Relational Knowledge Extraction from Neural Networks

Manoel Vitor Macedo França
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB
manoel.franca@city.ac.uk

Artur S. d'Avila Garcez
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB
a.garcez@city.ac.uk

Gerson Zaverucha
Prog. de Eng. de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil 21941–972
gerson@cos.ufrj.br

Abstract

The effective integration of learning and reasoning is a well-known and challenging area of research within artificial intelligence. Neural-symbolic systems seek to integrate learning and reasoning by combining neural networks and symbolic knowledge representation. In this paper, a novel methodology is proposed for the extraction of relational knowledge from neural networks which are trainable by the efficient application of the backpropagation learning algorithm. First-order logic rules are extracted from the neural networks, offering interpretable symbolic relational models on which logical reasoning can be performed. The well-known knowledge extraction algorithm TREPAN was adapted and incorporated into the first-order version of the neural-symbolic system CILP++. Empirical results obtained in comparison with a probabilistic model for relational learning, Markov Logic Networks, and a state-of-the-art Inductive Logic Programming system, Aleph, indicate that the proposed methodology achieves competitive accuracy results consistently in all datasets investigated, while either Markov Logic Networks or Aleph show considerably worse results in at least one dataset. It is expected that effective knowledge extraction from neural networks can contribute to the integration of heterogeneous knowledge representations.

1 Introduction

Integrating learning and reasoning efficiently and accurately has a vast track of research and publications in artificial intelligence [1, 2, 3, 4]. This integration can be done at different stages of learning, from data pre-processing, feature extraction, the learning algorithm, up to reasoning about learning. Neural-symbolic systems seek to integrate learning and reasoning by combining neural networks and symbolic knowledge representations using, e.g., propositional logic or first-order logic.

Relational learning can be described as the process of learning a first-order logic theory from examples and domain knowledge [5, 6]. Differently from propositional learning, relational learning does not use a set of attributes and values. Instead, it is based on objects and relations among objects, which are represented by constants and predicates, respectively. Relational learning has had applications in bioinformatics, graph mining and link analysis [7, 8].

Inductive Logic Programming (ILP) [5] performs relational learning either directly by manipulating first-order rules or through a process called propositionalization [9, 10, 11], which brings the relational task down to the propositional level by representing subsets of relations as features that can be used as attributes. In comparison with direct ILP, propositionalization normally exchanges accuracy for efficiency [12], as it enables the use of fast attribute-value learners [13, 10, 9], although the translation of first-order rules into features can cause information loss [14].

Much work has been done combining relational learning tasks with propositional learners, including decision trees or neural networks [15, 16, 17, 18, 19]. In this paper, we are interested in the, less investigated, inverse problem: how to extract first-order logic descriptions from propositional learners, in particular, neural networks, trained to solve relational learning tasks?

We extend the well-known CILP neural-symbolic system [3] to allow the extraction of meaningful first-order logic rules from trained neural networks. Propositionalization and subsequent attribute-value learning can destroy the original relational structure of the task at hand, so much so that the provision of interpretable relational knowledge following learning is made impossible [14]. In this paper, we show that by adapting the first-order version of the CILP system, called CILP++ [13], so as to enable the application of a variation of the well-known TREPAN knowledge extraction algorithm [20], a revised set of first-order rules can be extracted from trained neural networks efficiently and accurately, enabling first-order logical reasoning about what has been learned by the network. The result is a neural network, trained using an efficient backpropagation learning algorithm, and capable of receiving "first-order" examples as input and producing first order rules as output. The ability to perform reasoning directly opens a number of research and application possibilities integrating reasoning and learning [21, 7, 8, 22].

We have compared relational knowledge extraction in CILP++ with state-of-the-art ILP system Aleph [23] and Markov Logic Networks (MLN's) [15] on the Mutagenesis [7], UW-CSE [8], Alzheimer-amine [21] and Cora [22] datasets. Results indicate that the relational theories extracted from CILP++ have high fidelity to the trained neural network models, and that the use of neural networks can provide considerable speed-ups while achieving comparable accuracy and area under ROC curve results.

