Learning Datalog Programs from Input and Output

Yisong Wang, Xu Wang, and Yi Huang

Department of Computer Science, Guizhou University, Guiyang 550025, P.R. China

Abstract. A new inductive learning task is proposed for Datalog programs. In the learning task an example is a pair $\langle I, O\rangle$ where I and O, standing for input and output respectively, are sets of ground atoms. It means that O is the the least (Herbrand) model of a learned Datalog program when it is given the input I. An inductive learning algorithm is presented for this inductive learning task. It is a modular and anytime algorithm.

1 Introduction

Learning from examples and background knowledge, coined inductive logic programming (ILP in short) [1–3], plays a crucial role in knowledge discovering [4]. In general, different forms of examples motivate different learning settings. For instance, an example is a logic interpretation in learning from interpretation [5]; an example corresponds to an observation about the truth or falsity of a formula in learning from entailment [6]; an example is a proof learning from proofs [7].

We consider a new form of examples in an inductive learning task. Informally speaking, these examples are in the form of a pair $\langle I, O\rangle$, where I and O are sets of ground atoms. Intuitively, it means that if it is given the input I then its output is (or should be) O (under least model semantics).

We propose a new inductive learning task for Datalog programs where an example contains both input and output. An inductive learning algorithm is presented for this learning task, whose properties are analyzed.

The above learning setting approach is quite usual in debugging programs, and thus it is helpful in debugging Datalog programs by providing revision proposal. It is very common in locating possible problematic components of an imperative program by running the program with input, whose output implicitly hints somewhere may be wrong. As a matter of fact, inductive learning approaches have been investigated in debugging declarative programming [8–10].

2 Preliminaries

We assume a first-order language $\mathcal{L}$ without proper function symbols, *i.e.*, function symbols with arity greater than 0. The signature Σ of $\mathcal{L}$ includes a set $\mathcal{P}$ of predicate symbols and a set $\mathcal{C}$ of constants, the latter is the *domain* of variables of $\mathcal{L}$. The *terms* and *atomic formulas* (*atoms* in short) of $\mathcal{L}$ are defined as usual. The set of atoms constructing from $\mathcal{P}$ and $\mathcal{C}$ is denoted by $\mathcal{A}$. For our following learning purpose we assume both $\mathcal{P}$ and $\mathcal{C}$ are finite.

A *(Datalog) rule* [11] r is an expression of the form

$$A \leftarrow B_1, \ldots, B_n \ (n \geq 0) \tag{1}$$

where A, B_i $(1 \leq i \leq n)$ are atoms of $\mathcal{L}$ such that each variable occurring in A must occur in at least one of B_i $(1 \leq i \leq n)$. We denote *hd*$(r) = A$ and *bd*$(r) = \{B_i | 1 \leq i \leq n\}$. A *Datalog program* is a finite set of Datalog rules. Please note that Datalog programs do not permit true function symbols, while Horn logic programs do [12].

A *substitution* θ is a function from variables to terms, which is written in the following form

$$\{x_1/t_1, \ldots, x_n/t_n\} \tag{2}$$

where $x_i s$ $(1 \leq i \leq n)$ are pair-wise distinct variables and $t_i s$ $(1 \leq i \leq n)$ are terms (of the language $\mathcal{L}$). The *application* $e\theta$ of θ to an expression e is obtained from e by simultaneously replacing all occurrences of each variable x_i in e with the same term t_i, and $e\theta$ is called an *instance* of e [12]. The substitution θ is *ground* if t_i contains no variables for every x_i/t_i in θ. If the instance $e\theta$ of e contains no variable then it is a *ground instance* of e. The *ground* of a Datalog program P, written *ground*(P), is the set $\bigcup_{r \in P}$ *ground*(r), where

$$\textit{ground}(r) = \{r\theta \mid r\theta \text{ is a ground instance of } r\}. \tag{3}$$

Lemma 1. *Let P be a Datalog program and Q be a Datalog program without variables. Then ground$(P \cup Q)$ = ground$(P) \cup Q$.*

A *(Herbrand) interpretation* I is a set of atoms. An interpretation I *satisfies* a ground rule r if I satisfies *bd*(r) implies I satisfies *hd*(r). It *satisfies* a Datalog rule r if I satisfies every ground rule in *ground*(r). It *satisfies* a Datalog program P if it satisfies every rule in the Datalog program P. We use $\models$ to denote the satisfaction relation.

