, when they have taken these drugs, and what adverse effects they experienced.

In Section 6 we will provide experimental results showing that LRNNs can indeed learn useful soft clusters, using a scenario which is similar to the one from the previous example, in the domain of organic chemistry. Note that we could not perform experiments for the problem setting from example 9, as the required data are not publicly available for privacy reasons.

In the above example, soft clustering was essentially used to group related predicates. However, the underlying idea can also be applied to more complex relational structures. To illustrate this, the following example shows how we can group related (hyper)graphs in a similar way.

Example 10 *Let us consider the problem of predicting properties of organic molecules (e.g. toxicity) that depends on the presence of substructures from some rather large set. In this example, we will consider such substructures based on aromatic six-rings. The basic aromatic six-ring is the benzene ring, which is a ring of six carbon atoms, each connected to a hydrogen atom, connected by aromatic bonds. If some carbon atom is replaced by another atom, we speak of a* substitution.

If the patterns capturing classes of substructures in the molecules have the same structure, e.g. they are all aromatic six-rings with substitutions at some positions, one could in principle use probabilistic modeling to approximate them. The main idea would then be to estimate a probability distribution on the sets of substitutions, such that sets of substitutions which are jointly occurring in the individual patterns would have high probability. While such a probabilistic modeling approach is possible in principle, it would typically require us to introduce latent concepts, in which case we would have to resort to EM. Using LRNNs, on the other hand, learning latent representation patterns is straightforward. For instance, if we want to capture pairwise dependencies between the substitutions in neighboring atoms, we can first define auxiliary binary predicates of the following form:

$$\begin{aligned}
&w_1^{(1)} : s_1(carbon, nitrogen)\\
&w_2^{(1)} : s_1(carbon, oxygen)\\
&\dots
\end{aligned}$$

Each s_i is supposed to represent a group of substitution pairs which often appear together in discriminative substructures (because the weights will be learned in this way). Then, we can define a predicate sixRing as follows:

$$w_1^{(2)} : sixRing(A, B, C, D, E, F) \leftarrow ring(A, B, C, D, E, F) \wedge s_1(A, B) \wedge s_2(B, C) \wedge \dots s_6(F, A)$$

together with the following rule:

$$w_1 : toxic(M) \leftarrow atom(M, A) \wedge atom(M, B) \wedge \cdots \wedge atom(M, F) \wedge sixRing(A, B, C, D, E, F)$$

Intuitively, sixRing then represents a class of substructures whose presence in a molecule suggests that it is toxic. We can similarly define predicates for five-rings and other structures. For each of the these classes of substructures, we then add a rule, whose weight encodes how predictive that substructure is of toxicity, e.g.:

$$w_2 : \mathit{toxic}(M) \leftarrow \mathit{atom}(M, A) \wedge \mathit{atom}(M, B) \wedge \cdots \wedge \mathit{atom}(M, E) \wedge \mathit{fiveRing}(A, B, C, D, E) \ldots$$

When learning the weights of this LRNN (based on examples of toxic and non-toxic molecules), we then simultaneously discover the appropriate weights of the auxiliary predicates (e.g. s_1, sixRing) as well as the weights of the rules that predict the target predicate toxic. In other words, we jointly learn what the latent substructures represent and how predictive they are.

Finally, as an illustration of a more elaborate use of LRNNs for learning soft clusters, we refer to the work of Šourek, Manandhar, Železný, Schockaert, and Kuželka (2016), where a method is proposed to simultaneously learn soft clusters of predicates and soft clusters of entities, based on a reified representation of predicates.

4.2 Approximate Matching

If the body of a given rule is only approximately satisfied, it often makes sense to still derive the head of that rule, but with a lower degree. Using LRNNs we can easily learn how the different ways in which the body can be approximately satisfied should affect the degree to which we want to derive the head. We refer to this modelling construct as approximate matching.

Example 11 *Let us again consider the example about predicting who has the flu. Let us consider the following rule, expressing that if X is in a group of 4 people who are mutual friends and all of them have flu symptoms, then X has the flu:*

$$\begin{aligned} w_1^{(1)} : \mathit{hasFlu}(X) \leftarrow \quad & \mathit{clique}(W, X, Y, Z) \wedge \mathit{fluSymptoms}(W) \wedge \mathit{fluSymptoms}(X) \\ & \wedge \mathit{fluSymptoms}(Y) \wedge \mathit{fluSymptoms}(Z). \end{aligned} \tag{2}$$

The requirement that the friendship graph of W, X, Y, Z is a clique seems unnecessarily strict. For instance, the rule is still meaningful if two of these four people are not actually friends, although in such a case we may prefer to derive the conclusion that X has the flu with lower certainty. In general, the more of the friendship relations are missing, the lower the certainty of the conclusion should intuitively be. This can easily be expressed using LRNNs, e.g. by defining the predicate clique as a soft concept and automatically learning the respective weights:

$$\begin{aligned} w_1^{(2)} : \mathit{clique}(W, X, Y, Z) \quad & \leftarrow \quad f(W, X) \wedge f(W, Y) \wedge f(W, Z) \wedge f(X, Y) \wedge f(X, Z) \wedge f(Y, Z) \\ w_1^{(3)} : \mathit{f}(X, Y) \quad & \leftarrow \quad \mathit{friends}(X, Y) \wedge \mathit{friends}(Y, X) \\ w_2^{(3)} : \mathit{f}(X, Y) \quad & \leftarrow \quad \mathit{friends}(X, Y) \\ w_3^{(3)} : \mathit{f}(X, Y). & \end{aligned}$$

where the predicate friends is specified in the description of the examples. Note how the predicate f enables a flexible definition of the predicate clique, and how this allows us to *draw conclusions in situations which only partially match the body of the rule* (2). *Using the activation functions from the Max-Sigmoid family for the predicates hasFlu and f, we can obtain the desired behavior.*

