Towards neurosymbolic RL via inductive learning of answer set programs

Anonymous submission

Abstract

We present a novel approach that combines symbolic, logic-based reasoning with Reinforcement Learning (RL) to improve training outcome without compromising execution time. By integrating Inductive Logic Programming (ILP) with model-free RL, we learn human-interpretable policy heuristics that can bias the exploration process in RL algorithms. Utilizing Answer Set Programming (ASP) for logical representation of policy heuristics, and incremental ILP for efficient data stream management, we demonstrate the effectiveness of our approach on an approximate Q-learner for the Pac-Man scenario. Preliminary results show improved training outcome when offline learned specifications are used to bias the exploration phase. Moreover, the use of ASP heuristics does not significantly impact the training time, and incremental ILP paves the way towards a novel approach to neurosymbolic RL.

Introduction

The intersection of symbolic, logic-based reasoning and Reinforcement Learning (RL) to solve Markov Decision Processes (MDPs) represents a topic of interest in Artificial Intelligence (AI) research, aiming to harness the strengths of both domains to address their individual limitations. Current state-of-the-art approaches in model-free RL, in fact, predominantly rely on extensive data or pre-defined environmental models, facing significant challenges in terms of efficiency, scalability, and the integration of existing knowledge. Furthermore, the decision process underlying policy generation is not transparent to other agents and humans, reducing safety and trustworthiness. On the other hand, symbolic AI works well in the small data regime, but is not suitable for nonsymbolic data and is not noise tolerant (Vermeulen, Manhaeve, and Marra 2023). Incorporating symbolic and logical formalisms into AI systems, as highlighted by (Kambhampati et al. 2022), can significantly boost their interpretability, thereby fostering wider acceptance and diffusion, while also playing a crucial role in refining policy computation. For instance, the REBA framework (Sridharan et al. 2019) successfully leverages Answer Set Programming (ASP) (Calimeri et al. 2020), to map out spatial relationships to direct robots in domestic settings. Moreover, De Giacomo et al. (2019) and Leonetti, Iocchi, and Patrizi (2012) employ linear temporal logic to steer MDP exploration, demonstrating the utility of logical formalisms in enhancing AI system performance. These methods assume logical specifications are available in advance. Recently, in the field of neurosymbolic learning and reasoning, Marra and Kuželka (2021); Hazra and De Raedt (2023) combined statistical relational learning with neural networks and RL, which however poses challenges as limited expressiveness and poor scalability (Acharya et al. 2023).

Inspired by results obtained by Mazzi et al. (2023) and Meli, Castellini, and Farinelli (2024) in model-based RL, we present the first steps of a novel approach, whose final aim is to exploit Inductive Logic Programming (ILP, Muggleton (1991)) to learn *human-interpretable* policy heuristics during the execution of RL algorithms, and directly use them online to bias the agent in its exploration process. Specifically, starting from execution traces of a trained model-free RL agent, we first use ILP to discover logical specifications that map actions to a higher-level representation of the state space (features), without any need for user-provided knowledge. Feature definition enhances the interpretability of the trained policy. Moreover, with respect to approaches based on neural-networks (Lee, Cai, and Hsu 2021; Cai and Hsu 2022), the ILP solution requires significantly fewer examples. We then use the logical formalism of ASP (Lifschitz 1999) to represent logical specifications and employ them as online heuristics in the exploration phase of an approximate Q-learner. ASP is chosen as the state of the art in planning domain representation for autonomous agents (Meli, Nakawala, and Fiorini 2023). We also investigate the feasibility of integrated online planning and learning in model-free RL, adopting IncrementaLAS (Law, Broda, and Russo 2022) as an ILP system to efficiently handle incremental streams of data, and evaluating its computational impact and performance as more batches of RL experience are gathered.

