THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Logic + Reinforcement Learning + Deep Learning: A Survey

Citation for published version:
Bueff, A & Belle, V 2023, Logic + Reinforcement Learning + Deep Learning: A Survey. in *Proceedings of the 15th International Conference on Agents and Artificial Intelligence, 2023.* vol. 3, International Conference on Agents and Artificial Intelligence, SCITEPRESS, pp. 713-722, The 15th International Conference on Agents and Artificial Intelligence, 2023, Lisbon, Portugal, 22/02/23. https://doi.org/10.5220/0011746300003393

Digital Object Identifier (DOI):
10.5220/0011746300003393

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 15th International Conference on Agents and Artificial Intelligence, 2023

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Mar. 2025

Logic + Reinforcement Learning + Deep Learning: A Survey

Andreas Bueff and Vaishak Belle
University of Edinburgh, U.K.

Keywords: Logic-Based Reinforcement Learning.

Abstract: Reinforcement learning has made significant strides in recent years, including in the development of Atari and Go-playing agents. It is now widely acknowledged that logical syntax adds considerable flexibility in both the modelling of domains as well as the interpretability of domains. In this survey paper, we cover the fundamentals of how logic, reinforcement learning, and deep learning can be unified, with some ideas for future work.

1 INTRODUCTION

Reinforcement learning (RL) has made significant strides in recent years, including in the development of Atari and Go-playing agents (Denil et al., 2016; Foerster et al., 2016; Mnih et al., 2013). It is now widely acknowledged that logical syntax adds considerable flexibility in both the modelling of domains as well as the interpretability of domains (Milch et al., 2005; Hu et al., 2016; Narendra et al., 2018; Graves et al., 2014). In this survey paper, we cover the fundamentals of how logic, reinforcement learning, and deep learning can be unified, with some ideas for future work. In some cases, we go into some depth into the technical details because the modules depend on such details.

We begin with a brief review of inductive logic programming (ILP), the main paradigm we focus on in this article (Raedt et al., 2016). Although there are other approaches, such as (Sanner and Kersting, 2010), which uses Partially-observable Markov decision processes (POMDPs) as a powerful model for sequential decision-making problems with partially-observed states, and those that apply satisfiability (SAT) logical representations in Discrete Hopfield Neural Networks (DHNN) (Zamri et al., 2022; Guo et al., 2022; Chen et al., 2023), we focus on ILP because it allows the learning of rules, a powerful facet for understanding the policies learned by agents, and could play an important role in explainable AI (Bhatt et al., 2020). After discussing the basics of RL, we will delve into recent advances in extending ILP and differential ILP, which allows for neural integration in the learning of logical rules with RL.

2 LOGIC PROGRAMMING

Logic programming is a programming paradigm with relational logic (Kowalski, 1974), composed of a tuple (R,F) where R is a set of rules and F a set of facts. This tuple encapsulates an if-then rule. Such rules are known as *clauses* and a definite clause is a rule composed of a head atom α and a body $\alpha_1,\cdots,\alpha_m$, formally $\alpha \leftarrow \alpha_1,\cdots,\alpha_m$ with rules read from right to left. In order for α to be true, all atoms to the right must also be true. An atom α is a tuple $p(t_i,\cdots,t_n)$ where p is a predicate with an arity of n in conjunctive normal form (CNF). The body of the predicate is defined by the terms $t_1,\cdots,t_n$ which can be variables or constants. To ground an atom means to have terms defined by constants (Evans and Grefenstette, 2017).

2.1 ILP

Inductive logic programming (ILP) is a method of symbolic computing which can automatically construct logic programs provided a background knowledge base (KB) (Muggleton and de Raedt, 1994). An ILP problem is represented as a tuple $(\mathcal{B},\mathcal{P},\mathcal{N})$ of ground atoms, with $\mathcal{B}$ defining the background assumptions, $\mathcal{P}$ is a set of positive instances which help in defining the target predicate to be learned, and $\mathcal{N}$ defines the set of negative instances of the target predicate. The aim of ILP is to eventually construct a logic program that explains all provided positive sets and rejects the negative ones. Given an ILP problem $(\mathcal{B},\mathcal{P},\mathcal{N})$, the aim is to identify a set of hypotheses (clauses) $\mathcal{H}$ such that (Muggleton and de Raedt, 1994):