5. Related Work

The main inspiration for the work presented in this paper are lifted graphical models such as Markov logic networks (Richardson & Domingos, 2006), Bayesian logic programs (Kersting & De Raedt, 2001) or Relational dependency networks (Neville & Jensen, 2007). However, none of these existing lifted graphical models is particularly well suited for learning parameters of latent relational structures. Our approach is also generally related to prior art in combining logical rules with neural networks, also known as neural-symbolic integration (d'Avila Garcez, Broda, & Gabbay, 2012), such as in the KBANN system (Towell et al., 1990). While KBANN also constructs the network structure from given rules, these rules are propositional rather than relational and do not serve as a lifted template. Crucially, this means that KBANN cannot be used to learn latent relational structures. A more recent system, called CILP++(Franca et al., 2014), utilizes a relational representation, which is however converted into a propositional form through a propositionalization technique (Krogel et al., 2003). This again means that latent relational structures such as those exemplified in Section 4 cannot be learned by CILP++. The work on First Order Neural Networks (FONN) (Botta et al., 1997) is more closely related, in that it also involves a technique to construct neural networks from relational rule sets. However, in FONN only flat rule sets are considered, producing 1-layer (shallow) networks in which relational patterns cannot be hierarchically aggregated. While there are many other approaches to neural-symbolic integration that rely on relational (and first-order) representations (Bader & Hitzler, 2005), e.g. based on the CORE method (Hölldobler, Kalinke, & Störr, 1999), they typically search for a neural network modelling a given logic program, and thus principally differ from the presented lifted modeling approach.

While standard (and convolutional) feed-forward neural networks can be seen as a special case of LRNNs, a salient aspect of our method is that it allows for learning from structured (relational) examples, rather than just attribute vectors (or tensors). There has been previous work on adapting neural networks to cope with certain facets of relational representations. For example, an extension to multi-instance learning was presented by Ramon and De Raedt (2000). A similarly directed work (Blockeel & Uwents, 2004) facilitated aggregative reasoning, with the aim of processing sets of related tuples from a relational database as a sequence through a recurrent neural network structure. These approaches are fundamentally different from the presented method as they do not follow the lifted modeling strategy to cope with variations in the structure of relational samples. More loosely related works can also be found in the neural networks community, where various recursive autoencoders based on the idea of "reduced descriptions" (Hinton, 1990) are used to encode structured data. Another line of work are convolutional neural networks (LeCun, Bottou, Bengio, & Haffner, 1998) and techniques of indirect encoding (Clune, Stanley, Pennock, & Ofria, 2011), exploiting patterns and regularities in neural connections to create more compressed representations of large neural networks. However, these approaches are still geared towards learning from fixed-size propositional, rather than relational, data. Demeester, Rocktäschel, and Riedel (2016) developed a method that allows injecting knowledge in the form implication rules into distributed representations for the task of knowledge base construction, although their method is restricted to binary predicates and they only consider rules with a single atom in the body.

Recently[3] two other frameworks for combining logic programming and neural networks have been introduced (Rocktäschel & Riedel, 2016; Cohen, 2016). Rocktäschel and Riedel (2016) introduced differentiable theorem provers which explicitly represent constants in a distributed manner as vectors. Unification in their approach is soft and returns a value which is determined based on dot-products fed into sigmoids. When different unifying substitutions to the same variables need to be aggregated, their approach takes their maximum value. It is interesting to note that a similar deductive reasoning process can be implemented using LRNNs, with the activation functions $g_\vee$ and g_* chosen as the *maximum.* In particular, soft unification can be modelled by pre-computing the dot products between the vector representations of all (relevant) pairs of constants, and by encoding these dot products as the weights of facts of the form $match(c_i, c_j)$. Hence, it should be possible to represent any learned model from their framework in LRNNs. However, other modelling constructs are actually more natural for working with distributed representation in LRNNs, for instance soft clustering. Cohen (2016) introduced a system called TensorLog, which is a differentiable probabilistic database based on belief propagation. TensorLog implements a subset of Datalog. It restricts the factor graphs constructed for the belief propagation step to be tree-like. Further restrictions include the fact that only unary and binary predicates are allowed, and only certain types of queries are supported. Because of these restrictions, it would be difficult to directly compare TensorLog with LRNNs. Both approaches seem to be tailored towards different types of tasks. One advantage of TensorLog is that it does not require a complete grounding of the set of rules to perform inference. While we have relied on complete groundings in this paper, even for LRNNs it would often be sufficient to limit grounding to the proofs of the given query formula. Crucially, however, this requires a fast top-down inference engine. In preliminary experiments, we have found such top-down grounding of LRNNs to be significantly slower than the current bottom-up grounding. Another recent work (Niepert, 2016) introduced so called *discriminative Gaifman models* which are models that aggregate information from locally sampled neighbourhoods, motivated by Gaifman's locality theorem (Gaifman, 1982).

A number of efficient methods have recently been proposed for structured output prediction, such as the work on so-called input convex neural networks (Amos, Xu, & Kolter, 2017). In the future it would be interesting to try to combine this work with LRNNs for types of abductive reasoning, for which relational linear programming (Kersting, Mladenov, & Tokmakov, 2017) could turn out to be particularly suitable.

