Neural Relational Learning Through Semi-Propositionalization of Bottom Clauses

Manoel Vitor Macedo França
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB

Gerson Zaverucha
Prog. de Eng. de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil 21941–972

Artur S. d'Avila Garcez
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB

Abstract

Relational learning can be described as the task of learning first-order logic rules from examples. It has enabled a number of new machine learning applications, e.g. graph mining and link analysis in social networks. The CILP++ system is a neural-symbolic system which can perform efficient relational learning, by being able to process first-order logic knowledge into a neural network. CILP++ relies on BCP, a recently discovered propositionalization algorithm, to perform relational learning. However, efficient knowledge extraction from such networks is an open issue and features generated by BCP do not have an independent relational description, which prevents sound knowledge extraction from such networks. We present a methodology for generating independent propositional features for BCP by using semi-propositionalization of bottom clauses. Empirical results obtained in comparison with the original version of BCP show that this approach has comparable accuracy and runtimes, while allowing proper relational knowledge representation of features for knowledge extraction from CILP++ networks.

Introduction

Relational learning can be described as the task of learning a first-order logic theory from examples (Džeroski and Lavrač 2001; De Raedt 2008). 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. This enables a range of applications of machine learning, for example in bioinformatics, graph mining and link analysis in social networks, serious games, and responsible gambling (Bain and Muggleton 1994; Srinivasan and Muggleton 1994; Džeroski and Lavrač 2001; King and Srinivasan 1995; Dragicevic et al. 2013). Inductive Logic Programming (ILP) (Muggleton and Raedt 1994; Nienhuys-Cheng and de Wolf 1997) performs relational learning either directly by manipulating first-order rules or through a method called propositionalization (Lavrač and Džeroski 1994; Železný and Lavrač 2006), which brings the relational task down to the propositional level by representing subsets of relations as features that can then be used as attributes. In comparison with full ILP, propositionalization normally exchanges accuracy for efficiency (Krogel et al. 2003), as it enables the use of fast attribute-value learners such as decision trees or even neural networks (Quinlan 1993; Rumelhart, Widrow, and Lehr 1994), although the translation of first-order rules into features can incur information loss.

Bottom Clause Propositionalization (BCP) (França, Zaverucha, and Garcez 2013) is a recent propositionalization method, based on one of the search boundaries used by traditional ILP learners: bottom clauses. Bottom clauses are boundaries in the hypothesis search space, first introduced by Muggleton, 1995 as part of the Progol system, and are built from one random positive example, background knowledge (a set of clauses that describe what is known) and language bias (a set of clauses that define how clauses can be built). A bottom clause is the most specific clause (with most literals) that can be considered a candidate hypothesis. BCP uses bottom clauses for propositionalization because their body literals can be used directly as features in a truth-table, simplifying the feature extraction process (Muggleton and Tamaddoni-Nezhad 2008; DiMaio and Shavlik 2004; Pitangui and Zaverucha 2012).

The neural-symbolic system CILP has been shown effective at learning and reasoning from propositional data in a number of domains (Garcez and Zaverucha 1999; Garcez, Broda, and Gabbay 2002; Garcez, Lamb, and Gabbay 2008). CILP uses background knowledge in the form of propositional logic programs to build a neural network, which is in turn trained by examples using backpropagation (Rumelhart, Widrow, and Lehr 1994). A recent extension of CILP, called CILP++ (França, Zaverucha, and Garcez 2013), incorporates BCP as a novel propositionalization method and, differently from CILP, CILP++ networks are first-order in that each neuron denotes a first-order atom. Yet, for learning, CILP++ uses the same neural model as CILP, by transforming each first order example into a bottom clause. Experimental evaluations reported at (França, Zaverucha, and Garcez 2013) show that such a combination can lead to efficient learning of relational concepts.

When dealing with relational learning, it is mandatory to be able to generate rules that can describe what has been learned (Muggleton and Raedt 1994). Therefore, a propositionalization-based system must be able to extract

rules from the learned model. However, efficient knowledge extraction on neural networks is an open issue and features generated by BCP do not have a relational description, which unables sound knowledge extraction on such networks. In this paper, we present a methodology for generating independent propositional features for BCP which can be described by a first-order clause, by using semi-propositionalization. We call this approach $\text{BCP}_{semi-prop}$. The notion of semi-propositionalization, first introduced by an extension of the ILP system LINUS (Lavrač and Flach 2001) (henceforth called Extended LINUS in this paper), offers a way of obtaining sets of independent clauses which are semantically equivalent to a given clause. Our approach extracts such sets of independent clauses from bottom clauses and uses them as features. Empirical results show that our approach has comparable accuracy and runtimes with the original BCP. This indicates that efficiency results from (França, Zaverucha, and Garcez 2013) can be maintained while enabling, as future work, sound extraction from such networks and reasoning in a relational level, by applying a propositional rule extractor from neural networks such as TREPAN (Craven and Shavlik 1995).