- $\mathcal{B}\wedge\mathcal{H}\models\gamma$ for all $\gamma\in\mathcal{P}$
- $\mathcal{B}\wedge\mathcal{H}\not\models\gamma$ for all $\gamma\in\mathcal{N}$

Bueff, A. and Belle, V.
Logic + Reinforcement Learning + Deep Learning: A Survey.
DOI: 10.5220/0011746300003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3*, pages 713-722
ISBN: 978-989-758-623-1; ISSN: 2184-433X
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

Where $\models$ denotes logical entailment. Thus stating, that the conjunction of the background knowledge and hypothesis should entail all positive instances and the same should not entail any negative instances. We assume for example a KB with provided constants $\{bob, carol, volvo, jacket, pants, skirt, \cdots\}$, where the task is to learn the predicate $\texttt{Passenger}(X)$. Then the ILP problem is defined as:

- $\mathcal{B} = \{\texttt{Car}(ford), \texttt{Clothing}(jacket), \texttt{On}(jacket, bob), \texttt{Inside}(carol, volvo), \cdots\}$
- $\mathcal{P} = \{\texttt{Passenger}(bob), \texttt{Passenger}(carol), \cdots\}$
- $\mathcal{N} = \{\texttt{Passenger}(volvo), \texttt{Passenger}(jacket), \cdots\}$

The outcome of the induction performed is a hypothesis of the form:

$$\texttt{Passenger}(X) \leftarrow \texttt{Inside}(X, Y_1) \wedge \texttt{Car}(Y_1) \wedge \texttt{On}(Y_2, X) \wedge \texttt{Clothing}(Y_2)$$

The learned first-order logic rule from the KB states *"if an object is inside the car with clothing on it, then it is a passenger"*.

The ILP problem may also contain a language frame $\mathcal{L}$ and program template Π (Evans and Grefenstette, 2017). The language frame is a tuple which contains information on the target predicate, the set of extensional predicates, the arity of each predicate, and a set of constants, while the program template describes the range of programs that can be generated.

2.2 dILP

To make ILP more robust to noise in data, Evans et al. expanded the framework by making it differentiable (Evans and Grefenstette, 2017). Doing so provides a loss function which can be minimised with stochastic gradient descent. Differential inductive logic programming (*d*ILP) utilises a continuous interpretation of semantics where atoms are mapped to values $[0,1]$. *d*ILP requires a valuation vector $[0,1]^n$ with n atoms which map each ground atom $\gamma \in G$ to a real unit interval. Also defined is an atom label Λ which pairs a true binary identifier λ to atoms in $\mathcal{P}(\lambda = 1)$ and false binary to atoms in $\mathcal{N}(\lambda = 0)$.

Provided an ILP problem $(\mathcal{L}, \mathcal{B}, \mathcal{P}, \mathcal{N})$, a program template and class weights W. The differential model we are trying to solve for an atom α is $p(\lambda|\alpha, W, \Pi, \mathcal{L}, \mathcal{B})$ where λ acts as a classification problem for α. Thus, this sets our goal as minimising the expected negative log-likelihoods for pairs (α, λ).

$$loss = -\mathbb{E}_{(\alpha,\lambda)\sim\Lambda}[\lambda \log p(\lambda|\alpha, W, \Pi, L, B) + (1-\lambda) \log p(1 - p(\lambda|\alpha, W, \Pi, L, B))]$$

By forward chaining T steps, we can calculate the consequences of applying the rules, so-called "Conclusion Valuation". The probability of λ given α is computed using the auxiliary functions f_{exract}, f_{infer}, f_{covert}, and $f_{generate}$. We refer to the literature (Evans and Grefenstette, 2017) for further details on the calculations of the remaining auxiliary functions.

2.3 dNL

Payani et al. proposed an extension to ILP called differentiable neural logic (dNL) networks which utilise differentiable neural logic layers to learn Boolean functions (Payani and Fekri, 2019). The concept is to define Boolean functions that can be combined in a cascading architecture akin to neural networks. Giving deep learning an explicit symbolic representation that is interpretable, and redefines ILP as an optimisation problem. The dNL architecture uses membership weights and conjunctive and disjunctive layers with forward chaining to remove the need for the rule template to solve ILP problems.