6. Experiments

In this section, we describe experiments performed on 78 datasets about organic molecules: the Mutagenesis dataset (Lodhi & Muggleton, 2005), four datasets from the predictive toxicology challenge, and 73 NCI datasets (Ralaivola, Swamidass, Saigo, & Baldi, 2005). The

[3] Note that these two approaches were published after the workshop and arxiv papers that first described LRNNs (Šourek et al., 2015a; Šourek, Aschenbrenner, Železný, & Kuželka, 2015b).

Mutagenesis dataset contains information about 188 molecules, with labels denoting their mutagenicity. A number of published results for this dataset have relied on an extended set of features, providing additional expert knowledge about relational properties of molecules. Since we want to focus on the learning capabilities of our model, we will not rely on these additional features, and only use atom bond information. The predictive toxicology challenge dataset (PTC) (Helma, King, Kramer, & Srinivasan, 2001) is composed of four datasets about molecules, labeled by their toxicity for female rats (fr) and mice (fm) and male rats (mr) and mice (mm). Each of the NCI-GI datasets contains several thousands of molecules, labeled by their ability to inhibit growth of different types of tumors. Detailed statistics of these datasets are in Table 2.

We compare the performance of LRNNs with the state-of-the-art relational learners kFOIL (Landwehr, Passerini, De Raedt, & Frasconi, 2006) and nFOIL (Landwehr, Kersting, & Raedt, 2007), which respectively combine relational rule learning with support vector machines and with naive Bayes learning. We also compare LRNNs with MLN-boost (Khot, Natarajan, Kersting, & Shavlik, 2011) and RDN-boost (Natarajan, Khot, Kersting, Gutmann, & Shavlik, 2012), which are both based on functional gradient boosting (Friedman, 2001) together with Markov logic networks and relational dependency networks (Neville & Jensen, 2007), respectively. We also attempted a comparison with CILP++ (Franca et al., 2014), but were not able to transform the datasets into the propositional representation which is used by CILP++ using the publicly available part of the CILP++ implementation. In addition we performed experiments with Aleph (Srinivasan, 2000), which we used both in its abductive and inductive modes. In the abductive mode we gave Aleph the same graphlet defining rules as we give to LRNNs in the experiments with the soft clustering modelling construct that we report in Section 6.1. In theory, Aleph could learn definitions of crisp clusters by abduction, although it cannot learn soft clusters. In practice we found that it was not effective. Interestingly, the inductive mode of Aleph did not achieve competitive results on the NCI datasets either; it rarely exceeded majority-class accuracy.

To demonstrate the versatility of LRNNs, we perform experiments with different templates, representing some of the modeling constructs that were discussed in Section 4. Each time, we only make use of generic templates, ensuring that the rules that are provided are not predictive by themselves, and that the weight learning must thus create useful latent relational concepts in order to be succesful. In particular, the considered templates do not relate to any specific property of molecules and might be equally useful for other classification tasks. The idea is that useful latent relational concepts emerge from the gradient descent based weight learning process, rather than by explicit enumeration, in contrast to propositional approaches and ILP (De Raedt, 2008). Nonetheless, in real applications the fact that declaratively specified expert knowledge can be provided is of course an important strength of LRNNs. Table 3 lists statistics of the ground neural networks for LRNNs based on the different modelling constructs.

For all the reported experiments, we set the learning rate to 0.3 and training epochs to 3000. In general, we found that training was not very sensitive to the learning rate (with the effective range being up to 0.5) as long as a sufficient number of learning steps is used. We set the learning rate relatively high so as to keep the number of necessary epochs to converge reasonable. The time for training an LRNN on a standard commodity machine

Table 2: Statistics of the three groups of molecular datasets used in the experiments. Except for the number of datasets, the remaining numbers are averages over the datasets in the given group.

	#datasets	avg. #examples	avg. #bonds	avg. #atoms
NCI	73	3031	50	23
PTC	4	342	51	25
MUTA	1	188	56	26

Table 3: Average sizes of ground neural networks (average number of atom neurons per network) corresponding to the LRNNs based on the different modelling constructs.

	Soft Clusters	App. Matching	Atom Embeddings	Charge	Charge+Soft
NCI	520	1145	1241	1373	1396
PTC	598	1619	1292	1414	1435
MUTA	696	1800	1338	1445	1482

with one CPU was in the order of a few hours for the larger NCI-GI datasets, and in the order of a few minutes for the smaller datasets such as Mutagenesis.

6.1 Soft Clustering

We start with a simple hand-crafted LRNN template which is based on the idea of implicit soft clustering that was described in Section 4.1. The template defines 3 predicates for soft clusters of atom types and 2 predicates for soft clusters of bond types. The predicates *atgr1*, *atgr2*, and *atgr3*, representing soft clusters of atom types are defined by considering one rule for every atom type occurring in the dataset, e.g.:

$$\begin{aligned} w_1^{(1)} &: \mathit{atgr1}(X) \leftarrow o(X) \\ w_2^{(1)} &: \mathit{atgr1}(X) \leftarrow br(X) \\ &\dots \end{aligned}$$