Formally speaking, two Datalog programs P_1 and P_2 are *uniformly equivalent* if $P_1 \cup I$ and $P_2 \cup I$ have the same least model, for each Herbrand interpretation I [13–15].

A rule r *subsumes* a rule r' if there exists a substitution θ such that *hd*$(r)\theta =$ *hd*(r') and *bd*$(r)\theta \subseteq$ *bd*(r'). In this case, r is *more (or equally) general than* r', and r' is *less (or equally) general than* r. We denote it by $r \preceq r'$. A rule r *properly subsumes* a rule r', written $r \prec r'$, if r subsumes r' but r' does not subsume r.

Lemma 2. *Let r and r' be two rules s.t. r' mentions no variable. Then r subsumes r' if and only if there exists a rule $r^* \in$ ground(r) s.t. r^* subsumes r'.*

Recall that Sagiv shows that every Datalog program P can be minimized under uniform equivalence by removing redundant atoms from bodies of rules and by removing redundant rules from P [13]. The following corollary follows from Sagiv's minimizing algorithm.

Corollary 1. *Let P be a Datalog program and r be a rule in P.*

(i) Let $P' = (P \setminus \{r\}) \cup \{r'\}$ with hd(r') = hd(r) and bd(r') = bd$(r) \setminus M$ where M is the least model of P. Then P and P' are uniformly equivalent.

(ii) Let $P' = P \setminus \{r\}$. If there exists a rule $r' \in P'$ s.t. $r' \preceq r$ then P and P' are uniformly equivalent.

3 Learning from Input and Output

In this section we formalize our inductive learning task from examples of input and output. An inductive learning algorithm for the learning task is presented and its properties are investigated.

3.1 The Learning Task

A *background theory* is a Datalog program. An *example* is a pair $\langle I, O\rangle$ with $I \subseteq O \subseteq \mathcal{A}$, which means that the output is O when the input is I. As Datalog programs are concerned in our learning scenario and the output of a Datalog program is normally taken as its least model, so the input I must be contained in the output O for the example $\langle I, O\rangle$.

In addition, when considering two examples $\langle I_1, O_1\rangle$ and $\langle I_2, O_2\rangle$ in a learning task, to guarantee the existence of a solution, they must satisfy the following conditions:

- $I_1 \subseteq I_2$ implies $O_1 \subseteq O_2$, (monotonicity)
- $I_1 \subseteq O_2$ implies $O_1 \subseteq O_2$. (convergence)

For instance, let $E = \{\langle\{p\}, \{p, q\}\rangle, \langle\{q\}, \{q, r\}\rangle\}$. There is no Datalog program P such that the least model of $P \cup I$ is O for every $\langle I, O\rangle$ in E. As a matter of fact, the two examples in E violate the convergence condition.

Now we formalize the above constraints as the notion of coherence. A set E of examples is *coherent* if

- $I \subseteq O$ for every $\langle I, O\rangle \in E$, and
- each two distinct examples in E satisfy monotonicity and convergence.

Further more, a set E of observations is *consistent w.r.t.* a background theory B if $O \models B$ for every $\langle I, O\rangle$ in E. Now we are in the proposition to formalize our learning task.