Payani et al. mapped Boolean values ($true = 1, false = 0$) to real value ranges $[0,1]$, and defined fuzzy unary and dual Boolean functions of variables x and y as follows:

- $\bar{x} = 1 - x$
- $x \wedge y = xy$
- $x \vee y = 1 - (1-x)(1-y)$

Given an input vector $\mathbf{x}^n$ into the logical neuron, the implementation of the conjunction function requires first selection of a subset of $\mathbf{x}^n$ and applying multiplication(conjunction) of selected elements. Trainable Boolean membership weights m_i are also associated with each input element x_i from vector $\mathbf{x}^n$. The conjunction function in Equation 1 is defined by the Boolean function defined in Equation 2 which is able to include each element in the conjunction function.

$$f_{conj}(x^n) = \prod_{i=1}^{n} F_c(x_i, m_i) \tag{1}$$

$$F_c(x_i, m_i) = \overline{\bar{x}_i m_i} = 1 - m_i(1 - x_i) \tag{2}$$

By combining different Boolean functions, a multi-layered structure can be created. For example, cascading a conjunction function with a disjunction function(see Equation 3) creates a layer in disjunction normal form (DNF), so-called dNL-DNF.

$$f_{disj}(x^n) = 1 - \prod_{i=1}^{n} (1 - F_d(x_i, m_i)) \tag{3}$$

$$F_d(x_i, m_i) = x_i m_i \tag{4}$$

A dNL conjunction function F_p^i is associated with the i^{th} rule of intensional predicate p in a logic program and the membership weights m_i act as Boolean flags to each atom. As membership weights can be optimised, we find that dNL is similar to dILP. However, dILP maps Boolean flags to permutations of two atom sets and only uses the weights to infer the Boolean flags in selecting a single winning clause. We refer the reader to (Payani and Fekri, 2019) for a further description of the forward chaining for evaluation.

3 REINFORCEMENT LEARNING

Reinforcement learning (RL) seeks to solve the problem where an agent is tasked with learning an optimal action sequence to maximise the expected future reward (Sutton et al., 1998; Rodriguez et al., 2019). The environment $\mathcal{E}$ that an agent operates on is often modelled as a Markov Decision Process (MDP). An MDP is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$ where $\mathcal{S}$ is set of states, $\mathcal{A}$ is a set of actions that can be taken, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the reward function which takes as input the current state s_t, current action a_t, and provides reward from the transition to state s_{t+1}, $\mathcal{R}(s_t, a_t, s_{t+1})$. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition probability function which represents the probability of transitioning to state s_{t+1} from state s_t given action a_t was taken $p(s_{t+1}|s_t, a_t)$. $\mathcal{E} \subset \mathcal{S}$ are the set of terminal states (Sutton et al., 1998; Leike et al., 2018). A discount factor $\gamma \in [0,1]$ states how important it is to receive a reward in the current state versus the future state where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated return from the time step. The value function $V^{\pi}(\tau) = \mathbb{E}_{\pi}[R_t]$ of a policy is the measure of the expected sum of discounted rewards r_t. The objective is to find the optimal policy $\pi(s) : S \rightarrow A$ to maximise the reward over time defined by $\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t_{r_t} | s_0 = s]$ (Sutton et al., 1998; Rodriguez et al., 2019; Leike et al., 2018; Roderick et al., 2017; Icarte et al., 2018; Mnih et al., 2016). This is done by mapping a history of observations $\tau_t = (a_0, s_0, a_1, s_1, \cdots, a_{t-1}, s_t)$ to the next action a_t. In practice, at each time step t, the agent is in a particular state $s_t \in \mathcal{S}$, select an action a_t, and according to $\pi(\cdot|s_t)$, executes a_t.

Value Based Methods: depend on V^{π} to find the best action to take for each state. This method benefits from finite states. The Q-function value $Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a]$ under a policy π is the expected return for selecting action a in state s and following policy π. The Q-function updates using the Bellman equation, where the Bellman equation is a fundamental relationship between the value of a state action pair (s,a) and the value of subsequent state-action pair (s', a') (Sutton and Barto, 2012).

$$Q^{\pi}(s,a) = r + \gamma \mathbb{E}_{s',a'}[Q^{\pi}(s',a')], \quad a' \backsim \pi(s') \tag{5}$$

$$Q^*(s,a) = \arg\max_{\pi} Q^{\pi}(s,a) \tag{6}$$

For deep learning and the Q-function, we parameterise the Q function $Q(s,a;\theta)$ where the action value function is represented by a neural network.