Similarly, the predicates *bondgr1* and *bondgr2* representing soft clusters of bond types are defined by considering one rule for every bond type occurring in the dataset. These predicates are then used to define rules for different types of atom chains of length 3, one for each group choice for each of the 3 atoms' soft clusters and each of the 2 bonds' soft clusters, i.e. 243 rules in total:

$$\begin{aligned} w_{(1,1,1;1,1)}^{(2)} : \mathit{chain3} \leftarrow & \mathit{atgr1}(X) \wedge \mathit{bond}(X,Y,B1) \wedge \mathit{atgr1}(Y) \wedge \mathit{bond}(Y,Z,B2) \\ & \wedge \mathit{atgr1}(Z) \wedge \mathit{bondgr1}(B1) \wedge \mathit{bondgr1}(B2), \\ & \dots \\ w_{(3,3,3;2,2)}^{(2)} : \mathit{chain3} \leftarrow & \mathit{atgr3}(X) \wedge \mathit{bond}(X,Y,B1) \wedge \mathit{atgr3}(Y) \wedge \mathit{bond}(Y,Z,B2) \\ & \wedge \mathit{atgr3}(Z) \wedge \mathit{bondgr2}(B1) \wedge \mathit{bondgr2}(B2). \end{aligned}$$

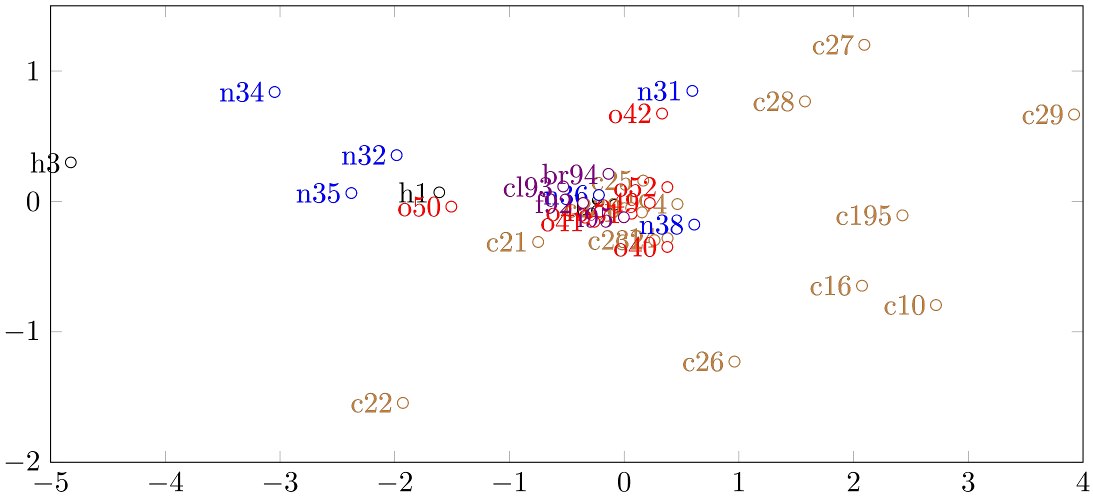

Figure 4: PCA projection of vector embeddings of atom types corresponding to the learned weights of soft clusters in the Mutagenesis dataset. Brown denotes the group 14 of the periodic table (carbon group), blue the group 15 (pnicotgens), red the group 16 (chalcogens), violet the group 17 (halogens) and black the group 1 (hydrogen).

The predicate *chain3* then represents the, possibly varying, learning target for each of the molecular datasets (e.g. toxicity or mutagenicity). A comparison between the results obtained with this LRNN template and those obtained with kFoil, nFoil, MLN-boost and RDN-boost is shown in Figure 5. As can clearly be seen from this figure, the LRNN method consistently outperforms the four baselines.

The learned weights of the rules defining the predicates *atgr1*, *atgr2* and *atgr3* can be interpreted as membership degrees of the atom types to the three soft clusters. These degrees might be interpreted as defining a three-dimensional vector space embedding of the atom types. The first two principal components of these embeddings, for different atom types from the Mutagenesis dataset, are shown in Figure 4. Note that the atom types in the Mutagenesis dataset have been enriched with contextual information, which is why there are different atom types *c21, c22, . . . , c195* which all refer to carbon atoms. The LRNNs are not given any explicit information about how these different atom types are related, and thus have to reconstruct this information from the available training data. It is therefore interesting to see that in the embedding from Figure 4, the nitrogen atoms are mostly in the top left corner, carbons are mostly in the bottom right corner and the rest of the atoms are around the center of the plot (where some further noticeable patterns can be observed, such as halogen atoms being clustered together).

Next we demonstrate the importance of relational information in the molecule classification tasks. Specifically, we show that a model which captures conformations of particular atom types leads to better classification accuracy than a model which is only based on a soft clustering of atom types. To this end, we created 3 templates with increasing complexity. The first template is based purely on soft clustering, i.e. it only considers relational chains of size 0. For each $i \in \{1, 2, 3\}$ we consider the following rules, one for each soft cluster of

