Relational Learning by Imitation

Grazia Bombini, Nicola Di Mauro, Teresa M.A. Basile, Stefano Ferilli, and Floriana Esposito

Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy
{gbombini,ndm,basile,ferilli,esposito}@di.uniba.it

Abstract. Imitative learning can be considered an essential task of humans development. People use instructions and demonstrations provided by other human experts to acquire knowledge. In order to make an agent capable of learning through demonstrations, we propose a relational framework for learning by imitation. Demonstrations and domain specific knowledge are compactly represented by a logical language able to express complex relational processes. The agent interacts in a stochastic environment and incrementally receives demonstrations. It actively interacts with the human by deciding the next action to execute and requesting demonstration from the expert based on the current learned policy. The framework has been implemented and validated with experiments in simulated agent domains.

Key words: Relational Learning, Learning by Imitation, Agents

1 Introduction

Learning from demonstration or *learning by imitation* [1, 2], represents a promising approach in the area of human-robot interaction. Recent research in other fields define the imitative learning as an essential part of human development [3]. Indeed, humans and animals use imitation as a mechanism for acquiring knowledge. Imitation-based learning [4] and learning by demonstration [5] are exploited to enable an unskilled agent, the *observer*, to learn tasks by simply observing performances of a skilled agent, the *teacher* [6, 7]. This approach can be viewed as a collaborative learning based on the interaction between the human and the agent. The agent gathers information about the task in the form of perceptual inputs and action results and estimates the latent control policy of the demonstrator. The estimated policy can then be used to control the agent's autonomous behaviour. As reported in [8], this method can reduce the learning time when compared to classical exploration-based methods such as reinforcement learning (RL) [7].

The goal of this paper is to provide a relational framework allowing an agent to learn and revise in an incremental way a logical representation of a task by imitating a skilled agent. In particular, we propose 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 teacher by

deciding the next action to execute and requesting demonstration. Furthermore, due to the relational description language, it is very simple to use background knowledge about the domain.

Learning from demonstration is strongly related to supervised learning where the goal is to learn a policy given a fixed set of labelled data [9]. From this point of view it is possible to collect the interactions between the teacher and the observer and using them as learning examples. Furthermore, data should be gathered in an incremental way by minimising the number of required labelled data useful to learn the given policy.

When we adopt a Reinforcement Learning (RL) approach, an agent in a certain state receives a rewards (positive or negative) regarding the goodness of the taken action. The agent's goal is to learn a function (*policy*) that maps the state of the world to the action that maximise the overall expected reward. A stochastic environment may require a lot of states needing the agent to take a large number of actions before to learn a good policy. In [10] has been presented a RL framework in which an agent learns observing the actions of other agents (*Implicit Imitation*) without requiring no explicit teacher. This increases the speed of policy learning, but requiring the knowledge or the ability to infer explicit state rewards, that are not always known or easy to infer. In [11] has been investigated how RL can profit from data acquired by demonstrations in the task of pole balancing.

In [12] the authors presented a model to find 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 by using some psychological observations reported in [13]. The same psychological observations are taken into account in [14] where agents learn new goals and how to achieve the goals. In [15] the agent's behaviour is predicted using a probabilistic distribution of the environment. In [16] has been proposed a framework for imitative learning that uses a probabilistic graphical model to describes a imitative process. In [8] has been used a demonstration based learning algorithm (*confident execution framework*) allowing an agent to learn a policy from demonstrations on how to execute actions. A supervised learning approach is used to learn a policy represented by a Gaussian model. All these approaches still do not use a relational representation formalism able to generalise the learned policies.

2 Logical background

Here we briefly review the used representation language for the domain and induced knowledge. For a more comprehensive introduction to logic programming and Inductive Logic Programming (ILP) we refer the reader to [17].

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 has a natural number (its *arity*) assigned to it. The arity assigned to a function symbol represents the number of arguments the function has. Constants may be viewed as function symbols of arity 0. A

term is a constant symbol, a variable symbols, or an n-ary function symbol f applied to n terms $t_1, t_2, \ldots, t_n$.

An atom $p(t_1, \ldots, t_n)$ (or atomic formula) is a predicate symbol p of arity n applied to n terms t_i. Both l and its negation $\bar{l}$ are said to be *literals* (resp. positive and negative literal) whenever l is an atomic formula. A *clause* is a formula of the form $\forall X_1 \forall X_2 \ldots \forall X_n (L_1 \vee L_2 \vee \ldots \vee \overline{L}_i \vee \overline{L}_{i+1} \vee \ldots \vee \overline{L}_m)$ where each L_i is a literal and $X_1, X_2, \ldots X_n$ are all the variables occurring in $L_1 \vee L_2 \vee \ldots \overline{L}_i \vee \ldots \overline{L}_m$. 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 *Horn clause* is a clause which contains at most one positive literal. 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 $X_i, 1 \leq i \leq n$ is a variable and $a_i, 1 \leq i \leq n$ is a term. 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.