The choice of using MLN's and Aleph for empirical comparisons is due the nature of their methodology for tackling relational learning, which are distinctively different: MLN's take a probabilistic approach for the relational learning problem, by attempting to find a distribution model that fits the ground atoms of a hypothesis interpretation as best as possible, while Aleph performs relational learning by searching the Herbrand space [5] of possible literals for a given dataset.

The remainder of the paper is as follows: section 2 introduces CILP++, the neural-symbolic system that uses the proposed approach in this paper for relational knowledge extraction from trained neural networks. Section 3 presents obtained experimental results with CILP++ on the Mutagenesis, UW-CSE, Alzheimer-amine and Cora datasets, comparing against MLN's and Aleph. Lastly, section 4 discusses outcomes from the experiments performed and also does an overview of other systems that are closely related with the work being presented in this paper.

CILP++ is available from Sourceforge (`https://sourceforge.net/projects/cilppp/`) and experimental settings will be made available for download.

2 Relational Learning with Neural Networks

This section introduces the proposed system for relational knowledge extraction from trained neural networks. It starts by presenting each module of the CILP++ system and how they can be adapted to allow direct first-order knowledge extraction and inference from the trained models.

2.1 Bottom Clause Propositionalization

Relational learning with CILP++ starts by applying bottom clause propositionalization (BCP) onto the first-order examples set. Each first-order example, in the form of a instantiated target clause, e.g. $target(a_1, \ldots, a_n)$, is converted into a numerical vector that a neural network can use as input. In order to achieve this, each example is transformed into a bottom clause and mapped onto features on an attribute-value table, and numerical vectors are generated for each example. Thus, BCP has three steps: bottom clause generation, feature generation and attribute-value mapping.

Firstly, before describing each BCP step, we present three first-order concepts which are used in this work: *clause*, *relational domain* and *bottom clause*.

– A *clause* is a definition of relations between facts, with structure

$$p_t(V_{1_t},\dots,V_{n_t}) \text{ :- } p_1(V_{1_1},\dots,V_{n_1}), p_2(V_{1_2},\dots,V_{n_2}),\dots,p_m(V_{1_m},\dots,V_{n_m}),$$

where $\{p_i, 1 \leq i \leq m\} \bigcup \{p_t\}$ is a set of predicates (relations), V_j is a set of variables, $p_1(V_{1_1},\dots,V_{n_1}), p_2(V_{1_2},\dots,V_{n_2}),\dots,p_m(V_{1_m},\dots,V_{n_m})$ are literals and **:-** represents implication. Literals on the left-hand side of the consequence operator are known as head literals and literals on the right-hand side are known as body literals.

– A *relational domain* is a tuple $< E, BK >$, where: E is a set of ground atoms of target concept(s) (i.e. first-order logic examples), in which labels are truth-values; and BK is a set of clauses known as background knowledge, which can be *facts* (grounded single-literal clauses that define what is known about a given task) or *clauses*, as define above.

– A *bottom clause* is a boundary in the hypothesis search space during ILP learning [24], and is built from one random positive example, background knowledge and language bias (a set of clauses that define how clauses can be built in an ILP model). A bottom clause is the most specific clause (with most literals) that can be considered a candidate hypothesis.

Having introduced clauses and relational domain, we are in position to describe BCP. In the first step of BCP, bottom clause generation, each example e_i from a first-order example set E is given to the bottom clause generation algorithm [25] to create a corresponding bottom clause set $E_\perp$, containing one bottom clause $\perp_i$ for each example e_i. To do so, a slight modification is needed to allow the same *hash* function to be shared among all examples, in order to keep consistency between variable associations, and to allow negative examples to have bottom clauses as well; the original algorithm deals with positive examples only. An extensive algorithm description is provided in [13].