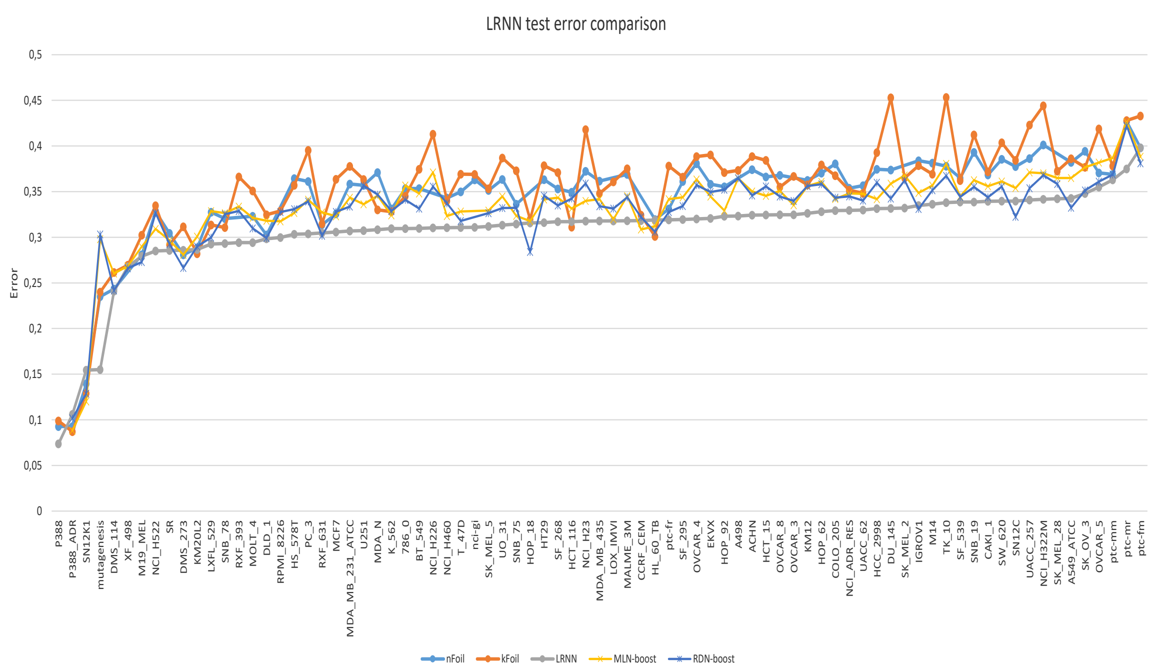

Figure 5: Prediction errors of LRNNs, kFOIL, nFOIL, MLN-boost and RDN-boost measured by cross-validation on 78 datasets about organic molecules.

atom types:

$$w^{(2)}_{(i)} : \mathit{chain1} \leftarrow \mathit{atgr}\mathrm{i}(X)$$

The second template involves relational chains composed of two atoms. It contains the following rule for each $i, j, k \in \{1, 2, 3\}$:

$$w^{(2)}_{(i,j,k)} : \mathit{chain2} \leftarrow \mathit{atgr}\mathrm{i}(X) \wedge \mathit{bond}(X, Y, B1) \wedge \mathit{atgr}\mathrm{j}(Y) \wedge \mathit{bondgr}\mathrm{k}(B1)$$

Finally, we consider the template with the *chain3* predicate, describing chains of 3 atoms, which we used in the previous experiment.

Results for the three templates *chain1*, *chain2* and *chain3* are shown in Figure 6. While most of the performance is clearly due to the use of soft clustering, using non-trivial relational chains does lead to improved predictive accuracy. It is evident from the graph that relational chains of length greater than 1 are better than relational chains of length 1. The difference between chains of lengths 2 and 3 is smaller but still statistically significant ($p = 0.002$, using binomial test).

6.2 Alternative Modeling Constructs

Beyond learning soft clusters of atom types and bond types, a wide variety of other constructs can be used to solve the considered learning tasks. In this section, we briefly discuss a number of these alternatives.

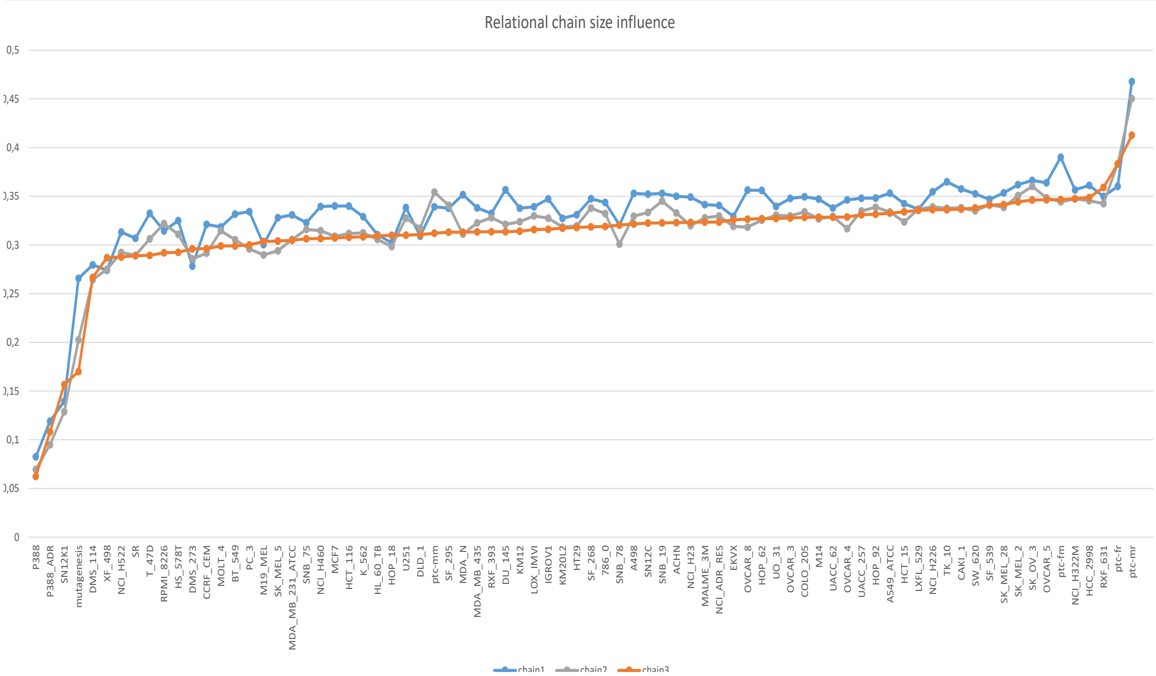

Figure 6: Test errors of three LRNN templates with growing relational chain sizes as measured by cross-validation on 78 datasets about organic molecules.