The learning problem for ILP can be formally defined:

Given: A finite set of clauses $\mathcal{B}$ (*background knowledge*) and sets of clauses E^+ and E^- (positive and negative *examples*).

Find: A theory Σ (a finite set of clauses), such that $\Sigma \cup \mathcal{B}$ is *correct* with respect to E^+ and E^-, i.e.: a) $\Sigma \cup \mathcal{B}$ is *complete* with respect to E^+: $\Sigma \cup \mathcal{B} \models E^+$; and, b) $\Sigma \cup \mathcal{B}$ is *consistent* with respect to E^-: $\Sigma \cup \mathcal{B} \not\models E^-$.

Given the formula $\Sigma \cup \mathcal{B} \models E^+$, deriving E^+ from $\Sigma \cup \mathcal{B}$ is *deduction*, and deriving Σ from $\mathcal{B}$ and E^+ is *induction*. In the simplest model, $\mathcal{B}$ is supposed to be empty and the deductive inference rule $\models$ corresponds to *θ-subsumption* between clauses. In particular, a clause c_1 *θ-subsumes* a clause c_2 if and only if there exists a substitution σ such that $c_1\sigma \subseteq c_2$. c_1 is a *generalization* of c_2 (and c_2 a *specialization* of c_1) under θ-subsumption. If c_1 θ-subsumes c_2 then $c_1 \models c_2$.

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. In particular, an action $a \in A(s)$ causes a transition to the state s'_a when executed in the state s, where $\mathcal{A}$ is the set of all the primitive actions.

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 of the pursuit domain consisting of two agent, prey and predator, moving in a 4x4 grid. The literal *observation(A, D, O)*, denotes that an agent A percepts the observation O in direction D.

{ *observation(a, d1, e), observation(a, d2, p), observation(a, d3, e), observation(a, d4, e), direction(d1, named1), direction(d2, named2), direction(d3, named3), direction(d4, named4), north(named1), south(named2), east(named3), west(named4), under(a, e), prey(p), predator(a), wall(w), empty(e)* } .

In this domain, the available actions that can be performed by the demonstrator are *move(X,Y)* , with $X \in \{p, a\}$ and $Y \in \{d1, d2, d3, d4\}$ (directions that an agent can take corresponding to the cardinal directions).

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 case of imitative learning, the action taken by the teacher agent may be considered as positive example and all the other possible actions as negative examples. In particular, given a state s, if the teacher takes the action $a_i \in A(s)$, then the observer can assume that a_i is a positive example and all the other actions $a_j \in A(s)\ 1 \leq j \neq i \leq |A|$ as negative ones. A negative example a_j is considerate reliable until the demonstrator performs a_j in a state s. 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 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 pursuit domain a learned rule may be the following:

{ *move(A, B) :- predator(A), direction(B, D), north(D), observation(A, B, E), wall(E), observation(A, F, G), direction(F, H), south(H), prey(G), observation(A, I, J), direction(I, K), east(K), empty(J), under(A, J)* }

In this example, moving towards direction north is a good choice for the predator. Indeed, there are more chances to cough the prey remaining in the same square.

4 Relational Incremental Learning

INTHELEX (INcremental THeory Learner from EXample) [18] is an incremental learning system for the induction of first-order logic theory from positive and negative examples. The approach used by this system may be defined a multistrategy one since the system realizes the combination of different forms of reasoning in the same symbolic paradigm. It learns theory in form of sets of

Datalog clauses, interpreted according the Object Identity (OI for short) [19] assumption. Under the OI assumption, within a clause, terms (even variables) denoted with different symbols must be distinct.

INTHELEX is a learning system fully and inherently incremental. The learning phase can start by taking in input a previously generated version of the theory or from an empty theory and from the first available example. When the theory is not correct compared with an example, this is rejected and a process of theory revision starts. It can learn simultaneously several concepts, possibly related to each other. It uses a full memory storage strategy, and therefore retains all the available example.

INTHELEX use two inductive refinement operator, one for generalising definitions that reject positive examples (completeness), and the other for specialising definitions that explain negative examples (consistency). In order to perform its task, the system exploits a previous theory (if any) and a *historical memory* of all the past (positive and negative) examples that led to the current theory.

Algorithm 1 reports the tuning procedure used in INTHELEX for refining the theory to be learned. 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. In particular, 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. The training examples are exploited by the system to modify incorrect hypotheses according to a data-driven strategy. In particular, when a positive example is not covered by the theory, a generalization process produces a revised theory obtained in one of the following way:

1. generalize one of the definitions pertaining to the theory that relate to the concept on the example, by removing some conditions. That ensures the revised theory covers the new example and is consistent with the all negative examples previous examined;
2. add a new definition to the theory to explain the example;
3. add the example as a positive exception.