In order to illustrate each BCP step, we introduce a small family relationship relational domain [26], with background knowledge

$$BK = \{\textit{mother(mom1, daughter1)}, \textit{wife(daughter1, husband1)}, \textit{wife(daughter2, husband2)}\},$$

with one positive example and one negative example *motherInLaw(mom1, husband1)* and *motherInLaw(daughter1, husband2)*, respectively. It can be noticed that the relation between *mom1* and *husband1*, which the positive example establishes, can be alternatively described by the sequence of facts *mother(mom1, daughter1)* and *wife(daughter1, husband1)* in the background knowledge. This states semantically that *mom1* is a mother-in-law because *mom1* has a married daughter, namely, *daughter1*. Applied to this example, the bottom clause generation algorithm would create a clause $\perp_i = motherInLaw(A,B) \leftarrow$ *mother(A, C)*, *wife(C, B)*. Comparing $\perp$ with the sequence of facts above, we notice that $\perp_i$ describes one possible meaning of mother-in-law: "*A* is a mother-in-law of *B* if *A* is a mother of *C* and *C* is wife of *B*", i.e. the mother of a married daughter is a mother-in-law. To generate features from bottom clauses, BCP generates one bottom clause for each (positive or negative) example e, which we denote as $\perp_e$. At the end of the first step of BCP, we end with a bottom clause set containing both positive and negative examples:

$$\begin{aligned} E_\perp = \{&motherInLaw(A,B) : -mother(A,C), wife(C,B);\\ &\sim motherInLaw(A,B) : -wife(A,C)\} \qquad . \end{aligned}$$

In the second step, feature generation, a feature table F is generated from $E_\perp$. Earlier versions of CILP++ used bottom clause literals directly as features, but this approach can lead to inconsistencies if knowledge is to be extracted from models which used such features [27]. In order to tackle this, an adapted version of the first-order feature generation algorithm presented in [16] has been used to generate independent propositional features which represent first-order descriptions.

For illustrating the second step of BCP, consider the following bottom clause $R_\perp$:

$$\begin{aligned} motherInLaw(A,B) :-& parent(A,C), wife(C,B),\\ & wife(A,D), brother(B,D) \end{aligned}$$

Semi-propositionalization [16] is used to generate a set of first-order features for $R_\perp$. First-order features are sets of literals that share variables that are not inside any head literal. Those variables are known as local variables. From the family relationship example, the following features are obtained:

$$F_1 = \{parent(A,C), wife(C,B)\}$$
$$F_2 = \{wife(A,D), brother(B,D)\}$$

BCP treats each decomposition as a feature and in the example above, two clauses would be generated from the decomposition of $R_\perp$:

$$L_1(A,B) :- parent(A,C), wife(C,B)$$
$$L_2(A,B) :- wife(A,D), brother(B,D)$$

Therefore, $R_\perp$ can be rewritten as the following semi-propositional rule $R'_\perp$:

$$motherInLaw(A,B) :- L_1(A,B), L_2(A,B)$$

If the only example to be propositionalized by BCP is r, the feature table F would, at the end, contain only two elements: $L_1(A,B)$ and $L_2(A,B)$.

Lastly, in the third step of BCP, the feature table F is applied onto E in order to generate binary vectors that a neural network can process. The algorithm, implemented on CILP++, is as follows:

1. Let $|F|$ be the number of elements in F;
2. Let E_v be the set of binary vectors, converted from E, initially empty;
3. For each example $e_i \in E$ do
   (a) For each feature $f_j \in F$ do
      i. Query E against the correspondent first-order description L_j of f_j against the relational domain background knowledge BK;
      ii. If query succeeds, assign 1 to the position j binary vector v_i; if not, assign 0 instead;
   (b) Associate a label 1 to v_i if e_i is a positive example, and -1 otherwise;
   (c) Add v_i to E_v;
4. Return E_v.