6.2.1 Learnable Numerical Transformations

In the previous section, we showed how the soft clusters that are learned by an LRNN can be interpreted as vector space embeddings. Conversely, we can also generate soft clusters from a given pre-trained embedding. To demonstrate this idea, we will use a simple vector space representation, which encodes for each atom type how its valence electrons are distributed across the s,p,d,f orbitals. For instance, the oxygen atom type O with electron configuration $(1s2)[2s2, 2p4]$ is encoded as the vector $O := [2, 4, 0, 0]$. To construct an LRNN that takes advantage of this external knowledge, we actually use the similarity degrees induced by these vectors, rather than the vectors themselves. For this experiment, we measure the similarity between two atom types as the cosine between their vector representations. We then construct the LRNN template as follows.

For each pair of atom-types (a_1, a_2) with similarity degree s, we add the following ground fact:

$$1.0 : \mathit{Similar}(a_1, a_2, s)$$

We also add rules which encode a *learnable transformation* of the similarities into a score that is useful for the considered predictive task:

$$\begin{aligned}
&w_{-1}: && \mathit{Similar}(X,Y) \leftarrow \mathit{Similar}(X,Y,S), S \geq -1.0\\
&w_{-0.9}: && \mathit{Similar}(X,Y) \leftarrow \mathit{Similar}(X,Y,S), S \geq -0.9\\
&... && ...\\
&w_{0.9}: && \mathit{Similar}(X,Y) \leftarrow \mathit{Similar}(X,Y,S), S \geq \quad 0.9
\end{aligned}$$

We then randomly sample three atom type vectors as $prototype1 := [2,0,0,14]$, $prototype2 := [1,0,10,0]$, $prototype3 := [2,6,10,0]$ and modify the definition of atom groups to reflect the similarity to one of these prototypes:

$$\begin{aligned}
&w_1^{(1)} : \mathit{atgr1}(X) \leftarrow \mathit{Similar}(X, prototype1)\\
&\dots\\
&w_3^{(1)} : \mathit{atgr3}(X) \leftarrow \mathit{Similar}(X, prototype3)
\end{aligned}$$

We will refer to this method as *atomEmbeddings*.

As an alternative, we also tested how well the atom groups can be induced from the charge of each atom within a given molecule. As this information is only available in the NCI datasets, for this variant we do not consider the predictive toxicology datasets. To generate the atom groups, we again use a learnable transformation, but this time based on partial atom charges. Noting that the partial atom charges in the datasets are always between -1 and 1, this can be done as follows:

$$\begin{aligned}
&w_{-1}: && \mathit{atgr1}(X) \leftarrow \mathit{Charge}(X,C), C \geq -1.0\\
&w_{-0.9}: && \mathit{atgr1}(X) \leftarrow \mathit{Charge}(X,C), C \geq -0.9\\
&... && ...\\
&w_{0.9}: && \mathit{atgr1}(X) \leftarrow \mathit{Charge}(X,C), C \geq \quad 0.9
\end{aligned}$$

This method will be referred to as *atomCharge*. Finally we also tried to combine the soft cluster definition of atom groups with the definition based on atom charges, the results of which can be seen as *charge+softCluster* in Figure 7. To construct these combined LRNNs, we simply merge the definitions of the atom groups *atgr*i from both of the LRNNs.

The experimental results for the considered methods are depicted in Figure 7. The test errors of the LRNNs based purely on the partial charges are higher than the test errors of LRNNs based purely on soft clustering, which was to be expected. Indeed, similar results for relational features based on atom types or partial charges have been previously reported (Kuželka, Szabóová, & Železný, 2012). However, the fact that the combined LRNNs did not outperform the soft clustering LRNNs is more surprising. It suggests that the soft clusters built from the extended atom types present in the NCI datasets (e.g. *c21, c22, . . .*) may already capture the information present in the information about partial charges.

6.2.2 Approximate Matching

The aim of this experiment is to demonstrate the capability of LRNNs to capture structural similarities within the relational features. Following the idea of the approximate matching construct (see Section 4.2), we create a more flexible variant of the relation representing the bond between two atoms, such that more complex structural patterns can be matched, with