BCP and CILP++

The first step of relational learning with CILP++ is BCP. Each instantiated target clause such as $target(a_1,\ldots,a_n)$ is converted into a numerical vector that an ANN 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 two steps: bottom clause generation and attribute-value mapping.

In the first step, each example is given to Progol's bottom clause generation algorithm (Tamaddoni-Nezhad and Muggleton 2009) to create a corresponding bottom clause representation. 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.

We are going to illustrate Progol's bottom clause generation with an example. Consider the well-known family relationship example (Muggleton and Raedt 1994), with background knowledge B = {*mother(mom1, daughter1)*, *wife(daughter1, husband1)*, *wife(daughter2, husband2)*}, with positive example *motherInLaw(mom1, husband1)*, and negative example *motherInLaw(daughter1, husband2)*. 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 of Progol would create a clause $\bot = motherInLaw(A,B) \leftarrow mother(A, C), wife(C, B)$. Comparing $\bot$ with the sequence of facts above, we notice that $\bot$ 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. This is why, in this paper, we investigate learning from bottom clauses. However, Progol generates only one bottom clause, for limiting its search space. To learn from bottom clauses, BCP generates one bottom clause for each (positive or negative) example e, which we denote as $\bot_e$. At the end of the bottom clause generation process, we end with a bottom clause set

$$E_\bot = \{motherInLaw(A,B) : -mother(A,C), wife(C,B); \\ \sim motherInLaw(A,B) : -wife(A,C)\}.$$

After the creation of the $\text{E}_\bot$ set, the second step of BCP is as follows: each element of $\text{E}_\bot$ (each bottom clause) is converted into an input vector v_i, $0 \leq i \leq n$, that a propositional learner can process. The algorithm for that, implemented by CILP++, is as follows:

1. Let $|L|$ be the number of distinct body literals (antecedents) in $E_\bot$;
2. Let E_v be the set of input vectors, converted from $E_\bot$, initially empty;
3. For each bottom clause $\bot_e$ of $E_\bot$ do
   (a) Create a numerical vector v_i of size $|L|$ and with 0 in all positions;
   (b) For each position corresponding to a body literal of $\bot_e$, change its value to 1;
   (c) Add v_i to E_v;
   (d) Associate a label 1 to v_i if e is a positive example, and -1 otherwise;
4. Return E_v.

After BCP is applied on the relational (first-order) examples, CILP++ training is next. Following CILP, CILP++ uses backpropagation. Given a bottom clause set $E_\bot^{train}$, the steps below are followed for training:

1. For each bottom clause $\bot_e \in E_\bot^{train}$, $\bot_e = h$:- $l_1, l_2, ..., l_n$, do
   (a) Add all $l_i, 1 \leq i \leq n$, that are not represented yet in the input layer as new neurons;
   (b) If h does not exist yet in the network, create an output neuron corresponding to it;
2. Add new hidden neurons, if required for learning;
3. Make the network fully-connected, by adding weights with zero value;
4. Apply backpropagation using each $\bot_e \in E_\bot^{train}$.

Additionally, notice that BCP does not combine first-order literals to generate features like Extended LINUS: it treats each literal of a bottom clause as a feature. The hidden layer of the ANN can be seen as a (weighted) combination of the features provided by the input layer. Thus, ANNs can combine propositional features when processing the data, but doing so dynamically (during learning), due to small changes of real-valued weights.

After training, CILP++ can be evaluated. Firstly, each test example e^{test} from a test set E^{test} is propositionalized with

BCP, resulting in a propositional data set $E_{\perp}^{test}$, where each $e^{test} \in E^{test}$ has a corresponding $\perp_e^{test} \in E_{\perp}^{test}$. Then, each $\perp_e^{test}$ is tested: each input neuron corresponding to a body literal of $\perp_e^{test}$ receives input 1 and all other input neurons (input neurons which labels are not present in $\perp_e^{test}$) receive value 0. Lastly, a feedforward pass through the network is performed, and the output will be CILP++'s answer to $\perp_e^{test}$ and consequently, to e^{test}.

BCP$_{semi-prop}$: Extracting Independent Features from Bottom Clauses

The CILP++ system has scope for improvements due to its limited ability to perform knowledge extraction from the trained network:

- Since an ANN, which is a propositional system, will be used to learn from BCP-propositionalized examples, most relational information regarding input-output properties of bottom clause literals (i.e., BCP features) will be lost;
- Propositional rules extracted from the ANN, after learning, will not have a relational meaning, not allowing relational reasoning after learning.