Continuing the family relationship example: $|F|$ is equal to 2, since there is only two features in the table: $L_1(A,B)$ and $L_2(A,B)$. Since r contain both features on its bottom clause, $v_r = (1,1)$. See [13] for more details.

2.2 Neural Network Learning and Relational Knowledge Extraction

CILP++ uses resilient backpropagation [28], with *early stopping* [29] for learning. Resilient backpropagation takes into account only the sign of the partial derivative over all training examples (not the magnitude), and acts independently on each weight. For each weight, if there was a sign change of the partial derivative of the total error function compared to the last iteration, the update value for that weight is multiplied by a factor $\eta-$, where $\eta- < 1$. If the last iteration produced the same sign, the update value is multiplied by a factor of $\eta+$ where $\eta+ > 1$. The update values are calculated for each weight in the above manner, and finally each weight is changed by its own update value, in the opposite direction of that weight's partial derivative, so as to minimize the total error function. We set $\eta+$ and $\eta-$ through validation.

With early stopping, when the validation error measure starts to increase, training is stopped. We have used a more permissive version of early stopping [29], which does not halt training immediately after the validation error increases. It stops when a combined measure of both number of consecutive epochs with increasing validation set error and absolute value of current validation set error reaches a certain threshold.

Following network training, in order to perform relational knowledge extraction, an adapted version of the TREPAN rule extractor [20] is applied to the trained neural network. TREPAN is originally a m-of-n propositional tree inducer which uses a learned neural network as oracle and through a set of examples *S*, possibly distinct from the example set used for training the neural network, a decision tree is recursively built, based on an information gain-based heuristic. We adapted TREPAN in order to allow the generation and query of first-order rules into Prolog [30], a well-known general purpose logic programming. Several simplifications have also been done in order to improve efficiency and readability. The adapted pseudo-algorithm for TREPAN can be seen on Algorithm 1, based on the original TREPAN algorithm [20]. Changes from the original are highlighted with an underline.

Algorithm 1 Adapted TREPAN algorithm

```
 1: Inputs: training set $E$, feature set $F$
 2: for all example $e \in E$ do
 3:     class label for $e = Oracle(E)$
 4: end for
 5: initialize the root of the tree, $T$, as leaf node
 6: initialize queue with tuple $< T, E, \{\} >$
 7: while queue not empty and $\underline{size(T) < tree\_size\_limit}$ do
 8:     remove node $N$ from head of queue
 9:     $examples_n$ = example set stored with $N$
10:     $constraints_n$ = constraint set stored with $N$
11:     use $F$ to build set of candidate splits for node
12:     use $examples_n$ and calls to $Oracle(constraints_n)$ to evaluate splits
13:     $S$ = best binary split
14:     $\underline{\text{search for best m-of-n split, } S'\text{, using } S \text{ as seed}}$
15:     make $N$ an internal node with split $S'$
16:     for all possible outcomes $s \in S'$ do
17:         make $C$, a new child node of $N$
18:         $constraints_c = constraints_n \cup \{S' = s\}$
19:         use calls to $Oracle(constraints_c)$ to determine if $C$ should remain a leaf
20:         if not then
21:             $examples_c$ = members of $examples_n$ with outcome $s$ on split $S'$
22:             put $< C, example_c, constraints_c >$ into Queue
23:         end if
24:     end for
25: end while
26: $\underline{TH = empty}$
27: for all $\underline{\text{path } p \text{ in } T}$ do
28:     $\underline{\text{Create a rule } r \text{ containing conjunctions of each m-of-n split conditions } s \text{ inside } p}$
29:     $\underline{\text{Add all possible } m \text{ combinations of the } n \text{ conditions inside } s \text{ as disjunctive body literals in } r}$
30:     $\underline{\text{Add } r \text{ to } TH}$
31: end for
32: $\underline{\text{perform a decreasing ordering on } TH \text{ on the number of examples covered by each path } p}$
33: return a first-order theory $TH$
```