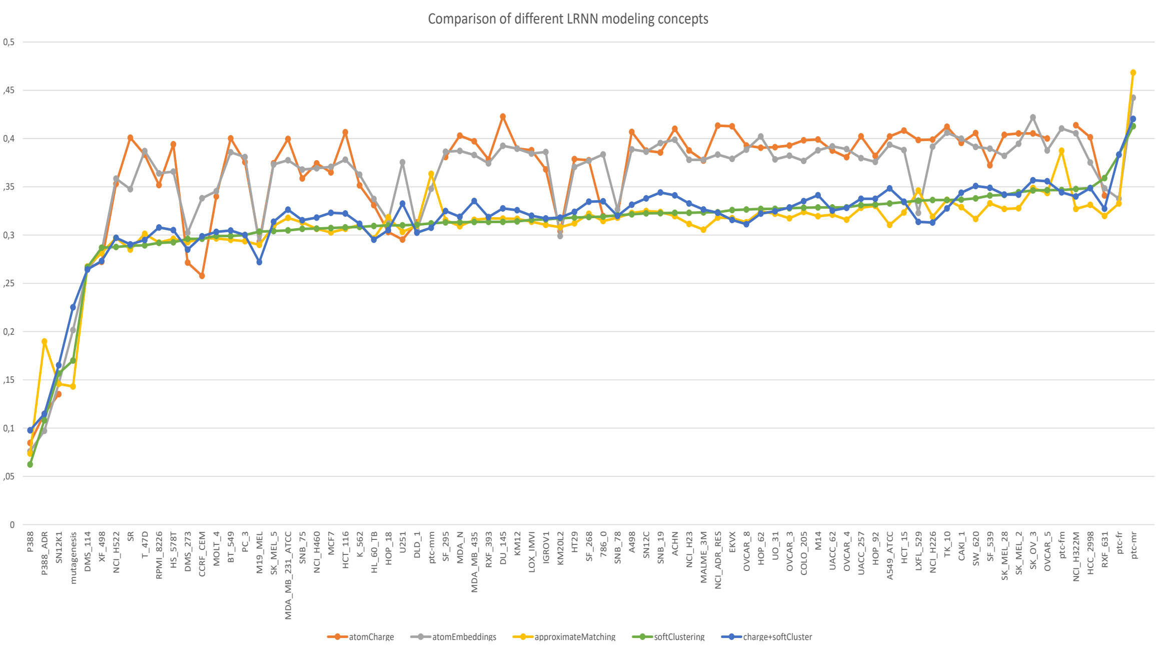

Figure 7: Prediction errors of the various introduced LRNN modeling concepts as measured by cross-validation on 78 datasets about organic molecules.

different degrees of similarity. We use the template with chains of 3 consecutive atoms from Section 6.1, but replace the predicate *bond* by a new predicate *bondK*. This new predicate is then defined in terms of the predicate *bond*, as follows:

$$w_1 : bondK(X, Y, B) \leftarrow bond(X, Y, B)$$
$$w_2 : bondK(X, Y, B) \leftarrow bond(X, Z, B), bond(Z, Y, B2)$$

The results of the experiments with this template are displayed as *approximateMatching* in Figure 7. Although the differences are small, the *approximateMatching* method obtained statistically significantly better accuracies than the LRNNs which only used relational chains of length 3 with soft clustering ($p = 0.008$, using binomial test).

7. Conclusions

In this paper, we have introduced LRNNs, a new framework for learning from relational data. Similar as in lifted probabilistic frameworks such as Markov logic and Problog, learned LRNN models are represented as sets of weighted first-order formulas. However, while Markov logic and Problog models serve as templates for constructing probabilistic graphical models, LRNN models serve as templates for constructing feedforward neural networks. This means that we can employ neural network learning, based on backpropagation, to efficiently discover latent relational structures from training data. Thanks to the use of first-order logic rules, we can easily specify what kind of latent structures we want the network to learn. In the experimental results, we have shown that very general rules, essentially indicating that we want to find predictive groups of atom types and predictive groups of bond types, allow us to achieve state-of-the-art predictive accuracies on various datasets about organic molecules. Furthermore, we have discussed and evaluated several other modelling constructs, e.g. based on learning latent groups of graph patterns, approximate matching of relational patterns, and using pre-trained vector space embeddings.

There are several interesting avenues for future work. First, in our experiments we have only considered generic templates, while one of the advantages of using logic-based representations is that we can easily incorporate domain knowledge into the learning process. Such domain knowledge could be explicitly provided by experts, or could be derived automatically using rule induction methods. These rules could be learned from the training data itself, but also possibly from related datasets about the same domain. Related to this latter point, we believe that LRNNs can allow for a natural way of modelling various forms transfer learning. Another possibility, which stays closer to the way in which we have been using LRNNs in this paper, is to learn rules from training data that aim to capture which types of latent relational structures are meaningful for the considered setting. Some initial work along these lines is presented in (Šourek, Svatoš, Železný, Schockaert, & Kuželka, 2017). At the technical level, it seems interesting to study a wider variety of activation functions, and to consider LRNNs that correspond to recurrent neural networks.

Acknowledgments

GŠ and FŽ acknowledge support by project no. 17-26999S granted by the Czech Science Foundation. SS is supported by ERC Starting Grant 637277. This work was done while OK was with Cardiff University and supported by a grant from the Leverhulme Trust (RPG-2014-164) and ERC Starting Grant 637277. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

References

Achs, Á., & Kiss, A. (1995). Fuzzy extension of datalog. *Acta Cybernetica*, *12*, 153–166.

Amos, B., Xu, L., & Kolter, J. Z. (2017). Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, pp. 146–155.

Bader, S., & Hitzler, P. (2005). Dimensions of Neural-symbolic Integration - A Structured Survey. *arXiv preprint*.

Blockeel, H., & Uwents, W. (2004). Using neural networks for relational learning. In *ICML-2004 Workshop on Statistical Relational Learning and its Connection to Other Fields*.

Blondeel, M., Schockaert, S., Vermeir, D., & De Cock, M. (2013). Fuzzy answer set programming: An introduction. In *Soft Computing: State of the Art Theory and Novel Applications*, Vol. 291 of *Studies in Fuzziness and Soft Computing*, pp. 209–222. Springer.

Botta, M., A, G., & Piola, R. (1997). Combining first order logic with connectionist learning. In *Proceedings of the 14th International Conference on Machine Learning*.

Clune, J., Stanley, K. O., Pennock, R. T., & Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Trans. Evolutionary Computation*, *15*(3), 346–367.

Cohen, W. W. (2016). Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

Damásio, C. V., & Pereira, L. M. (2001a). Antitonic logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 379–392.

Damásio, C. V., & Pereira, L. M. (2001b). Antitonic logic programs. In *Proceedings of the International Conference on Logic Programming and Non-Monotonic Reasoning*, pp. 379–393.

d'Avila Garcez, A. S., Broda, K., & Gabbay, D. M. (2012). *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer-Verlag London.