When a negative example is covered, a specialization process outputs a revised theory by performing one of the following actions:

1. add some conditions to one of the definitions that explains the example. These conditions must be able to characterize all positive example already considered and exclude the current negative example;
2. add a negative condition to one of the definitions to differentiate the current negative example by positive examples previously considered;
3. add the negative example as an negative exception.

The system can be provided with a background knowledge (i.e. some partial concept definitions know to be correct about the domain and hence not modifiable in the same format of a theory rule).

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

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)

task. In our framework we assume that an agent A aims to learn how 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. All the 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 until the expert agent does not take any of them in the same state s. In this case 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 the action a. For each class a theory must be learned in order to be able to predict the corresponding action of unseen observations. As reported in algorithm 2, the procedure starts with an empty theory and an empty historical memory. The agent observes the sequence of actions performed by the demonstrator. For each time-step the agent tries to classify the observation (i.e., to predict the corresponding action) by using its learned model. All the actions that are allowed in the domain, but are not performed by the demonstrator are supposed to be negative examples for the class a (the correct action) in the state s. This set N is generated by taking into account all the possible actions that the agent can perform in s. The classification task returns an action c that is compared to the correct action a the demonstrator performs. When the action c does not correspond to the action a a theory revision is needed. All the negative examples for the class a with the same body of o are excluded from historical memory. A generalization process on the current theory with the example e and 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 a pursuit domain, generally used in the field of agent learning. The 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. We assume that the predator caught the prey if the prey lands on the same square as the predator at the end of its move. The prey moves

