A LOGIC PROGRAMMING FRAMEWORK FOR LEARNING BY IMITATION

Grazia Bombini, Nicola Di Mauro, Teresa M.A. Basile, Stefano Ferilli and Floriana Esposito
Dipartimento di Informatica, Università degli Studi di Bari, via Orabana 4, Bari, Italy

Keywords: Learning by imitation, Agents.

Abstract: Humans use imitation as a mechanism for acquiring knowledge, i.e. they use instructions and/or demonstrations provided by other humans. In this paper we propose a logic programming framework for learning from imitation in order to make an agent able to learn from relational demonstrations. In particular, demonstrations are received in incremental way and used as training examples while the agent interacts in a stochastic environment. This logical framework allows to represent domain specific knowledge as well as to compactly and declaratively represent complex relational processes. The framework has been implemented and validated with experiments in simulated agent domains.

1 INTRODUCTION

Learning from demonstration or *learning by imitation* represents a key research topic in robotics (Billard and Siegwart, 2004; Schaal, 1999; Schaal et al., 2003), being a promising approach, based on human-robot interaction, towards effective robot programming. Indeed, the need of robots that can learn in a human environment motivates recent research to investigate forms of social learning, such as imitation-based learning (Schaal, 1999) and learning by demonstration (Nicolescu and Mataric, 2003), inspired by the way people learn. By means of this approach, a robot should learn to imitate a teacher by observing demonstrations. Recent research in other fields considers imitative learning as an essential part of human development (Meltzoff, 2007). Humans and animals use imitation as a mechanism for acquiring knowledge. This method of learning has been shown (Chernova and Veloso, 2007) to reduce learning time compared to classical exploration-based methods such as reinforcement learning (Smart and Kaelbling, 2002).

Learning from demonstration is an approach that enables robots (an unskilled agent or the *observer*) to learn tasks by simply observing performances of a skilled agent (the *teacher*) (Atkeson and Schaal, 1997; Smart and Kaelbling, 2002). The robot gathers information about the task in the form of perceptual inputs and action outputs and estimates the latent control policy of the demonstrator. The estimated policy can then be used to drive the robots's autonomous behavior.

This paper aims at providing a framework that allows an agent to learn and revise in an incremental way a high level representation of a task by imitating a skilled agent.

Learning from demonstration is strongly related to supervised learning, in which the goal is to learn a policy given a fixed set of labeled data (Bentivegna et al., 2004). In this perspective it is possible to collect the interaction performed by teaching in the real world and to use it as examples for the learning phase. Such examples represent the best action to be taken in that context. Additionally, data are gathered incrementally, thus minimizing the number of labeled data required to learn the given policy.

In this paper we present an incremental policy learning approach based on a relational language used to describe both the demonstration examples and the learnt policy. The agent actively interacts with the human by deciding the next action to execute and requesting demonstration from the expert based on the currently learned policy.

2 LOGICAL BACKGROUND

We used Datalog (Ullman, 1988) as a representation language for the domain and induced knowledge, that here is briefly reviewed. For a more comprehensive introduction to Logic Programming and Inductive Logic Programming (ILP) we refer the reader

Bombini G., Di Mauro N., M. A. Basile T., Ferilli S. and Esposito F. (2009).
A LOGIC PROGRAMMING FRAMEWORK FOR LEARNING BY IMITATION.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Artificial Intelligence and Decision Support Systems*, pages 218-223
DOI: 10.5220/0002007502180223

to (Bratko, 2001; Muggleton and De Raedt, 1994; Lavrac and Dzeroski, 1994).

A first-order *alphabet* consists of a set of *constants*, a set of *variables*, a set of *function symbols*, and a non-empty set of *predicate symbols*. Each function symbol and each predicate symbol is associated to a natural number (its *arity*). The arity represents the number of arguments the function has. A *term* is a constant symbol, a variable symbol, or an n-ary function symbol f applied to n terms $t_1, t_2, \ldots, t_n$. An atom (or atomic formula) $p(t_1, \ldots, t_n)$ is a predicate symbol p of arity n applied to n terms t_i. An atomic formula l, or its negation $\bar{l}$, is said to be a (respectively, positive or negative) *literal*.