Policy Based Methods: seek to directly learn a policy $\pi_\theta(s)$, parameterised by θ as opposed to a value function as in Q-learning. This is useful when the action space is continuous or stochastic. Gradient ascent on $\mathbb{E}[R_t]$ is done to update the parameters θ for a parameterised policy $\pi(a|s;\theta)$. REINFORCE is a policy based algorithm that updates policy parameters θ in the direction $\delta_\theta log \pi(a_t|s_t;\theta) R_t$ which is an unbiased estimate of $\delta_\theta \mathbb{E}[R_t]$. By subtracting the learned function of the state, the baseline $b_t(s_t)$, it is possible to reduce the variance of the estimate. The resulting gradient is $\delta_\theta log \pi(a_t|s_t;\theta)(R_t - b_t(s_t))$ (Sutton et al., 1998; Mnih et al., 2016).

A mixed approach which takes advantage of both value and policy-based methods is *Actor-Critic* where a learned estimate of the value function is commonly used as the baseline $b_t(s_t) = V^{\pi}(s_t)$ leading to a lower variance estimate of the policy gradient. The quantity $R_t - b_t$ used in the policy gradient can be seen as an estimate of the *advantage* of action a_t in state s_t, or $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ because R_t is an estimate of $Q^{\pi}(a_t, st)$ and b_t is an estimate of $V^{\pi}(s_t)$. The critic estimates the value function which can be either $Q^{\pi}(a,s)$ or $V^{\pi}(s_t)$. The actor updates the policy distribution in the direction suggested by the critic via policy gradients (Sutton et al., 1998).

A model is the internal representation of the environment for an agent, and this is often represented with the transition function $\mathcal{T}$ and the reward function $\mathcal{R}$ in an MDP. For RL, the approach with regard to models is either model-based or model-free. Model-based methods tasks an RL agent where the model is input and an agent learns a policy through interacting with the model, so-called planning. The opposite of planning is seen with the trial and error approach of model-free which relies on learning state and action values to achieve a goal, solely through interacting with the environment (Sutton et al., 1998). While model-free methods lack sample efficiency from using a model, they tend to be easier to implement and tune.

With planning, the agent's observation of the environment results in the computation of the optimal policy with respect to the model. The policy in this framework describes a series of actions over a fixed

time window to achieve a goal. The agent executes the first action of a plan then discards the rest and sees to it that each interaction with the environment generates a new plan.

Model-free methods include policy optimisation methods, where the parameters for a policy are learned via gradient ascent on the performance objective $J(\pi_\theta)$. Here the learning is done on-policy so each update uses information learned while acting in accordance with the most recent version of the policy. Q-learning is the other model-free method which is off-policy and learns the optimal action-value function $Q^*(s,a)$ so each update uses information collected at any point during training.

3.1 Relational RL

Relational Reinforcement learning (RRL) seeks to combine ILP or relational learning with the general RL framework. RRL has found applications in planning tasks (model-based) such as the simple block world task seen in Figure 1. RRL like classic RL is tasked with learning a policy that provides agents with the optimum actions to take in a given state. As RRL integrates ILP, a state is represented relationally as a set of ground facts. As a symbolic approach, RRL learns first-order logic (FOL) rules as a policy for a given task (Driessens, 2010).

As RRL initially was developed for planning (Driessens, 2010) the block worlds problem was used as a basis for testing. The objective in planning problems is to start from a state s and find a sequence of actions $a_i, \cdots, a_n$ with $a_i \in \mathcal{A}$ such that $goal((\cdots,(s,a_1))\cdots,a_{n-1}) = true$ and $pre((\cdots(s,a_1)\cdots,a_{i-1}),a_i) = true$. In the case of block worlds, say we have 3 blocks called a,b,c on the $floor$, then states can be described by blocks either being on each other or the floor, and actions are represented by $move(x,y)$ where $x \neq y$ and $x \in \{a,b,c\}, y \in \{a,b,c,floor\}$. An example state is given:

$$s_1 = \{clear(a), on(a,b), on(b,c), on(c,floor)\}$$

The initial work into RRL (Driessens, 2010) used a modified Q-learning algorithm with a standard relational learning algorithm (TILDA). Background knowledge (BK) provides general predicates that can be used for induction, where predicates are valid for all training examples. As BK is prior knowledge defined in an expert-defined predicate language, RRL allows experts to input inductive biases to restrict the search space. This not only provides policies that are crafted for the expert's interpretation but also speeds up the learning process (Payani and Fekri, 2020).