Definition 1 (Inductive learning from Input and Output). *An* inductive learning task *is, given a background theory B and a set E of examples, to find a hypothesis H (a Datalog program) such that, for every example $\langle I, O\rangle$ in E, O is the least (Herbrand) model of $B \cup H \cup I$.*

The above inductive learning task is denoted by *ILT*(B, E). A hypothesis H of the inductive learning task *ILT*(B, E) is called a *solution* to *ILT*(B, E).

Example 1. Let us consider the following inductive learning tasks.

- Let $B_1 = \{p \leftarrow q\}$ and $E_1 = \{\langle\{q\}, \{p, q\}\rangle, \langle\{p\}, \{p, q\}\rangle\}$. Then $H_1 = \{q \leftarrow p\}$ is a solution to the inductive learning task *ILT*(B_1, E_1).
- Let $B_2 = \{p(x) \leftarrow q(x)\}$ and $E_2 = \{\langle\{q(a)\}, \{p(a), q(a)\}\rangle, \langle\{p(a)\}, \{p(a), q(a)\}\rangle\}$. Then it is not difficult to verify that $H_2 = \{q(x) \leftarrow p(x)\}$ is a solution of *ILT*(B_2, E_2). What's more, one more general solution is $H_2' = \{p(x) \leftarrow q(y)\}$. In the case $E_2' = E_2 \cup \{\langle\{q(b)\}, \{p(b), q(b)\}\rangle\}$, H_2 is still a solution to *ILT*(B_2, E_2'), but H_2' is not.

The following property identifies the sufficient and necessary condition for the existence of a solution to an inductive learning task.

Proposition 1. *Given an inductive learning task ILT(B, E) where B is a background theory and E is a set of examples, there exists a solution H to ILT(B, E) if and only if E is coherent and E is consistent w.r.t. B.*

Based on the above proposition, we assume that the given examples in E are coherent and are consistent *w.r.t.* the background theory in the inductive learning task *ILT*(B, E) in what follows, unless explicitly stated otherwise.

3.2 An Inductive Learning Algorithm

In the following we present an algorithm for the inductive learning task. Firstly we construct Datalog programs P and P^* for a given set E of examples and a background theory B as follows.

Definition 2. *Let $E = \{\langle I_i, O_i\rangle | 0 \leq i \leq n-1\}$ be a set of examples and B a ground background theory. We define the two Datalog programs P and P^* as follows:*

$$P = \bigcup_{0 \leq i \leq n} P_i \qquad \textit{and} \qquad P^* = \bigcup_{0 \leq i \leq n} P_i^* \tag{4}$$

where

- $P_0 = P_0^* = B$,
- $P_{i+1} = \{p \leftarrow I_i \mid p \in O_i \setminus I_i\}$ *for* $0 \leq i \leq n-1$, *and*
- $P_{i+1}^* = \{p \leftarrow (I_i \setminus M_i) \mid p \in O_i \setminus (I_i \cup M_i)\}$ *for* $0 \leq i \leq n-1$, *where M_i is the least model of* $\bigcup_{0 \leq j \leq i} P_j^*$.

Proposition 2. *Let E, B, P and P^* be as the ones in Definition 2. If E is coherent and it is consistent w.r.t. B then O_i is the least model of $P \cup I_i$ and $P^* \cup I_i$ for every i $(0 \leq i \leq n-1)$.*

In this process of computing such a rules, some simplifying techniques can be used to refine these rules, *e.g.*, removing being subsumed rules and removing redundant atoms from rule bodies [13].

The next proposition shows that we can safely replace the input background theory B with *ground*(B) in the algorithm ***LFIO***.

Proposition 3. *Let E a coherent set of examples, B be a background theory such that E is consistent w.r.t. B, $P =$ **LFIO**(B, E) and $P' =$ **LFIO**$($ground$(B), E)$. Then ground$(P) =$ ground(P').*

Now we can show the soundness of the inductive learning algorithm ***LFIO***.