A *clause* is a formula of the form $\forall X_1 \forall X_2 \ldots \forall X_n (L_1 \vee L_2 \vee \ldots \vee \bar{L}_i \vee \bar{L}_{i+1} \vee \ldots \vee \bar{L}_m)$ where each L_i is an atom and $X_1, X_2, \ldots X_n$ are all the variables occurring in $L_1 \vee L_2 \vee \ldots \bar{L}_i \vee \ldots \bar{L}_m$. Most commonly the same clause is written as an implication $L_1, L_2, \ldots L_{i-1} \leftarrow L_i, L_{i+1}, \ldots L_m$, where $L_1, L_2, \ldots L_{i-1}$ is the *head* of the clause and $L_i, L_{i+1}, \ldots L_m$ is the *body* of the clause.

Clauses, literals and terms are said to be *ground* whenever they do not contain variables. A *Datalog clause* is a clause with no function symbols of non-zero arity; only variables and constants can be used as predicate arguments.

A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \ldots, X_n \leftarrow a_n\}$ where $X_i, 1 \leq i \leq n$ are variables and $a_i, 1 \leq i \leq n$ are terms. A substitution θ is applicable to an expression e, obtaining the expression $e\theta$, by replacing all variables X_i with their corresponding terms a_i.

3 LEARNING FROM DEMONSTRATION

Here, we assume that the environment is defined by a finite set of states S. For each state $s \in S$, the agent has available a finite set of actions $A(s) \subseteq \mathcal{A}$ which cause stochastic state transition, where $\mathcal{A}$ is the set of all the primitive actions. In particular, an action $a \in A(s)$ causes a transition to state s'_a when executed in state s.

The agent is assumed to observe a demonstrator that performs the correct sequence of actions useful to reach a given goal by starting from an initial state of the environment. During each training sequence, the agent records the observation about the environment and the corresponding action performed by the demonstrator. An observation $o \in S$ is represented by a set of ground Datalog literals.

Example 1. *The following set of literals represents an observation in a blocks world consisting of* 4 *blocks (a, b, c and d), where blocks can be on the floor or can be stacked on each other. Literal* $on(X,Y)$ *denotes that block X is on block Y, and that X and Y belong to the same stack.* { *clear(a), on(a, c), on(c, d), on(d, f), clear(b), on(b, f), block(a), block(b), block(c), block(d), floor(f)* }.

In this domain, the available actions that can be performed by the demonstrator are of the kind $move(X,Y)$, *with* $X \in \{a,b,c,d\}$ *and* $Y \in \{a,b,c,d,f\}$, $X \neq Y$.

Each training example, $e = \{a, o\}$, consists of an action $a \in \mathcal{A}$ selected by the demonstrator and an observation $o \in S$. Obviously, we assume that the demonstrator uses a good policy π to achieve the goal. Hence, the aim of the agent is to learn this hidden policy $\pi : S \rightarrow \mathcal{A}$ mapping states (observations) to actions.

Classical supervised learning is based on an inductive learning method able to generalize from positive and negative examples labeled by the user. In the case of imitative learning, the action taken by the teacher agent may be considered as a positive example and all other possible actions as negative examples. In particular, given a state s, if the teacher takes action $a_i \in A(s)$, then the observer can assume that a_i is a positive example and all other actions $a_j \in A(s)$ $1 \leq j \neq i \leq |A(s)|$ are negative ones. A negative example a_j is considered reliable until the demonstrator performs a_j in state s. The general process must be slightly modified in order to retract this kind of action.