The adapted version of TREPAN presented on Algorithm 1 have the following differences when compared to original TREPAN:

- Line 7: Tree generation has been simplified, only maximum size criterion is used for stopping the process.
- Line 14: The search heuristic for best m-of-n split is now weighted by size of m. The original heuristic value for a given split is now subtracted by m/n.
- Lines 26-32: The m-of-n tree is transformed into a set of (possibly) disjunctive rules, in order to allow first-order inference with logic programming languages such as Prolog.

After extracting rules from the trained network (after obtaining TH), the definitions of the semi-propositional first-order features (L_i clauses) obtained during BCP are added to TH, resulting in a first-order theory that can be used to perform first-order inference. In the following, the well-known east-west trains ILP dataset [31] is used in order to demonstrate how CILP++ performs relational learning, relational knowledge extraction and reasoning.

In the first step of CILP++ (propositionalization with BCP), 20 bottom clauses were generated from the 10 positive and 10 negative examples of eastbound and westbound trains. From those bottom clauses, 41 features were obtained by using semi-propositionalization. Therefore, 41 input neurons will be created in CILP++'s initial neural network structure, each one representing one feature. A small sample of bottom clauses generated, the features generated with BCP and the resulting initial neural network structure are presented in Figure 1.

```
Bottom clauses

eastbound(A) :-
    has_car(A,B), long(B), wheels(B,2).
eastbound(A) :-
    has_car(A,B), has_car(A,C), long(C), double(B).
eastbound(A) :-
    has_car(A,B), has_car(A,C), long(B), shape(C,u_shaped).

Features

F1(A) :- has_car(A,B), long(B), wheels(B,2).
F2(A) :- has_car(A,B), double(B).
F3(A) :- has_car(A,B), long(B).
F4(A) :- has_car(A,C), shape(C, u_shaped).
```

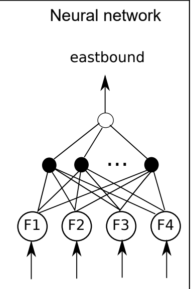

Figure 1: Bottom clauses and BCP features example for the east-west trains dataset.

After neural network training, the adapted TREPAN rule extractor algorithm (Algorithm 1) is used to generate first-order rules from the network. Leave-one-out cross-validation was used, i.e., 20 folds have been generated from the 20 first-order examples. Figure 2 shows the resulting first-order theory. The first part of the generated theory is the extracted TREPAN rules, whilst the second part is the added semi-propositional clauses generated by BCP.

```
Generated theory

eastbound(A) :- F1(A).
eastbound(A) :- F2(A).

Semi-propositionalization clauses

F1(A) :- has_car(A,B), short(B), wheels(B,1).
F2(A) :- has_car(A,B), short(B), closed(B), wheels(B, 2), jagged(B).
```

Figure 2: Theory obtained with CILP++ for the east-west trains dataset[1]

The obtained results for the east-west trains are:

- **Accuracy**: 90% for the trained network and 90% accuracy for the extracted rules (the extracted rules were evaluated in Prolog, by querying the first-order examples);
- **Fidelity**: 95% of fidelity between the trained neural network and the extracted rules. Fidelity is defined as the percentage of examples classified in the same way by both models. It does not matter if the result is a hit or a miss for a given example *e*: as long as both the neural network and the rules classify *e* identically, it is considered a hit towards fidelity.

[1]Compare with the rules extracted by LINUS in [16]; our method seems to produce more readable rules.