We conduct preliminary tests in the Pac-Man scenario, a benchmark RL domain posing challenges as sparse rewards, very long planning horizon and the presence of contrastive agents (the ghosts chasing Pac-Man) (Rohlfshagen and Lucas 2011; Samothrakis, Robles, and Lucas 2011). After providing some essential background knowledge, in the upcoming sections we will first outline the methodology used to conduct preliminary tests on the feasibility of the proposed approach. This will be followed by a presentation of the results, and finally a reflection on future goals.

Background

We now introduce some basic notions about the approximate Q-Learning algorithm, ASP and ILP.

Approximate Q-Learning

Approximate Q-learning (Sutton 1995) addresses scalability issues inherent in traditional Q-learning, particularly in environments with large or continuous state spaces (Buşoniu et al. 2011). Given a MDP $\langle S, A, R, T, \gamma \rangle$, respectively denoting the state and action spaces, the reward and transition maps and the discount factor, function $Q(s, a; \theta)$ approximates the Q-value of state-action pairs with parameters θ, updated at each batch of experience episodes (i.e., observed states and rewards) via gradient descent. At each episode, the agent adopts an ϵ-greedy policy to balance between exploration and exploitation: with probability $1 - \epsilon$, the agent selects the action believed to have the highest Q-value (exploitation), and with probability ϵ, it selects a random action (exploration). This policy prevents premature convergence to suboptimal policies by ensuring adequate exploration of the action space.

Answer Set Programming

Answer Set Programming (ASP) defines a domain as a set of logical statements (axioms) articulating the logical relationships between entities represented as variables and predicates (atoms) (Calimeri et al. 2020). Axioms considered in this work are *normal rules* $\mathtt{h} : -\mathtt{b}_1, \dots, \mathtt{b}_\mathtt{n}$, which define the body of the rule (i.e. the logical conjunction of terms $\bigwedge_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{b}_\mathtt{i}$) as a precondition for the head $\mathtt{h}$. We say that a variable is grounded when assigned a particular value. Consequently, an atom is grounded when all its variables are grounded. Given an ASP program P, its Herbrand base $\mathcal{H}(P)$ defines the set of ground atoms which can be generated from it. From an ASP domain definition, an ASP solver computes the *answer sets*, i.e., the minimal sets of ground atoms that satisfy the axioms. In this work, we assume that body atoms represent environmental features describing S in the MDP, while head atoms are actions. Answer sets will then represent feasible actions for the RL agent.

Inductive Logic Programming

An ILP (Muggleton 1991) task is defined as a tuple $\mathcal{T} = \langle B, M, E \rangle$, consisting of background knowledge B expressed in a logic formalism F, a *mode declaration* defining how to generate all possible axioms that compose the search space S_M, and a set of examples E, both expressed in the syntax of F. The goal is to find a hypothesis (i.e. an axiom) $H \subseteq S_M$ covering E. Under the ASP semantics, examples are *weighted context dependent partial interpretations* (WCDPI's) (Law, Russo, and Broda 2018). A partial interpretation e is a pair of sets of ground atoms $\langle e^{inc}, e^{exc} \rangle$. An interpretation (i.e., a set of ground atoms) I extends e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A WCDPI is a tuple $e = \langle e^{id}, e^{pen}, e^{pi}, e^{ctx} \rangle$, where e^{id} is an identifier for e, e^{pen} is either a positive integer or ∞, called a penalty, e^{pi} is a partial interpretation, and e^{ctx} is an ASP program called a context. A WCDPI e is accepted by a program P iff there is an answer set of $P \cup e^{ctx}$ that extends e^{pi}. Following the definition by Law, Russo, and Broda (2018), the goal of ILP is then to find a hypothesis $H \subseteq S_M$ with minimal length (i.e., number of atoms) and $\sum_e e^{pen}$, for all examples e not accepted by $H \cup B$. Since we are interested in discovering normal rules matching actions to environmental features, e^{inc}, e^{exc} represent executed and not executed actions, respectively, while e^{ctx} is the set of ground environmental features. We employ IncrementaLAS learner by Law, Broda, and Russo (2022) for fast computation from an incremental list of examples, gathered from RL batches.