Furthermore, the process of imitative learning is naturally modeled by an agent able to modify its theory in an incremental way, where each new incoming example may give rise to a theory revision process.

We represent the policy as a set of logical clauses where the *head* of the clause represent the action while the *body* represents the state. In particular, a clause represents an action that may be performed in a given state.

Example 2 . *In the blocks world domain a learned rule may be the following:* {*move(A, B) :- goal_on(D, E), clear(A), on(A, F), on(D, B), block(D), block(E), block(A), block(F), floor(B), not(on(F,B))*}

Moving block A on the floor is a good choice if block A is on block F, block F is not on the floor, block D is on the floor, and the goal is to put block D on block E.

Algorithm 1. Tuning(E,T,M).

Input: E: example; T: theory; M: historical memory;
1: Add E to M
2: **if** E is a positive example not covered from T **then**
3: generalize(T, E, M)
4: **else**
5: **if** E is a negative example covered by T **then**
6: specialize(T, E, M)

4 RELATIONAL INCREMENTAL LEARNING

4.1 INTHELEX

INTHELEX (INcremental THeory Learner from EXample) (Esposito et al., 2004) is the learning system for the induction of first-order logic theories from positive and negative examples exploited in this paper. It learns theories in form of sets of Datalog clauses, interpreted according the Object Identity (OI for short) (Semeraro et al., 1996) assumption, according to which, within a clause, terms (even variables) denoted with different symbols must be distinct. It can learn simultaneously several concepts, possibly related to each other. It uses a full memory storage strategy, and therefore retains all the available examples in a *historical memory*. A set of examples of the concepts to be learned is incrementally provided by an expert. Whenever a new example is taken into account, it is also stored in the historical memory.

INTHELEX is fully and inherently incremental. The learning phase can start by taking in input a previously generated version of the theory or from the first available example and an empty theory. The training examples are exploited by the system to modify incorrect hypotheses according to a data-driven strategy. When the theory is not correct with respect to an example, it is rejected and a process of theory revision starts. Such a process is based on two inductive refinement operator, one for generalising definitions that reject positive examples (completeness), and the other for specialising definitions that explain negative examples (consistency).

Algorithm 1 reports the procedure used in INTHELEX for refining a theory. M represents the set of all positive and negative examples already processed, E is the current example to be examined, and T is the theory learned from the examples in M.

4.2 INTHELEX$_I$

As already pointed out, INTHELEX works by taking into account the examples provided in incremental way by the expert. In the scenario of an agent acting in a stochastic world, the *correct actions* of the agent may be considered as positive examples in a supervised learning task.

In our framework we assume that an agent A aims at learning to act in a world by imitating an *expert agent* E. Hence, each action taken from E in a given state may be considered as a positive example for A. All other possible actions in the same state should be considered as negative examples. In this way A assumes that E acts in a correct way. Given an action-state pair (a,s) and a set $A(s)$ of possible actions that can be taken in state s, all other actions in $A(s) \setminus a$ is assumed to be negative examples while the expert agent does not take any of them in the same state s. When it happens, it is necessary to retract the previous hypothesized negative example.

Given a goal, each training example $e = \{a,o\}$ belongs to a specific learning class based on action a. For each class a theory must be learned in order to be able to predict the corresponding action to be taken on unseen observations.

The algorithm starts with an empty theory and an empty historical memory. The agent observes the sequences of actions performed by the demonstrator. For each timestep the agent tries to classify the observation (i.e., to predict the corresponding action) by using its learned model. The set N of all actions that are allowed in the domain and that the agent can perform in a state s, but are not performed by the demonstrator, are supposed to be negative examples for the class a (the correct action) in the state s.

The classification task returns an action c that is compared to the correct action a the demonstrator performs. When c does not correspond to the action a a theory revision is needed. All negative examples for class a with the same body as o are expunged from the historical memory. A generalization process of the current theory against example e, according to filtered historical memory, starts. At the end of the learning process, the learned theory T represents the optimal policy.