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}
```

with random actions, while the predator follows a *good user-defined* strategy in order to capture the prey. Both predator and prey can move in four directions, specifically north, east, south and west. The action of an agent involves moving in the square immediately adjacent corresponding to the selected direction. In case of the target square is a wall, then the agent remains in the same square. The two agents move alternate turns.

An observation is made up of the agent's perception about the squares surrounding it in the four directions and the square under it. The state of each square 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, a predator agent having a wall to the west and the prey to the east, the observation is represented by the following set of literals: { *observation(a, d1, e), observation(a, d2, e), observation(a, d3, p), observation(a, d4, w), under(a, e), direction(d1, named1), direction(d2, named2), direction(d3, named3), direction(d4, named4), north(named1), south(named2), east(named3), west(named4),* } where a stay for predator agent, p for prey agent, e for empty, and w for wall.

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

We have 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.

A first experiment has been performed on the whole dataset without providing the system with any background knowledge. Over all the 10 sequences the system learned a theory (policy) made up of 9,4 clauses, obtained by 14,8 generalisation and 2,2 specialisation (26,4 errors) on average.

In order to evaluate the behaviour of the learning process, we generated a sequence made up of 1332 observation-action pairs. The Figure 1 reports on the left, the number of errors (a generalisation or a specialisation request) the agent made during the learning phase. In particular, as we can see the number of errors grows until the system learns the correct policy (i.e., the learned classification theory). The Figure 1 on the right, reports the prediction accuracy of the policy, learnt by the agent during the imitation process, on the complete historical memory.

An example of learned rule is { *move(A, B) :- predator(A), direction(B, D), direction(F, H), observation(A, I, J), observation(A, B, E), observation(A, F, G), south(H), north(D), wall(E), prey(G), empty(J), under(A, J).* }

where a predator A moves toward north when at south there is the prey and at north there is a wall.

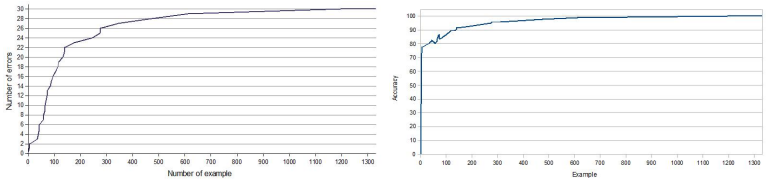

Fig. 1. Errors during the learning phase. The figure on the right represents the prediction accuracy (%) over the entire historical memory for each theory revision.

A new experiment has been performed with the system provided with a background knowledge containing several rules defining the concept of next direction. An example is { *next(A,B):- north(A), east(B).*}. This rule denotes that to north direction follows the east direction. The learned theory abstracts the concept of direction. An example of the learned rule is

{*move(A, B) :- predator(A), direction(B, D), direction(G, F), next(D, H), next(H, F), next(F, E), next(E, D), observation(A, B, I), observation(A, G, K), wall(I), under(A, J), empty(J), prey(K).* }

where the predator A moves in the direction D when it observes that in D there is a wall and opposite there is the prey K at any edge of the grid (D may be one of the for direction). This action applies to four different states.

Over all the same 10 sequences of the previous experiments, the system learned a theory made up of 6,4 clauses, obtained by 12,1 generalization and 3,5 specialization (22 errors) on average. In order to evaluate the behavior of

the learning process with the use of a background knowledge, we generated a sequence made up of 1276 observation-action pairs.

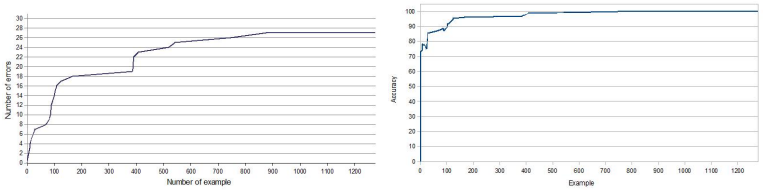

Fig. 2. Errors during the learning phase. The figure on the right represents the prediction accuracy (%) over the entire historical memory at each revision theory with the use background knowledge.

The Figure 2 reports the number of errors (a generalization or a specialization request) the agent made during the learning phase and prediction accuracy of the policy.

It is interesting to note that the theory learned with the use of background knowledge does not represent the actual directions, but the relationship between them. This allows an abstraction on states. Indeed, the policy learned results more compact (a lower number of clauses on average compared with the first experiment) and each clause is applicable to a subset of states.

6 Conclusion

Recently have been studied forms of social learning inspired by the ways people learn and their applications in robotics. In the field of agent and multi-agent systems most of the approaches used to learn does not allow the use of a high-level language to describe the domain and knowledge about it. In this paper we have presented a relational framework that allows to quickly, accurately and incrementally train an unskilled agent to imitate a demonstrator (human or artificial). The proposed technique has been implemented and tested on a dataset belonging to a pursuit domain generally used in the field of agent learning, and the results confirm the validity of the technique. As a future work, we are planning to extend the framework to a domain in which an agent can learn different skills by different teachers autonomously choosing skills and demonstrations.

References

1. Billard, A., Siegwart, R.: Robot learning from demonstration. Robotics and Autonomous Systems **47**(2-3) (2004) 65–67
2. Schaal, S., Ijspeert, A., Billard, A.: Computational approaches to motor learning by imitation. Philosophical Transactions: Biological Sciences **358**(1431) (2003) 537–547

3. Meltzoff, A.: The "like me" framework for recognizing and becoming an intentional agent. Acta Psychologica **124**(1) (2007) 26–43
4. Schaal, S.: Is imitation learning the route to humanoid robots? Trends in cognitive sciences **3**(6) (1999) 233–242
5. Nicolescu, M., Mataric, M.: 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), ACM (2003) 241–248
6. Atkeson, C., Schaal, S.: Robot learning from demonstration. In Fisher, D., ed.: Proceedings of the 14th International Conference on Machine Learning (ICML). (1997) 12–20
7. Smart, W., Kaelbling, L.: Effective reinforcement learning for mobile robots. In: IEEE International Conference on Robotics and Automation (ICRA). Volume 4. (2002) 3404–3410
8. Chernova, S., Veloso, M.: 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, New York, NY, USA, ACM (2007) 1–8
9. Bentivegna, D., Atkeson, C., Cheng, G.: Learning from observation and practice using primitives. In: AAAI Fall Symposium Series, 'Symposium on Real-life Reinforcement Learning'. (2004)
10. Price, B., Boutilier, C.: Accelerating reinforcement learning through implicit imitation. Journal of Artificial Intelligence Research **19** (2003) 2003
11. Schaal, S.: Learning from demonstration. Advances in Neural Information Processing Systems **9** (1997) 1040–1046
12. Jansen, B., Belpaeme, T.: A computational model of intention reading in imitation. Robotics and Autonomous Systems **54**(5) (2006) 394–402
13. Wohlschlager, A., Gattis, M., Bekkering, H.: 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** (2003) 501–515
14. Billard, A., Epars, Y., Calinon, S., Cheng, G., Schaal, S.: Discovering Optimal Imitation Strategies. robotics and autonomous systems, Special Issue: Robot Learning from Demonstration **47**(2-3) (2004) 69–77
15. Jebara, T., Pentland, A.: Statistical imitative learning from perceptual data. In: Proc. ICDL 02. (2002) 191–196
16. Verma, D., Rao, R.: Imitation learning using graphical models. In Kok, J.N., Koronacki, J., de Mántaras, R.L., Matwin, S., Mladenic, D., Skowron, A., eds.: 18th European Conference on Machine Learning. Volume 4701 of LNCS., Springer (2007) 757–764
17. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
18. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., Di Mauro, N.: 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) (2004) 213–237
19. Semeraro, G., Esposito, F., Malerba, D.: Ideal refinement of datalog programs. In Proietti, M., ed.: Logic Program Synthesis and Transformation. Volume 1048 of LNCS., Springer (1996) 120–136