Table 1: Accuracy results (ACC) with standard deviations, area under ROC curve results (AUC) and runtimes in seconds (RUN) for the Mutagenesis, UW-CSE and Alzheimers-anime datasets

SYSTEM	**METRICS**	**DATASETS**			
		Mutagenesis	UW-CSE	Alzheimer-amine	Cora
Aleph	*ACC*	80.85%(±10.51)	85.11%(±7.11)	78.71%(±5.25)	−−
	AUC	0.80(0.02)	0.81(±0.07)	0.79(±0.09)	−−
	RUN	721	875	5465	−−
MLN	*ACC*	62.11%(±0.022)	75.01%(±0.028)	50.01%(±0.002)	70.10%(±0.191)
	AUC	0.67(±0.022)	0.76(±0.12)	0.51(±0.02)	0.809(±0.001)
	RUN	4341	1184	9811	97148
CILP++	*ACC*	91.70%(±5.84)	77.17%(±9.01)	79.91%(±2.12)	68.91%(±2.12)
	AUC	0.82(±0.02)	0.77(±0.07)	0.80(±0.01)	0.79(±0.02)
	RUN	851	742	3822	11435

Table 2: Rule accuracy results (ACC) with standard deviations and fidelity (FID) for the Mutagenesis, UW-CSE, Alzheimer-anime and Cora datasets

SYSTEM	**METRICS**	**DATASETS**			
		Mutagenesis	UW-CSE	Alzheimer-anime	Cora
Aleph	*ACC*	80.85%(±10.51)	85.11%(±7.11)	78.71%(±5.25)	−−
CILP++	*ACC*	77.72(±2.12)	81.98(±3.11)	78.70%(±6.12)	67.98%(±5.29)
	FID	0.89	0.90	0.88	0.82

3 Experimental Results

CILP++ has been tested empirically and results over ten-fold cross validation for the trained neural network can be seen on Table 1. CILP++ is being tested against a well-known ILP system, Aleph [23] and Markov Logic Networks (MLN's) [15]. Four relational domains have been used: Mutagenesis [7], UW-CSE [8], Alzheimers-anime [21] and Cora [22]. The same parameters as [13] have been used for training CILP++. For Aleph, the settings suggested in [18] have been used. For MLN's, the reported results on three publications [15, 32, 33] have been collected. Lastly, on TREPAN, $tree_size_limit$ has been set as 5. All experiments were run on a 3.2 Ghz Intel Core i3-2100 with 4 GB RAM.

Results show that CILP++ has comparable accuracy and AUC measurements with both Aleph and MLN's, while having considerably better runtimes. While CILP++ was able to run and generate competitive results on all tested datasets, Aleph ran out of memory while running Cora. Standard MLN's performed very poorly on Alzheimer-amine [32] and had higher training times.

Table 2 shows comparative results with Aleph for querying the extracted first-order rules from the trained CILP++ neural network on Prolog. Also, fidelity measurements are provided.

Results show that competitive accuracy with Aleph has been maintained after extraction, and also good fidelity measures have been obtained in comparison with the trained neural network. This indicates that CILP++ neural networks are capable of efficiently solve relational tasks with BCP.

4 Concluding Remarks

In this paper, we have presented an integrated and efficient method and system for the extraction of first-order logic rules from neural networks. Experimental results show that the first-order rules extracted from trained neural networks, in terms of accuracy and AUC, are comparable with a well-known probabilistic system for relational learning, MLN's, and a search-based ILP system, Aleph, while being considerably faster. Those results indicate the promise of CILP++ as a relational learner.

Further comparisons with related work include the analysis of propositionalization-based systems such as LINUS/DINUS/SINUS [9, 11] and RelF[34], which rely on the quality of their feature generation to reduce the information loss of the propositionalization approach and, consequently, within the rules extracted from the learner. Both the LINUS/DINUS/SINUS family of ILP systems and RelF generate a number of constrained first-order features f from the Herbrand base H (H is the set of possible clauses for a given domain knowledge). From the collection of features f, a final set of features F is obtained for representing the training examples, according to a given score function. CILP++, on the other hand, uses the concept of bottom clause, which is a clause that uniquely describes a single relational example. CILP++ uses bottom clauses to train a neural network, and an algorithm based on the concept of semi-propositionalization [16, 27] to generate F.