5 EXPERIMENTAL RESULTS

The proposed learning framework has been applied to two domains generally used in the field of agent learning.

Algorithm 2. IIL.

```
1: errors ← 0
2: T ← ∅
3: M ← ∅
4: loop
5:    (o,a) ← get an observation-action pair
6:    N ← {(o,¬a_i)|a_i ∈ A(o) \ {a}}
7:    c ← classify(o,T)
8:    if c ≠ a then
9:       errors++
10:      for all e ∈ M do
11:         if e is negative for the class a with the
            same body as o then
12:            M ← M \ {e}
13:      generalize( T, a ← o, M)
14:   M ← M ∪ {a ← o}
15:   for all a_i ∈ N do
16:      if M does not contain the positive example
         a_i ← o then
17:         if c = a_i then
18:            errors++
19:            specialize( T, ¬a_i ← o, M)
20:         M ← M ∪ {a ← o}
```

5.1 The Predator Prey Environment

The first experiment regards the problem of learning a policy in a domain where a predator should capture a prey. This stochastic environment consists of a 4x4 grid surrounded by a wall, with a predator and a prey inside. The predator catches the prey if the prey comes on the same square as the predator at the end of its move. The prey moves with random actions, while the predator follows a *good user-defined* strategy in order to capture the prey. Both the predator and the prey can move in four directions: north, east, south and west. The action of an agent consists in moving to the cell immediately adjacent in the selected direction. In case the target cell is a wall, the agent remains in the same cell. The two agents move in alternate turns.

An observation is made up of the agent's perception about the cells surrounding it in the four directions and the cell it occupies. The state of each cell may be *empty*, *wall* or *agent*. Starting from an initial state, once captured the prey the sequence of observations does not restart by placing agents in random positions, but it continues from the positions of catch.

For example, Figure 1 reports a predator agent having a wall to the west and the prey to the east. This observation is represented by the following set of literals: {*north(a, e), south(a, e), east(a, p), west(a, w), under(a, e)*} where *a* stays for the predator agent, *p* for the prey agent, *e* for empty, and *w* for wall. In this example, the positive action taken from the agent is *move_north(a)*, while *not(move_east(a))* and *not(move_south(a))* and *not(move_west(a))* are wrong actions.

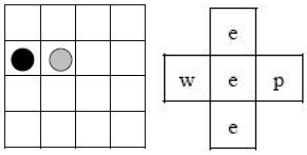

Figure 1: A sample predator-prey domain. The black circle represents the predator and the gray circle represents the prey. The figure on the right represents the predator agent percept.

Once fixed the strategy to capture the prey, we simulated a scenario in which a predator instructs another agent to capture the prey in a minimum number of a steps. Hence, given an observation, the action taken from the predator represents a positive training example, while all other possible actions are supposed to be negative examples.

We generated 10 sequences of observations. Each sequence, containing the traces of prey's captures, is made up of 322.5 positive and 967.5 negative observation-action pairs on average. Starting from a positive instance each alternative action has been hypothesised to be a negative instance.

On all of the 10 sequences the system learned a theory made up of 9.4 clauses, obtained by 14.8 generalizations and 2.2 specializations (17 errors) on average. It is worth noting that, in this domain, 1303 examples are sufficient on average for the system to learn a good policy. All further examples do not affect the learned policy.

In order to evaluate the behavior of the learning process, we generated a sequence made up of 1568 observation-action pairs. Figure 2 reports the number of errors (i.e., a generalization or a specialization request) the agent made during the learning phase. It can be noticed that the number of error grows until the system learns the correct policy (i.e., the learned classification theory). Figure 3 reports the predictive accuracy of the policy learnt by the agent during the imitation process on the complete historical memory made up of the complete set of 1568 positive observation-action pairs.