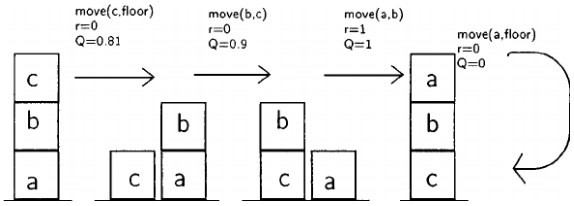

Figure 1: Relational reinforcement learning in the box world environment (Driessens, 2010).

RRL provides several benefits over classical RL: (1) The learned policy is human interpretable and can be analysed by an expert. (2) The learned RRL policy is also better able to generalise than policies learned under classical RL. (3) As the language framework for defining states and actions is reliant on the expert, inductive bias can be injected into learning which entails that the expert can control agent behaviour to a degree. (4) RRL allows for prior background knowledge to act as heuristics in learning (Driessens, 2010; Payani and Fekri, 2020).

Formally, the task formulation of RRL is given as:

1. States are described relationally using predicates.
2. Actions are described with relational language such as that in the box world.

When combined with Q-learning, the standard RRL algorithm uses the stochastic selection of actions and a relational regression tree. Similarly, the Q-values are instantiated to zero.

Recent works in RRL have sought to learn end-to-end in order to avoid the restrictions of regression trees. In work done by Jiang et al. RRL is extended to learn FOL policies with policy gradients (Jiang and Luo, 2019). The authors apply dILP to sequential decision-making tasks such as the block world task and cliff walking task.

Taking dILP's valuation vector $\mathbf{e}$, the authors expand on its architecture to create so-called Differential recurrent logic machines (DRLM) which perform iterative deduction on the valuation vector by taking the probabilistic sum of possible grounded atoms $\mathbf{a}$ and $\mathbf{b}$, denoted as $\oplus$ and defined as $\mathbf{a} \oplus \mathbf{b} = \mathbf{a} + \mathbf{b} - \mathbf{a} \odot \mathbf{b}$, where $\odot$ is the component-wise product. DRLM treats the valuation produced by the last step deduction only as input, so the last step of the deduction is just a function of the initial valuation vector $\mathbf{e}_0$ and the sum of the deduced valuation. DRLM also extends the expressiveness of dILP policies by assigning weights to individual clauses whereas dILP is limited to combinations of clauses (Jiang and Luo, 2019).

$$f_\theta^t(\mathbf{e}_0) = \begin{cases} g_\theta(f_\theta^{t-1}(\mathbf{e}_0)), & \text{if } t \leq 0 \\ \mathbf{e}_0, & \text{if } t = 0 \end{cases} \tag{7}$$

The deduction of facts $\mathbf{e}_0$ using weights w for individual clauses is done iteratively. Each iteration decomposes the valuation vector with a single-step deduction function g_θ as seen in Equation 7 where t is the deduction step. The single-step deduction function g_θ takes in the clause weights and valuation vector on the clause, as seen in Equation 8.

$$g_\theta(\mathbf{e}) = (\overset{\oplus}{\sum_n} \sum_j w_{n,j} F_{n,j}(\mathbf{e})) + \mathbf{e}_0 \tag{8}$$

The function $F_{n,j}$ performs one step deduction on the valuation vector $\mathbf{e}$ with the j^{th} definition of the n^{th} clause.