Approaches based on Bayesian networks [35] also perform relational learning, but represent the learned knowledge without the use of explicit relational rules. Statistical Relational Models [36] contain a rich representation language which combines a frame-based logical representation with probabilistic semantics based on bayesian networks. BLOG [37] is a first-order probabilistic modeling language that specifies probability distributions over possible worlds with varying sets of objects. A BLOG model contains statements that define conditional probability distributions for a certain set of random variables; the model also specifies certain context-specific independence properties. Inference is done on BLOG using Markov Chain Monte Carlo [38] algorithms. In CILP++, inference is deterministic with first-order rules learned from the neural (statistical) model being applicable directly onto a Prolog theorem prover for reasoning.

As future work, a study on how CILP++ deals with noisy datasets (noise in the background knowledge and/or examples) can provide interesting results, due to how backpropagation naturally deals with incomplete data and noisy inputs. Also, an investigation on how CILP++ can be adapted to deal directly with numeric data can overcome a well-known flaw in ILP systems, which is its inability to deal directly with numbers. ILP systems use auxiliary predicates to indicate relations between numeric variables such as greater-than, less-than and so on.

References

[1] S. Hölldobler and Y. Kalinke, "Towards a massively parallel computational model for logic programming," in *In: Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77, 1994.

[2] G. G. Towell and J. W. Shavlik, "Knowledge-Based Artificial Neural Networks," *Artif. Intell.*, vol. 70, no. 1-2, pp. 119–165, 1994.

[3] A. S. D. Garcez and G. Zaverucha, "The Connectionist Inductive Learning and Logic Programming System," *Applied Intelligence*, vol. 11, pp. 59–77, 1999.

[4] R. Basilio, G. Zaverucha, and V. Barbosa, "Learning Logic Programs with Neural Networks," in *Inductive Logic Programming*, vol. 2157 of *LNAI*, pp. 15–26, Springer Berlin / Heidelberg, 2001.

[5] S. Džeroski and N. Lavrač, *Relational Data Mining*. Relational Data Mining, Springer, 2001.

[6] L. De Raedt, *Logical and Relational Learning*. Cognitive Technologies, Springer, 2008.

[7] A. Srinivasan and S. H. Muggleton, "Mutagenesis: ILP experiments in a non-determinate biological domain," in *Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien*, pp. 217–232, 1994.

[8] J. Davis, E. S. Burnside, I. de Castro Dutra, D. Page, and V. S. Costa, "An integrated approach to learning bayesian networks of rules," in *ECML* (J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, eds.), vol. 3720 of *Lecture Notes in Computer Science*, pp. 84–95, Springer, 2005.

[9] N. Lavrač and S. Džeroski, *Inductive logic programming: techniques and applications*. Ellis Horwood series in artificial intelligence, E. Horwood, 1994.

[10] F. Železný and N. Lavrač, "Propositionalization-based Relational Subgroup Discovery With RSD," *Machine Learning*, vol. 62, pp. 33–63, 2006.

[11] S. Kramer, N. Lavrač, and P. Flach, "Relational Data Mining," ch. Propositionalization approaches to relational data mining, pp. 262–286, New York, NY, USA: Springer-Verlag New York, Inc., 2000.

[12] M.-A. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel, "Comparative Evaluation Of Approaches To Propositionalization," in *ILP*, vol. 2835 of *LNAI*, pp. 194–217, Springer-Verlag, 2003.

[13] M. V. M. França, G. Zaverucha, and A. dAvila Garcez, "Fast relational learning using bottom clause propositionalization with artificial neural networks," *Machine Learning*, vol. 94, no. 1, pp. 81–104, 2014.