Tackling those issues is the main goal of the work presented here, and it is done by ensuring that each BCP feature has an **independent relational description**.

In this section, an approach that tackles the issues described above is presented, based on Extended LINUS. The propositionalization methodology of Extended LINUS groups first-order literals in order to define a propositional feature, by using the definition of **semi-propositionalized rules**. In a nutshell, semi-propositionalized rules are rules that cannot be decomposed into a set S of two or more rules that have equivalent semantic meaning as the original first-order rule.

Definition 1 (Semi-Propositional Rules)**.** (Lavrač and Flach 2001) The variables occurring in the head (consequent of a clause) of the rule are called global variables. Variables occurring only in the body are called local variables. A Prolog rule (i.e., a definite horn clause) in which there are no local variables is called constrained; a constrained rule in which every global variable occurs once in every literal is called *semi-propositional*.

Definition 2 (First-Order Features)**.** (Lavrač and Flach 2001) For any two body literals L_1 and L_2 of a given rule, L_1 and L_2 belong to the same equivalence class, $L_1 \sim_{lv} L_2$ iff they share a local variable. Clearly, $\sim_{lv}$ is reflexive and symmetric, and hence its transitive closure $=_{lv}$ is an equivalence relation inducing a partition on the body. The conjunction of literals in an equivalence class is called a *first-order feature*.

Proposition 3 (Decomposition of Rules)**.** *(Lavrač and Flach 2001) Let R be a Prolog rule, and let R' be constructed as follows. Replace each first-order feature F in the body of R by a literal L consisting of a new predicate with R's global variable(s) as argument(s), and add a rule $L :- F$. R' together with the newly constructed rules is equivalent to R, in the sense that they have the same success set. R' is also a semi-propositional rule.*

The methodology we propose in this paper is to transform bottom clauses into semi-propositional rules by applying Proposition 3 and thus, obtaining features with a first-order description. To illustrate this, consider the following bottom clause $R_{\perp}$ (with the addition of the concept *brother/2*):

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

The original version of BCP would generate four features from $R_{\perp}$: *parent(A,C)*, *wife(C,B)*, *wife(A,D)*, *brother(B,D)*, i.e., the literals themselves are the features. Therefore, neurons of the input layer of the underlying ANN of CILP++ would represent bottom clause body literals. A relational example e would cause an input neuron i_n activation only if its correspondent bottom clause $\perp_e$ contains the literal represented by i_n. Figure 1 exemplifies how an example propositionalized with the original BCP is used for training.

On the other hand, only one possible set of first-order features (see Definition 2) for $R_{\perp}$ can be found:

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

BCP$_{semi-prop}$ will treat each decomposition as a feature, instead of assuming that literals are features. 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 $R'_{\perp}$:

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

In this case, neurons of the input layer of the underlying ANN of CILP++ would represent the features L_i. A relational example e would cause an input neuron i_n activation only if its correspondent bottom clause $\perp_e$ contains all the body literals of the feature L_n, represented by i_n. Figure 1 illustrates this process.

BCP$_{semi-prop}$ has also been tested empirically and results over ten-fold cross validation can be seen on Table 1, showing that using BCP$_{semi-prop}$ provides comparable accuracy and runtimes over the original BCP. Moreover, fewer features are generated and thus, a smaller network is built.

Concluding Remarks

In this paper, we have presented a new methodology for BCP, by generating propositional features with a relational description, through semi-propositionalization of bottom clauses, called BCP$_{semi-prop}$. Experimental results show that the features generated with the semi-propositionalization of bottom clauses are comparable with the original BCP, in terms of both accuracy and runtimes, although our approach generates features with a relational description. Therefore, as future work, we will automatically extract relational rules from the trained neural network by using TREPAN in the domain of responsible gambling, where we have access to data and domain expertise as part of the BetBuddy research project.

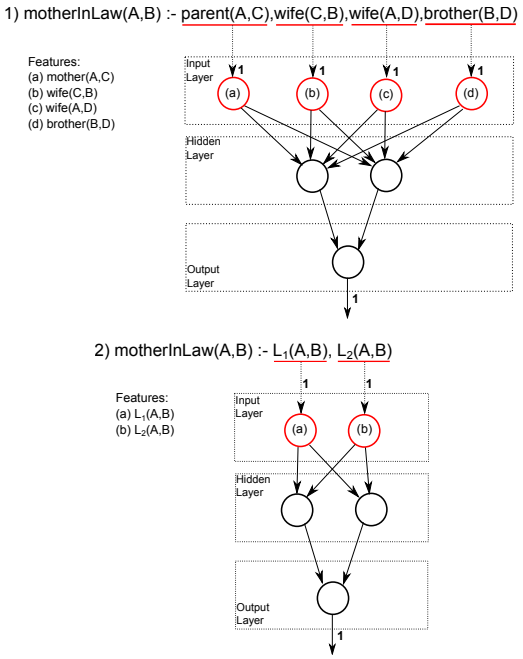

Figure 1: Illustration of CILP++'s training step when using the original BCP (1) and our new approach for BCP (2). The shown output values are the labels which are used for backpropagation training. Notice that our approach leads to a more compact neural network.