The authors rely on encoders and decoders to translate first-order atoms to policy valuation and actionable policy for the agent. The state encoder $p_s : S \to 2^G$ maps game world states to sets of atoms, and the decoder $p_a : [0,1]^{|D|} \to [0,1]^{|A|}$ maps atoms back to actions. The authors trained their agent using the REINFORCE algorithm and the design of the encoder and decoder architecture was done with neural networks. In order to ensure the system is differentiable, they represent the probability of actions based on the valuations $\mathbf{e}$.

$$p_A(a|e) = \begin{cases} \frac{l(e,a)}{\sigma}, & \text{if } \sigma \geq 1 \\ \frac{l(e,a)}{\sigma} + \frac{1-\sigma}{|A|}, & \text{if } \sigma < 1 \end{cases} \tag{9}$$

Here, we have the valuation vector $\mathbf{e}$ determining the value of a given action. The first case is the proportional value of an action where the total sum of the valuations is over 1, and if the sum of values is less than one total valuation is evenly distributed to all actions. The function $l : [0,1]^{|D|} \times A \to [0,1]$ maps the valuation vector to the valuation of that action atom and σ is the sum of all action valuations $\sum_a l(e,a)$.

As a model-based approach, Neural Logic Reinforcement Learning (NLRL) is limited to learning in a discrete space. Zambaldi et al. sought to extend RRL to a model-free and continuous-based setting by taking advantage of Deep RL and relational learning (Zambaldi et al., 2019). The authors incorporate relational inductive bias for entity and relation-focused representations by applying an iterative relational reasoning mechanism. In an environment, entities correspond to local regions of an image and the agent learns the values of certain entities and compares their interactions with other entities. Visual data is handled by a convolutional neural network (CNN) front-end and the agent itself learns with the off-policy A2C algorithm.

For the actor, the policy consists of logits over a set of possible actions, and the critic generates a baseline value which is used to calculate the temporal-difference error to optimise π. As mentioned, input for the agent is done via a CNN which takes an image from a game environment and transforms it into an embedded state representation. Entity vectors E are computed by transforming an $m \times n \times f$ feature matrix to an $N \times f$ matrix where $(N = m \cdot n)$ and where f represents the number of feature maps. This allows each row e_i in E to correspond to a feature vector $s_{x,y}$ for any point location x, y in the state image.

The authors tested their framework on the block-worlds problem and on the popular real-time strategy (RTS) game Starcraft 2. The lack of a formal ILP module prevents the policies from being represented in a FOL language, losing some human interpretability. The relational model proposed by the authors is defined as a stack of multiple relational blocks, and these blocks perform one-step relational updating using a shared recurrent neural network (RNN) on unshared parameters to derive higher-order entity relations.

Relational reasoning on the entity interactions is performed by taking the entity interactions $p_{i,j}$ and updating each entity based on interactions in the gaming environment. The self-attention network, so-called multi-head dot-product attention (MHDPA), projects the entities E into matrices query (Q), key (K), and value (V) to compute the similarities. The dot product is taken for a query q_i and all keys $k_{j=1:N}$, which are then normalised via a softmax function into attention weights w_i.

The pairwise interactions are computed using the attention weights and value matrix $p_{i,j} = w_{i,j}\mathbf{v}_j$. The equation 10 defines the transformation performed to derive the accumulated interactions with d defining the number of dimensions of the Q and K matrices $(N \times d)$.

$$A = softmax\Big(\frac{QK^T}{\sqrt{d}}\Big)V \tag{10}$$

Updates to the entities E' is done via a multi-layer perception (MLP) g_θ which takes as input the accumulated interactions $e' = g_\theta(a_i^{h=1:H})$.

Another approach for end-to-end learning with model-based agents is the work by Dong et al. In their work they develop so-called, Neural Logic Machines (NLM), for a variety of relational reasoning tasks including the RRL setting of the block worlds problem (Dong et al., 2019). Like NLRL, NLM is able to learn lifted rules from a policy which are generalisable and handle high-order relational data, for example, the transitional rule $r(a,c) \leftarrow \exists b\ r(a,b) \wedge r(b,c)$ where reasoning is performed on objects (a,b,c). Another common issue addressed in this work is that of ILP's issue with scaling. In (Yang and Song, 2019),

MLPs for NLRL act as decoders/encoders while in the following work, MLPs are integrated into the rule evaluation and generation process.

The logical framework of ILP is refactored as so-called logic predicate tensors. The NLM architecture allows for neural-symbolic realisations of Horn clauses in FOL. A Horn clause is a clause with at most one positive literal such that a rule $\hat{p} \leftarrow p_1(x) \wedge p_2(x)$ implies $\forall \hat{p} \leftarrow p_1(x) \wedge p_2(x)$. Using MLPs, an NLM is able to learn logical operations such as AND and OR. NLMs take as input a KB comparing base predicates and variable objects. NLMs apply FOL rules to draw conclusions using tensors to represent logic predicates. Predicates are grounded with permutations of possible variable input and evaluated as being true or false. The grounded predicates are represented by logic tensors and implemented by neural operators for a sequential logic deduction. The final output is a set of generalisable rules which contain the probability of variables and their relations.