Davis, J., Costa, V. S., Berg, E., Page, D., Peissig, P. L., & Caldwell, M. (2012). Demand-driven clustering in relational domains for predicting adverse drug events. In *Proceedings of the 29th International Conference on Machine Learning, ICML*.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer.

De Raedt, L., Kersting, K., Natarajan, S., & Poole, D. (2016). *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). Problog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2462–2467.

Demeester, T., Rocktäschel, T., & Riedel, S. (2016). Lifted rule injection for relation embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016*, pp. 1389–1399.

Franca, M. V., Zaverucha, G., & Garcez, A. S. d. (2014). Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, *94*(1), 81–104.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

Gaifman, H. (1982). On local and non-local properties. In Stern, J. (Ed.), *Proceedings of the Herbrand Symposium*, Vol. 107 of *Studies in Logic and the Foundations of Mathematics*, pp. 105 – 135. Elsevier.

Helma, C., King, R. D., Kramer, S., & Srinivasan, A. (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics*, *17*(1), 107–108.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, *46*(1-2), 47–75.

Hölldobler, S., Kalinke, Y., & Störr, H. P. (1999). Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, *11*(1), 45–58.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with bayesian networks. In *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings*, pp. 118–131.

Kersting, K., Mladenov, M., & Tokmakov, P. (2017). Relational linear programming. *Artif. Intell.*, *244*, 188–216.

Khot, T., Natarajan, S., Kersting, K., & Shavlik, J. W. (2011). Learning markov logic networks via functional gradient boosting. In *11th IEEE International Conference on Data Mining, ICDM 2011*, pp. 320–329.

Kimmig, A., Mihalkova, L., & Getoor, L. (2015). Lifted graphical models: a survey. *Machine Learning*, *99*(1), 1–45.

Klement, E. P., Mesiar, R., & Pap, E. (1997). Triangular norms. *Tatra Mountains Mathematical Publications*, *13*, 169–193.

Kok, S., & Domingos, P. M. (2007). Statistical predicate invention. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pp. 433–440.

Krogel, M.-A., Rawles, S., Železný, F., Flach, P. A., Lavrač, N., & Wrobel, S. (2003). *Comparative evaluation of approaches to propositionalization*. Springer.

Kuželka, O., Szabóová, A., & Železný, F. (2012). Relational learning with polynomials. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012*, pp. 1145–1150.

Landwehr, N., Passerini, A., De Raedt, L., & Frasconi, P. (2006). kFOIL: learning simple relational kernels. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, pp. 389–394. AAAI Press.

Landwehr, N., Kersting, K., & Raedt, L. D. (2007). Integrating naive bayes and foil. *The Journal of Machine Learning Research*, *8*, 481–507.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323.

Lloyd, J. W. (2012). *Foundations of logic programming*. Springer Science & Business Media.

Lodhi, H., & Muggleton, S. (2005). Is mutagenesis still challenging. *ILP-Late-Breaking Papers*, *35*.

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of las vegas algorithms. *Information Processing Letters*, *47*(4), 173–180.

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., & Shavlik, J. W. (2012). Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, *86*(1), 25–56.

Neville, J., & Jensen, D. D. (2007). Relational dependency networks. *Journal of Machine Learning Research*, *8*, 653–692.

Niepert, M. (2016). Discriminative gaifman models. *CoRR*, *abs/1610.09369*.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*(1), 77–105.

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Netw.*, *18*(8), 1093–1110.

Ramon, J., & De Raedt, L. (2000). Multi instance neural networks. In *Proceedings of the ICML Workshop on Attribute-Value and Relational Learning.*

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine learning*, *62*(1-2), 107–136.

Rocktäschel, T., & Riedel, S. (2016). Learning knowledge base inference with neural theorem provers. *Proceedings of AKBC*, 45–50.

Rondogiannis, P., & Wadge, W. W. (2005). Minimum model semantics for logic programs with negation-as-failure. *ACM Transactions on Computational Logic (TOCL)*, *6*(2), 441–467.

Smullyan, R. M. (1995). *First-order logic.* Courier Corporation.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., Potts, C., et al. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Vol. 1631, p. 1642. Citeseer.

Srinivasan, A. (2000). The Aleph manual (Technical Report). Computing Laboratory, Oxford University.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the eighth National conference on Artificial intelligence*, pp. 861–866. Boston, MA.

Uwents, W., Monfardini, G., Blockeel, H., Gori, M., & Scarselli, F. (2011). Neural networks for relational learning: an experimental comparison. *Machine Learning*, *82*(3), 315–349.

Van Emden, M. H., & Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, *23*(4), 733–742.

Šourek, G., Aschenbrenner, V., Železný, F., & Kuželka, O. (2015a). Lifted relational neural networks. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015).*

Šourek, G., Aschenbrenner, V., Železný, F., & Kuželka, O. (2015b). Lifted relational neural networks. *arXiv preprint arXiv:1508.05128.*

Šourek, G., Manandhar, S., Železný, F., Schockaert, S., & Kuželka, O. (2016). Learning predictive categories using lifted relational neural networks. In *Inductive Logic Programming - 26th International Conference, ILP 2016, Revised Selected Papers*, pp. 108–119.

Šourek, G., Svatoš, M., Železný, F., Schockaert, S., & Kuželka, O. (2017). Stacked structure learning for lifted relational neural networks. In *Proceedings of the 27th International Conference on Inductive Logic Programming*, pp. 140–151.