Theorem 1. *Algorithm 1 is sound, viz, if E is coherent and it is consistent w.r.t. B then O is the least model of $B \cup$ **LFIO**$(B, E) \cup I$ for every $\langle I, O\rangle \in E$.*

Algorithm 1: ***LFIO***(B, E)

Input: E: a set of observations, B: a background theory
Output: A Datalog program P

```
1  P ← ∅;
2  foreach ⟨I, O⟩ ∈ E do
3      M ← the least model of B ∪ P;
4      foreach p ∈ O \ (M ∪ I) do
5          Let r be the rule p ← (I \ M);
6          if ∄r′ ∈ B ∪ P such that r′ ⪯ r then
7              Remove each rule r″ with r ⪯ r″ from P;
8              P ← P ∪ {r};
9          end
10     end
11 end
12 return P;
```

Proof. Let $P = B \cup \textbf{\textit{LFIO}}(B, E)$, $E = \{\langle I_j, O_j\rangle | 0 \leq j \leq n-1\}$. Since *ground*$(P \cup I_j) =$ *ground*$(P) \cup I_j$ for every j $(0 \leq j \leq n-1)$ by Lemma 1, we can assume that B is ground by Proposition 3, so that *ground*$(P \cup I) = P \cup I$ for each $\langle I, O\rangle \in E$. By Corollary 1, removing subsumed rules preserves the least model, this implies that we can further assume the condition of the algorithm ***LFIO*** (line-6) is tautology, namely, every such rule at line-5 is always added into P. In this case, $B \cup P$ is exactly the logic program P^* in Proposition 2. It follows that O is the least model of $P \cup I$ for each $\langle I, O\rangle \in E$ by Proposition 2.

3.3 Properties of the Algorithm

In the following we investigate the properties of the proposed inductive learning algorithm. On the one hand, this algorithm is modular and can be incrementally implemented; on the other hand, the algorithm can also be simplified.

The next proposition shows that the algorithm is modular.

Proposition 4. *Let $E = E_1 \cup E_2$ be a coherent set of examples which is consistent w.r.t. a background theory B. Let $P_1 =$ **LFIO**(B, E_1) and $P_2 =$ **LFIO**(B, E_2). Then O is the least model of $B \cup P_1 \cup P_2 \cup I$ for every $\langle I, O\rangle \in E$.*

The following proposition shows that the inductive learning algorithm can be incremental.

Proposition 5. *Let $E = E_1 \cup E_2$ be a coherent set of examples which is consistent w.r.t. a background theory B, and $E_1 \cap E_2 = \emptyset$. Let $P_1 =$ **LFIO**(B, E_1) and $P_2 =$ **LFIO**$(B \cup P_1, E_2)$. Then O is the least model of $B \cup P_2 \cup I$ for every $\langle I, O\rangle \in E$.*

As a matter of fact, the input examples of E in the algorithm ***LFIO*** can be somehow simplified. Firstly, in the case $I = O$ for some $\langle I, O\rangle \in E$, no rule is added in terms of the algorithm ***LFIO***. This kind of example can be safely discarded. Formally, we denote $\widetilde{E} = \{\langle I, O\rangle \in E \mid I \neq O\}$.

Furthermore, Two examples $\langle I_1, O_1\rangle$ and $\langle I_2, O_2\rangle$ are *equivalent* if $O_1 \setminus I_1 = O_2 \setminus I_2$. For a set E of examples and an example $\langle I, O\rangle \in E$, the *equivalent class* of $\langle I, O\rangle$ is the set $[\langle I, O\rangle]_E = \{\langle I', O'\rangle \in E \mid O' \setminus I' = O \setminus I\}$, we denote by $[\langle I, O\rangle]_E^m$ the set of examples in the equivalent class $[\langle I, O\rangle]_E$ such that I is minimal among the set $\{I' \mid \langle I', O'\rangle \in [\langle I, O\rangle]_E\}$; and by $[E]^m$ we denote the set $\bigcup_{\langle I,O\rangle \in E}[\langle I, O\rangle]_E^m$. For instance, for the set E of examples in the last paragraph, we have $[E]^m = \{\langle \emptyset, \{p\}\rangle\}$.