5.2 The Blocks World Environment

This domain consists of a surface (floor) on which there are four blocks. Blocks may be on the floor or one on top of another in a stack. To describe the world we use relations such as *on(a, b)*, i.e. block *a* is on

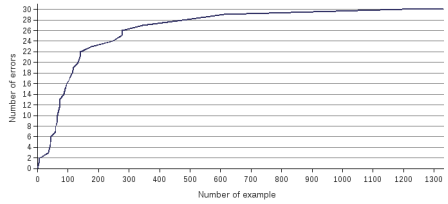

Figure 2: Errors during the learning phase.

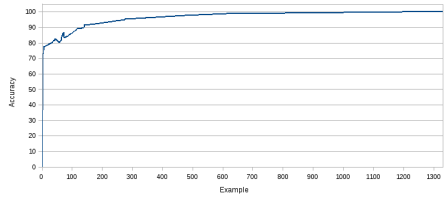

Figure 3: Prediction accuracy (%) over the entire historical memory at each revision theory.

block *b*, and *clear(a)*, i.e. block *a* is clear. Only clear blocks can be moved. An agent can move a block at a time, on another block or on the floor. We used the literal *move(a, b)* to describe the action of moving block *a* on block *b*.

For example, the configuration reported in Figure 4 is represented by the following set of literals $\{on(c, f), clear(d), on(d, f), clear(a), on(a, f), clear(b), on(b, c)\}$ where *f* stas for floor and *a, b, c, d* are blocks.

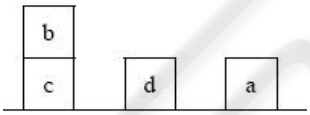

Figure 4: A blocks world example.

The tasks taken into account are: *stack*, achieved when all blocks are on one stack; *unstack*, achieved when all blocks are on the floor; and *on(a,b)*, reached when block *a* is on block *b*. We simulated the behavior of an agent able to select effective actions to achieve a goal (stack, unstack and on(a,b)). Each observation along with the corresponding action represents a positive example; all the other actions of moving other blocks are considered as negative examples. For instance, considering the scenario reported in Figure 4, if the goal is *goal_on(a, b)*, the correct action is *move(a, b)* and the wrong actions are *move(a, d), move(b, f), move(b, d), move(b, a), move(d, a), move(d, b)*.

We have generated 10 sequences of observations. Each sequence is made up on average of 382.1 positive observation-action pairs, and 2055.2 negative observation-action pairs generated starting from the positive action.

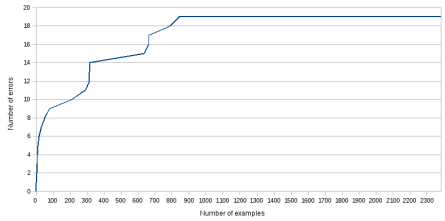

Figure 5: Errors during the learning phase.

Figure 5 reports the number of errors the agent made during the learning phase on a sequence of 2379 observation-action pairs.

On all of the 10 sequences the system learned a theory made up of 5.2 clauses, obtained by 5.8 generalization and 3.1 specialization (8.2 errors) on average. It is worth noting that, in this domain, 821 examples on average are sufficient for the system to learn a good policy.

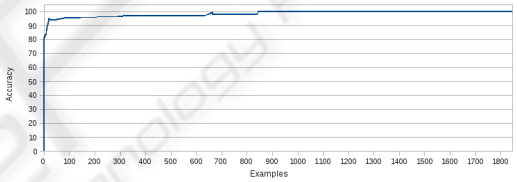

Figure 6: Prediction accuracy (%) over the entire historical memory at each revision theory.

Figure 6 reports the predictive accuracy of the agent on the historical memory at the and of the learning process.

6 CONCLUSIONS AND RELATED WORKS

An agent can learn using real examples of agent interaction with the world. In (Jebara and Pentland, 2002) the distribution of the environment is used to predict an agent's behavior. (Verma and Rao, 2007) proposes a framework for imitative learning that uses a probabilistic Graphical Model to describe an imitation process. A Graphical model is used to model the behavior of both the agent and the observed teacher.