[14] M. V. M. França, A. S. D. Garcez, and G. Zaverucha, "Relational Knowledge Extraction from Attribute-Value Learners," in *2013 Imperial College Computing Student Workshop*, vol. 35 of *OpenAccess Series in Informatics (OASIcs)*, (Dagstuhl, Germany), pp. 35–42, Schloss Dagstuhl, 2013.

[15] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, pp. 107–136, 2006.

[16] N. Lavrač and P. A. Flach, "An extended transformation approach to inductive logic programming," *ACM Trans. Comput. Logic*, vol. 2, no. 4, pp. 458–494, 2001.

[17] B. Kijsirikul and B. K. Lerdlamnaochai, "First-Order Logical Neural Networks," *Int. J. Hybrid Intell. Syst.*, vol. 2, pp. 253–267, Dec. 2005.

[18] A. Paes, F. Železný, G. Zaverucha, D. Page, and A. Srinivasan, "ILP Through Propositionalization and Stochastic k-Term DNF Learning," in *ILP*, vol. 4455 of *LNAI*, (Berlin, Heidelberg), pp. 379–393, Springer-Verlag, 2007.

[19] R. Basilio, G. Zaverucha, and A. S. Garcez, "Inducing Relational Concepts with Neural Networks via the LINUS System," in *In ICONIP*, pp. 1507151–0, 1998.

[20] M. Craven and J. W. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," in *NIPS*, pp. 24–30, 1995.

[21] R. King and A. Srinivasan, "Relating chemical activity to structure: An examination of ILP successes," *New Generation Computing*, vol. 13, no. 3-4, pp. 411–434, 1995.

[22] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, (New York, NY, USA), pp. 39–48, ACM, 2003.

[23] A. Srinivasan, "The Aleph System, version 5." http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html, 2007. Last accessed on may/2013.

[24] S. Muggleton, "Inverse Entailment and Progol," *New Generation Computing, Special issue on Inductive Logic Programming*, vol. 13, no. 3-4, pp. 245–286, 1995.

[25] A. Tamaddoni-Nezhad and S. Muggleton, "The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause," *Machine Learning*, vol. 76, no. 1, pp. 37–72, 2009.

[26] S. Muggleton and L. D. Raedt, "Inductive Logic Programming: Theory and Methods," *Journal of Logic Programming*, vol. 19, no. 20, pp. 629–679, 1994.

[27] M. V. M. França, G. Zaverucha, and A. S. D. Garcez, "Neural relational learning through semi-propositionalization of bottom clauses," in *AAAI Spring Symposium Series*, 2015.

[28] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.

[29] L. Prechelt, "Early stopping - but when?," in *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pp. 55–69, Springer-Verlag, 1997.

[30] R. A. Kowalski, "The early years of logic programming," *Commun. ACM*, vol. 31, pp. 38–43, Jan. 1988.

[31] J. Larson and R. S. Michalski, "Inductive inference of VL decision rules," *SIGART Bull.*, no. 63, pp. 38–44, 1977.

[32] T. N. Huynh and R. J. Mooney, "Discriminative structure and parameter learning for markov logic networks," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), pp. 416–423, ACM, 2008.

[33] S. Kok and P. Domingos, "Learning the structure of markov logic networks," in *Proceedings of the 22Nd International Conference on Machine Learning*, (New York, NY, USA), pp. 441–448, ACM, 2005.

[34] O. Kuželka and F. Železný, "Block-wise construction of tree-like relational features with monotone reducibility and redundancy," *Machine Learning*, vol. 83, pp. 163–192, 2011.

[35] J. Pearl, *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[36] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *In IJCAI*, pp. 1300–1309, Springer-Verlag, 1999.

[37] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov, "BLOG: probabilistic models with unknown objects," IJCAI, (San Francisco, CA, USA), pp. 1352–1359, Morgan Kaufmann Publishers Inc., 2005.

[38] A. E. Gelfand and A. F. M. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American Statistical Association*, vol. 85, no. 410, pp. 398–409, 1990.