Table 1: Accuracy results (with standard deviation), runtimes and number of features for the Alzheimers benchmark (accuracies in the first line, runtimes in the second line, and number of features in the third line). $BCP_{semi-prop}$ has obtained comparable accuracy and runtimes with $BCP_{original}$, although generating less features.

	Alz-ami	***Alz-ace***	***Alz-mem***	***Alz-tox***
$BCP_{original}$	78.82(±5.25) 0:19:49 721	65.34(±4.6) 0:23:21 875	67.41(±5.7) 0:25:11 856	80.5(±4.83) 0:17:41 799
$BCP_{semi-prop}$	81.72(±2.12) 0:17:54 716	67.98(±2.13) 0:22:43 644	70.11(±4.12) 0:19:08 567	80.44(±3.84) 0:17:11 618

Acknowledgments

The authors are grateful for Luc De Raedt for his useful comments. The second author is thankful for the Brazilian agencies CNPq and FACEPE.

References

Bain, M., and Muggleton, S. 1994. Learning optimal chess strategies. In *Machine Intelligence 13*, 291.

Craven, M., and Shavlik, J. W. 1995. Extracting Tree-Structured Representations of Trained Networks. In *NIPS*, 24–30.

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

DiMaio, F., and Shavlik, J. W. 2004. Learning an Approximation to Inductive Logic Programming Clause Evaluation. In *ILP*, volume 3194 of *LNAI*, 80–97.

Dragicevic, S.; Percy, C.; Kudic, A.; and Parke, J. 2013. A descriptive analysis of demographic and behavioral data from internet gamblers and those who self-exclude from online gambling platforms. *Journal of Gambling Studies* 1–28.

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

França, M. V. M.; Zaverucha, G.; and Garcez, A. S. D. 2013. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning* 1–24.

Garcez, A. S. D., and Zaverucha, G. 1999. The Connectionist Inductive Learning and Logic Programming System. *Applied Intelligence* 11:59–77.

Garcez, A. S. D.; Broda, K. B.; and Gabbay, D. M. 2002. *Neural-Symbolic Learning System: Foundations and Applications*. Perspectives in Neural Computing Series. Springer Verlag.

Garcez, A. S. D.; Lamb, L. C.; and Gabbay, D. M. 2008. *Neural-Symbolic Cognitive Reasoning*. Springer Publishing Company, Incorporated, 1 edition.

King, R., and Srinivasan, A. 1995. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):411–434.

Krogel, M.-A.; Rawles, S.; Železný, F.; Flach, P.; Lavrač, N.; and Wrobel, S. 2003. Comparative Evaluation Of Approaches To Propositionalization. In *ILP'2003*, volume 2835 of *LNAI*, 194–217. Springer-Verlag.

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

Lavrač, N., and Flach, P. A. 2001. An extended transformation approach to inductive logic programming. *ACM Trans. Comput. Logic* 2(4):458–494.

Muggleton, S., and Raedt, L. D. 1994. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming* 19(20):629–679.

Muggleton, S., and Tamaddoni-Nezhad, A. 2008. QG/GA: a stochastic search for Progol. *Machine Learning* 70:121–133.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):245–286.

Nienhuys-Cheng, S., and de Wolf, R. 1997. *Foundations of Inductive Logic Programming*, volume 1228 of *LNAI*. Springer.

Pitangui, C. G., and Zaverucha, G. 2012. Learning theories using estimation distribution algorithms and (reduced) bottom clauses. In *ILP*, volume 7207 of *LNAI*, 286–301. Berlin, Heidelberg: Springer-Verlag.

Quinlan, J. R. 1993. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Rumelhart, D. E.; Widrow, B.; and Lehr, M. A. 1994. The Basic Ideas In Neural Networks. *Commun. ACM* 37(3):87–92.

Srinivasan, A., and Muggleton, S. H. 1994. 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*, 217–232.

Tamaddoni-Nezhad, A., and Muggleton, S. 2009. The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. *Machine Learning* 76(1):37–72.

Železný, F., and Lavrač, N. 2006. Propositionalization-based Relational Subgroup Discovery With RSD. *Machine Learning* 62:33–63.