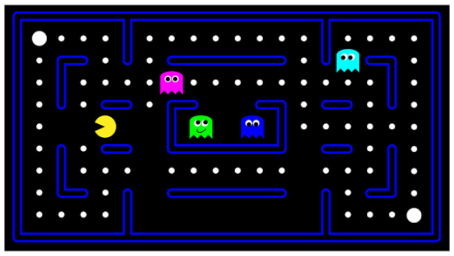
Figure 1: Example scenario for the Pac-Man domain

Methodology

This section describes our methodology to integrate approximate Q-learning and symbolic learning and reasoning in ASP. We exemplify it in the context of the standard RL Pac-Man domain used for empirical evaluation (Figure 1), where the Pac-Man agent (yellow) must collect food pellets (yellow dots) in a grid world, while avoiding walls and ghosts, with the possibility to collect a power pill (white dots) to chase and eat ghosts.

ASP representation of the domain

We start from the representation of the domain in ASP syntax. This requires the definition of ASP environment features $\mathcal{F}$ and actions $\mathcal{A}$. For the Pac-Man domain, $\mathcal{F}$ contains: `wall(Dir)`, which denotes the presence of a wall in front of the agent (i.e., one cell away) in that direction, `food(Dir, Dist)`, `ghost(Dir, Dist)`, and `capsule(Dir, Dist)`, representing Manhattan distance `Dist` $\in [0, 10]$ between PacMan and a cell containing food (or ghost, or capsule) in direction `Dir` $\in$ {`north, south, east, west`}. Finally, we need to introduce upper and lower bounds on atoms `Dist`. To this aim, we define atoms `X_dist_Y(Dir,Dist,D)`, where `X` is either `ghost, food` or `caps`, and `Y` is either `geq` or `leq`. These atoms state that item `X` has Manhattan distance `Dist` from Pac-Man along direction `Dir`, and `Dist` is greater (`geq`) or lower (`leq`) than `D`. Action atoms are constructed as `move(Dir)`, denoting the movement of the agent in direction `Dir`. The agent also has the option to perform a 'stop' action. However, we chose not to include it in the learning phase as it was rarely used in the examples collected during training. Once $\mathcal{F}$ and $\mathcal{A}$ are defined, we need a *feature map* $F_{\mathcal{F}} : S \rightarrow \mathcal{H}(\mathcal{F})$ and an *action map* $F_{\mathcal{A}} : A \rightarrow \mathcal{H}(\mathcal{A})$ to ground atoms from collected batches of RL. The only information needed to build $\mathcal{F}$ are the positions of the agent, food, ghosts and capsules in the environment, all of which are available in S.

Definition of the learning task

Once the ground atoms are obtained from execution traces, they serve to construct WCDPIs for IncrementaLAS. For each training batch, we extract the episodes in which the agent obtained the highest reward. We then use them to define, for each state-action pair $\langle \bar{a}_i, \bar{s}_i \rangle$, happening at time step $i \in 1 \ldots N$ (where N is the lenght of the episode from start to win/lose), a WCDPI of the following form:

$$e_i = \langle id_i, w_i, \langle \{\bar{a}_i\}, a \in A \setminus \{\bar{a}_i\} \rangle, F_{\mathcal{F}}(\bar{s}_i) \rangle,$$

where id_i is a unique identifier and w_i represents the penalty of the WCDPI. We assign the same weight to each step coming from the same run, normalising the reward obtained by the agent in that episode to a value $w \in [1, 100]$. This allows us to give more relevance to the behaviour of the agent that resulted in a higher reward, in order to generate rules that can lead to the same performance. We populate e^{inc} with the agent's chosen action and e^{exc} with the grounding for all unobserved actions. The background knowledge of the task only contains the definition of the ASP variables and ranges, and the mode bias is defined in order to only have rule's heads in the form `move(Dir)`, and bodies containing atoms from $\mathcal{F}$. Once the task is defined, we run IncrementaLAS once for each batch of training, incrementally adding WCDPIs to the task in order to refine rules as the training progresses.