In (Jansen and Belpaeme, 2006) the authors present a model for finding the intended goal of a demonstration using iterative interactions. They infer the goal of a demonstration without imitating the steps on how to reach the goal, but based on some psychological observations reported in (Wohlschlager et al., 2003). The same psychological observations

are taken into account in (Billard et al., 2004), in which agents learn new goals and how to achieve them.

In (Chernova and Veloso, 2007) a demonstration-based learning algorithm (*confident execution framework*) is used to train an agent. Such a framework allows an agent to learn a policy from demonstration. In the learning process, the agent observes the execution of an action. An agent is provided with a decision-making mechanism that allows it to actively choose whether observing or acting, with a gradually increasing autonomy. To learn a policy a supervised learning approach is used and the training data are acquired from the demonstration. All these approaches still do not use a relational representation formalism able to generalize the learned policies.

In this paper we have presented a logic framework that allows quickly, incrementally and accurately to train an agent to imitate a demonstrator of the task.

REFERENCES

Atkeson, C. and Schaal, S. (1997). Robot learning from demonstration. In Fisher, D., editor, *Proceedings of the 14th International Conference on Machine Learning (ICML)*, pages 12–20.

Bentivegna, D., Atkeson, C., and Cheng, G. (2004). Learning from observation and practice using primitives. In *AAAI Fall Symposium Series, 'Symposium on Real-life Reinforcement Learning'*.

Billard, A., Epars, Y., Calinon, S., Cheng, G., and Schaal, S. (2004). Discovering Optimal Imitation Strategies. *robotics and autonomous systems, Special Issue: Robot Learning from Demonstration*, 47(2-3):69–77.

Billard, A. and Siegwart, R. (2004). Robot learning from demonstration. *Robotics and Autonomous Systems*, 47(2-3):65–67.

Bratko, I. (2001). *Prolog programming for artificial intelligence, 3rd ed.* Addison-Wesley Longman Publishing Co.

Chernova, S. and Veloso, M. (2007). Confidence-based policy learning from demonstration using gaussian mixture models. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA. ACM.

Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., and Di Mauro, N. (2004). Incremental learning and concept drift in inthelex. *Intelligent Data Analysis Journal, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, 8(3):213–237.

Jansen, B. and Belpaeme, T. (2006). A computational model of intention reading in imitation. *Robotics and Autonomous Systems*, 54(5):394–402.

Jebara, T. and Pentland, A. (2002). Statistical imitative learning from perceptual data. In *Proc. ICDL 02*, pages 191–196.

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, New York.

Meltzoff, A. N. (2007). The "like me" framework for recognizing and becoming an intentional agent. *Acta Psychologica*, 124(1):26–43.

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679.

Nicolescu, M. N. and Mataric, M. J. (2003). Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS03)*, pages 241–248. ACM.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions: Biological Sciences*, 358(1431):537–547.

Semeraro, G., Esposito, F., and Malerba, D. (1996). Ideal refinement of datalog programs. In Proietti, M., editor, *Logic Program Synthesis and Transformation*, volume 1048 of *LNCS*, pages 120–136. Springer.

Smart, W. and Kaelbling, L. (2002). Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3404–3410.

Ullman, J. (1988). *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press.

Verma, D. and Rao, R. P. N. (2007). Imitation learning using graphical models. In Kok, J. N., Koronacki, J., de Mántaras, R. L., Matwin, S., Mladenic, D., and Skowron, A., editors, *18th European Conference on Machine Learning*, volume 4701 of *LNCS*, pages 757–764. Springer.

Wohlschlager, A., Gattis, M., and Bekkering, H. (2003). Action generation and action perception in imitation: An instantiation of the ideomotor principle. *Philosophical Transaction of the Royal Society of London: Biological Sciences 358*, 1431:501–515.