The authors combine a probabilistic view on a logical predicate with their so-called *U-grounding* tensor p^u provided a set of variables $U = \{u_1, u_2, \cdots, u_m\}$, and predicates $p(x_1, x_2, \cdots, x_r)$-arity r, which can be grounded as a tensor of size $[m^{\hat{r}}]$ where $m^{\hat{r}}$ is defined as $[m, m-1, m-2, \cdots, m-r+1]$. The properties of a variable are defined here as unary predicates "$moveable(x)$", relations of predicates are defined as a binary predicate "$on(x,y)$", and global properties are nullary predicates "$allmatched()$". The grounding of each entity $p^u(u_{i1}, u_{i2}, \cdots, u_{ir})$ represents whether p is true provided a given instantiation.

The authors extend this representation of a logic predicate p^u by stacking them, they define $C^{(r)}$ to be the number of predicates of arity r. This creates a tensor of size $[m^{\hat{r}}, C^{(r)}]$. NLM applies a probabilistic interpretation by taking each ground predicate and deriving a likelihood $\in [0,1]$ for being true.

The modules of the architecture are reliant on select meta-rules. (1) Boolean logic rules for operations AND, OR, NOT and (2) quantifications which link predicates of different arities via logic quantifiers $(\forall, \exists)$. The neural Boolean logic rule is predicate logic in the form $\hat{p}(x_1, x_2, \cdots, x_r) \leftarrow expression(x_1, x_2, \cdots, x_r)$ and can be defined as an example $moveable(x) \leftarrow \neg isground(x) \wedge clear(x)$. In the context of NLM, provided a set of predicates $P = \{p_1, p_2, \cdots, p_k\}$ with all the same arity, they can be stacked as a tensor of shape $[m^{\hat{r}}, |P|]$ with arbitrary permutations of potential groundings. An MLP takes as input all the permutations of p_i^u to create $r!$ tensors with shape $[m^{\hat{r}}, r! \times |P|]$ as well as trainable parameters θ. A sigmoid is applied to the MLP to derive the likelihood of a specific predicate.

The neural quantifiers are defined by two rules, (1) Expansion which constructs a new predicate q from p by introducing a new variable x_{r+1}, where x_{r+1} is not a previous variable in the body of the predicate:

$$\forall x_{r+1} q(x_1, x_2, \cdots, x_r, x_{r+1}) \leftarrow p(x_1, x_2, \cdots, x_r) \quad (11)$$

The $Expand()$ method, given a set of C r-arity predicates with a shape $[m^r, C]$, expands the logic predicate tensor by the addition of a distinct variable x_{r+1}. The additional dimension to the tensor creates a shape $[m^{\hat{r+1}}, C]$. The second rule is reduction, where using either the $\forall, \exists$ quantifier, reduces the arity of a body request by marginalising over a given variable.

$$q(x_1, x_2, \cdots, x_r) \leftarrow \forall x_{r+1} p(x_1, x_2, \cdots, x_r, x_{r+1}) \quad (12)$$

The implementation $Reduce()$ is just the reverse architecture of $Expand()$ where given a $[m^{\hat{r+1}}, C]$ shaped tensor with a set of $C_{(r+1)}$-arity predicates reduces the tensor along the x_{r+1} dimension by taking the max and min and stacking the resulting tensors for a dimension of $[m^{\hat{r}}, 2C]$.

The authors combine these meta-rules to create the formal NLM, comprising D layers with $B+1$ computations per layer. NLM computes one split into two distinct computational phases. Provided an input layer O_{i-1} to produce the output layer $O_i = \{O_i^{(0)}, O_i^{(1)}, \cdots, O_i^{(B)}\}$. The inter-group computation relies on Equation 11 and 12 which takes in tensors from say layer i and connects the previous layer $i-1$ into vertically neighbouring groups of arity r, $r+1$, etc. Aligning of the dimensions is done by $Reduce()$ or $Expand()$ to form the tensor $I_i^{(r)}$.

$$I_i^{(r)} = Concat(Expand(O_{i-1}^{(r-1)}), O_{i-1}^{(r)}, Reduce(O_{i-1}^{r+1})) \quad (13)$$

This addresses the issue of the aligning predicates of the proximal arities and different orders.