Integration of extracted rules in RL

Once the rules are obtained from the execution of IncrementLAS, they can be used to guide the agent's exploration in subsequent training runs. We then modify the approximate Q-Learning algorithm so that it chooses, with probability ϵ, not a random action from among all the actions allowed in the environment, but one among the actions whose premises are satisfied (i.e. the actions that are represented as heads of rules whose bodies are entirely grounded). Algorithm 1 shows the modified version of the Approximate Q-Learning we implemented. Assuming that the **getBiasedActionsSet(s)** function calls the ASP program in which extracted rules have been integrated, the current state s is converted to ASP formalism via $F_{\mathcal{F}}(s)$, to adapt the answer set generation based on real-time information. The returned answer set, then, is the set of actions (after $F_{\mathcal{A}}^{-1}$ transformation) that can be accomplished based on the available rules. Actions not recommended by ASP are excluded from the list of available actions for the agent. From this set, the agent selects an action to perform with uniform probability. The variation of ϵ gives control over the influence of the rules in the algorithm. Moreover, while this is still not implemented in this work, rules can be iteratively updated at each batch of learning, due to the incremental nature of IncrementaLAS.

Preliminary Results

Preliminary results investigating the feasibility of the methodology outlined in the previous section are now presented. All experiments have been performed on a computer equipped with 5.1GHz e.g. 13th Gen Intel i5 processor and 64GB RAM.

Algorithm 1: Biased Approximate Q-Learning

Input: S, A, $\phi(s, a)$
Parameters: $\alpha \in (0, 1]$, $\gamma \in [0, 1]$, $\epsilon \in [0, 1]$
Output: $Q(s, a)$

```
1:  Initialize weight vector $\mathbf{w}$ with small random values
2:  while $\neg$stop do
3:      Observe initial state $s$
4:      while $s$ is not terminal do
5:          Generate a random number $rand$
6:          if $rand < \epsilon$ then
7:              a $\leftarrow$ rand(getBiasedActionsSet(s))
8:          else
9:              a $\leftarrow \arg\max_{a'} \mathbf{w}^\top \phi(s, a')$
10:         end if
11:         Execute $a$ and observe reward $r$ and new state $s'$
12:         $a^* \leftarrow \arg\max_{a'} \mathbf{w}^\top \phi(s', a')$
13:         $\delta \leftarrow r + \gamma \mathbf{w}^\top \phi(s', a^*) - \mathbf{w}^\top \phi(s, a)$
14:         $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \phi(s, a)$
15:         $s \leftarrow s'$
16:     end while
17: end while
18: return $\mathbf{w}$
```

From the benchmark domain, we executed the Approximate Q-Learning agent and collected 50,000 training episodes. These episodes were then divided into batches of 100 each and, for each batch, we saved the top five episodes with the highest reward to generate the WCDPIs for the learning task. We then used IncrementaLAS on the resulting tasks, gradually increasing the number of examples (we generated ≈ 650 WCDPIs from each batch) by adding those from the new batches. The search space S_M remains constant during the whole procedure and consists of 226 unique rules. Below are the rules we have learned, covering 84% of WCDPIs:

```
move(V0) :- food_dist_leq(V0,V1,4),          (1a)
            not wall(V0),
            ghost_dist_geq(V0,V2,4)
move(V0) :- food_dist_leq(V0,V2,1).          (1b)
move(V0) :- caps_dist_leq(V0,V2,4).          (1c)
```

First rule allows movement if food is close, no wall is in the way, and ghosts are far enough. The second permits movement when food is very close, and the third when a capsule is nearby. Incrementally performing the learning procedure on the whole set of acquired examples, we were able to confirm that the acquired rules converge as RL progresses.