Proposition 6. *Let E be a coherent set of examples which is consistent w.r.t. a background theory B, $P_1 = \textbf{\textit{LFIO}}(B, E)$ and $P_2 = \textbf{\textit{LFIO}}(B, [\widetilde{E}]^m)$. Then $B \cup P_1 \cup I$ and $B \cup P_2 \cup I$ have the same least model O for each example $\langle I, O\rangle$ in E.*

4 Related Work

As far as we know, the existing inductive learning paradigms do not consider both input and output in observations [2, 3]. Among the well-known learning settings, including learning from entailment [6], learning from interpretation [5] and learning from proofs [7], the closest one to our learning setting is the paradigm of learning from interpretation transition [16].

In the paradigm of learning from interpretation transition, an example is a pair $\langle I, J\rangle$ where I and J are interpretations (an interpretation is a set of propositional atoms). This learning task is, for a given set E of examples, to find a normal logic program P such that $T_P(I) = J$ for every $\langle I, J\rangle$ in E. Though, the operator T_P [17] is defined for normal logic program, which is more general than Datalog programs, our learning task requires the output of P is J if the given input is I, *i.e.*, J is the least model of $P \cup I$. In other words, the semantics of Datalog programs considered in the paper is the least model, instead of the one-step interpretation transition T_P in [16].

In addition, the inductive learning task in the paper is also different from FOIL [18], which learns Horn clauses from data expressed as relations. Unlike FOIL, our learning task has no target predicates, and examples are not restricted to target predicate.

The form of examples in our learning setting is also different from the one in statistic learning [7], for which training data consist of input-output pairs (x, y) where y indicates the classification or regression for the instance x.

5 Concluding Remarks

In this paper we proposed a new inductive learning framework for which an example is a pair of input and output. The output is the least (Herbrand) model of a solution to an inductive learning task together with its background theory and input. A modular and incremental inductive learning algorithm was presented for this inductive learning task.

Acknowledgement. We thank reviewers for their helpful comments. This work is partially supported by NSFC under grant 63170161, Stadholder Fund of Guizhou Province under grant (2012)62, Outstanding Young Talent Training Fund of Guizhou Province under grant (2015)01 and Science and Technology Fund of Guizhou Province under grant [2014]7640.

References

1. Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.
2. Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19 / 20:629–679, 1994.
3. Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20 - biography and future challenges. *Machine Learning*, 86(1):3–23, 2012.
4. Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The automation of science. *Science*, 324(5923):85–89, 2009.
5. Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.
6. Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Min. Knowl. Discov.*, 3(1):59–93, 1999.
7. Andrea Passerini, Paolo Frasconi, and Luc De Raedt. Kernels on prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research*, 7:307–342, 2006.
8. Ehud Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, USA, 1983.
9. R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. A new proposal for debugging datalog programs. *Electronic Notes in Theoretical Computer Science*, 216:79 – 92, 2008. Proceedings of the 16th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2007).
10. Tingting Li, Marina De Vos, Julian Padget, Ken Satoh, and Tina Balke. Debugging ASP using ILP. In *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015), Cork, Ireland, August 31 - September 4, 2015.*, 2015.
11. Abiteboul Serge, Hull Richard, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
12. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
13. Y. Sagiv. Foundations of deductive databases and logic programming. chapter Optimizing Datalog Programs, pages 659–698. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
14. Thomas Eiter and Michael Fink. Uniform equivalence of logic programs under the stable model semantics. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2003.
15. Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *proceedings of The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 695–700. AAAI Press / The MIT Press, 2005.
16. Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.
17. Krzysztof R Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming*, pages 293–322. Morgan Kaufmann, Los Altos, 1988.
18. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.