Figure 2 shows the convergence of the learned rules expressed as the Hamming distance between the atoms composing the body of rules extracted from subsequent batches. Since every batch consists of 100 episodes, it is immediate to notice that rules convergence is acquired at the same point in which return convergence is reached by the approximate Q-Learning algorithm, which is shown in Figure 3 (red

Table 1: Execution times (in seconds) for the RL algorithm with and without bias across 5 different seeds. Both for the whole training and for a single batch. The last row shows the average execution time for each method.

Seed	Approx Q-Learning Time (s)	Avg Time Per Batch (s)	Biased Approx Q-Learning Time (s)	Avg Time Per Batch (s)
0	3433.67	6.87	5345.87	10.7
1	4754.06	9.51	4175.16	8.35
2	2748.81	5.50	4124.22	8.25
3	2738.73	5.47	4143.45	8.28
4	4700.36	9.40	4154.31	8.30
Average	3674.26	7.35	4388.02	8.78

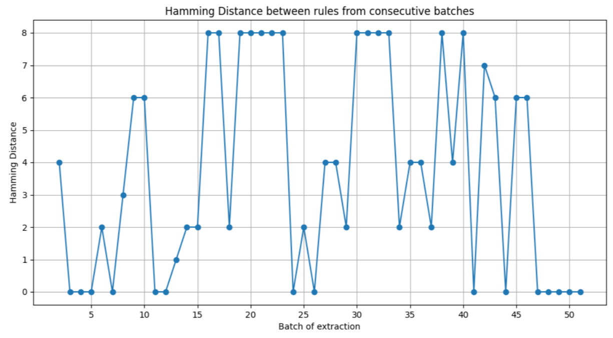

Figure 2: Convergence of learned rules. In order to facilitate representation, episodes beyond the 50th (with null Hamming distance) are omitted.

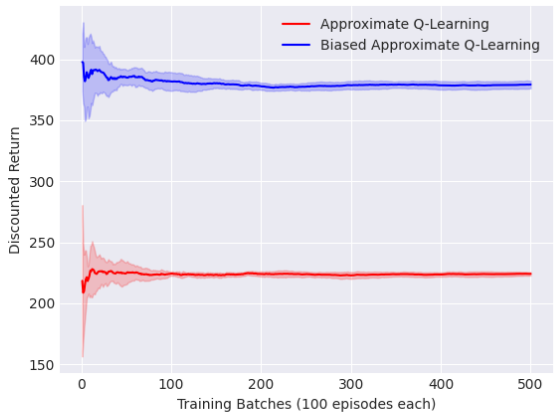

Figure 3: Average reward over a 50k episodes training performed with the approximate Q-Learning algorithm (red) and on the biased approximate Q-Learning version (blue), in which learned rules are employed. We show mean and standard deviation over 5 random seeds.

curve). The figure also shows the return obtained by the execution of our biased approximate Q-Learning algorithm 1, in which rules that were learned from the previous training were inserted to bias the exploration. In addition to the significantly higher reward, which is in part a consequence of the inserted rules reflecting the behavior of an already fully trained agent, this test allowed us to investigate the impact of calculating the grounded set of actions in terms of time. The training procedure was repeated for both agents using five different seeds. The execution times, shown in Table 1, demonstrate that the time overhead from adding the bias is negligible (less than 20% more) compared to the total training time.

For the future integration of ILP into the online RL pipeline, we also investigated the learning time of IncrementaLAS. On average, the rule extraction procedure took less that 2 seconds per batch, which is again not impacting if compared to the algorithm's execution time on a batch of examples. Moreover, this time can be significantly reduced by keeping the number of WCDPIs of the task constant instead of progressively increasing it. This can be accomplished by gradually substituting the gathered examples with the recently obtained ones, once they have attained a higher reward.

Conclusion

We proposed a novel approach that merges symbolic, logic-based reasoning with Reinforcement Learning (RL) to address the challenges of interpretability, scalability, and integration of existing knowledge within AI systems. Our method employs incremental Inductive Logic Programming (ILP) in the ASP semantics to extract human-interpretable policy heuristics from the behavior of a model-free RL agent, based on approximate Q-learning. This has the potential to significantly enhance the transparency and efficiency of the learning process, and the generalization of the learned policy. Our approach has achieved promising results in preliminary tests in the benchmark Pac-Man domain in terms of computational time and performance, laying a promising foundation for further research on neurosymbolic planning and learning. As future work, we plan to implement the online rule learning procedure in the RL pipeline, extend this methodology to other RL algorithms than Q-learning, and investigate the generalizability of the obtained policy heuristics in more complex scenarios.

References

Acharya, K.; Raza, W.; Dourado, C.; Velasquez, A.; and Song, H. H. 2023. Neurosymbolic reinforcement learning and planning: A survey. *IEEE Transactions on Artificial Intelligence*.

Buşoniu, L.; Ernst, D.; De Schutter, B.; and Babuška, R. 2011. Approximate reinforcement learning: An overview. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 1–8.

Cai, P.; and Hsu, D. 2022. Closing the Planning–Learning Loop With Application to Autonomous Driving. *IEEE Transactions on Robotics*, 39(2): 998–1011.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, 20(2): 294–309.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 128–136.

Hazra, R.; and De Raedt, L. 2023. Deep explainable relational reinforcement learning: a neuro-symbolic approach. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 213–229. Springer.

Kambhampati, S.; Sreedharan, S.; Verma, M.; Zha, Y.; and Guan, L. 2022. Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 12262–12267.

Law, M.; Broda, K.; and Russo, A. 2022. Search Space Expansion for Efficient Incremental Inductive Logic Programming from Streamed Data. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 2697–2704. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Law, M.; Russo, A.; and Broda, K. 2018. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*.

Lee, Y.; Cai, P.; and Hsu, D. 2021. MAGIC: Learning Macro-Actions for Online POMDP Planning. In *Proceedings of Robotics: Science and Systems*. Virtual.

Leonetti, M.; Iocchi, L.; and Patrizi, F. 2012. Automatic generation and learning of finite-state controllers. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, 135–144. Springer.

Lifschitz, V. 1999. Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 373–374. Springer.

Marra, G.; and Kuželka, O. 2021. Neural markov logic networks. In *Uncertainty in Artificial Intelligence*, 908–917. PMLR.

Mazzi, G.; Meli, D.; Castellini, A.; and Farinelli, A. 2023. Learning Logic Specifications for Soft Policy Guidance in POMCP. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 373–381.

Meli, D.; Castellini, A.; and Farinelli, A. 2024. Learning Logic Specifications for Policy Guidance in POMDPs: An Inductive Logic Programming Approach. *Journal of Artificial Intelligence Research*, 79: 725–776.

Meli, D.; Nakawala, H.; and Fiorini, P. 2023. Logic programming for deliberative robotic task planning. *Artificial Intelligence Review*, 56(9): 9011–9049.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing*, 8(4): 295–318.

Rohlfshagen, P.; and Lucas, S. 2011. Ms pac-man versus ghost team cec 2011 competition. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, 70–77.

Samothrakis, S.; Robles, D.; and Lucas, S. 2011. Fast approximate max-n monte carlo tree search for ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2): 142–154.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2019. REBA: A refinement-based architecture for knowledge representation and reasoning in robotics. *Journal of Artificial Intelligence Research*, 65: 87–180.

Sutton, R. S. 1995. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In Touretzky, D.; Mozer, M.; and Hasselmo, M., eds., *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

Vermeulen, A.; Manhaeve, R.; and Marra, G. 2023. An Experimental Overview of Neural-Symbolic Systems. In *International Conference on Inductive Logic Programming*.