Intra-group computation takes the tensor from the inter-group computation $I_i^{(r)}$ and permutes the predicate inputs to produce the output tensor $O_i(r)$. The final output shape of the tensor is $[m^{\hat{r}}, C_i^{(r)}]$.

$$O_i^{(r)} = \sigma(MLP(Permute(I_i^{(r)}; \theta_i(r)))). \quad (14)$$

Taking the concepts of dNL (Section 2.3), Payani et al. combine their dNL framework with RL (Payani and Fekri, 2020). Like NLRL, the author test on the block world gaming environment and take advantage of the declarative bias with provided BK. The authors seek to expand RRL to handle complex scene interpretations similarly to standard deep RL. Payani et al. take their dNL-ILP differentiable deductive engine

to give RRL an end-to-end learning framework, so-called dNL-RRL. This allows for a deep RL approach with ILP. The authors focus on-policy gradients in order to improve interpretability and also regression in the form of expert constraints.

RRL has been limited in the past by non-differentiable ILP (Muggleton and De Raedt, 1994), so we have seen an overreliance on explicit relational representations of states and actions in the BK. Having differentiable ILP with the dNL-ILP engine allows an RRL agent to learn from raw pixels to extract low-level relational representations from CNNs. The authors made significant changes to the original RRL algorithm with regard to the state representation, language bias, and action representation.

In the first case, state representations, the objective is to take advantage of dNL-ILP for policy learning by extracting explicit state representations via deep learning methods. The authors propose first processing raw images with CNNs and having the last layer act as a feature vector which can contain info such as pixel point position (x,y). Then feed the feature map into a relational unit to extract non-local features (this approach is similar to that seen with (Zambaldi et al., 2019)), however, they do not rely on graph networks instead seeking explicit predicates for ILP).

Three strategies are implemented for learning the desired relational states from raw input. Where lower-level state representations are first learned by CNNs trained on the raw input, followed by the abstraction to higher-level representations. For example, a high-level predicate $on(X,Y)$ can be defined by position predicates $posH(X,Y)$ and $posV(X,Y)$ for X representing the block and Y the coordinates with *posH* referring to the horizontal axis and *posV* the vertical axis. The second strategy is state constraints, which rely on the regularisation of the loss function and connects to logic constraints for relational reasoning. This requires the modeller to define what heuristics to constrain. For example, $posV(floor)$ should always be zero. The third strategy is a semi-supervised setting, where the modeller provides input which defines certain states with labels to aid the agent in learning. This information is added to the loss function as well.

The action representation is handled by having the action probability distribution mapped to the grounding of true predicates, for example, $move(X,Y)$ inferring possible permutations of $move(X,Y)$ that are true in a given state. In the case of multiple true predicates of $move(X,Y)$, the probability distribution can be estimated by applying a softmax function to the valuation vector on the learned predicate *move*.

Outside of RRL, other avenues seeking to combine RL with symbolic learning or explanations have also been explored. Similar to the abstraction approach of Payani et al., Verma et al. also pursued a similar approach in their contribution (Verma et al., 2018), where the authors present Programmatically Interpretable Reinforcement Learning (PIRL) and Neurally Directed Program Search (NDPS). PIRL is able to present policies in a human-readable language, not unlike the dNL-ILP policies. The parameters of PIRL include a high-level programming language to define policies. These parameters infer a $policy(sketch)$ which defines the declarative language for an RL problem in a high-level format. The ideal $policy(sketch)$ uses a language to maximise long-term reward. The primary benefits of PIRL's language are first the interpretable output, followed by encoding the modeller's bias, effective pruning of undesired policies, and then a symbolic program verification method analogous to ILP. The authors also propose NDPS which uses Deep RL to compute a high-performing yet non-interpretable policy in an RL environment. Inspired by imitation learning (Ho and Ermon, 2016), the authors take the high-performing policy and treat it as an oracle for searching out an interpretable policy defined by PIRL. An interpretable policy is iteratively selected to minimise the output difference between the high-performance policy and itself.

An older approach by Garnel et al. establishes the neural back-end and symbolic front-end framework seen so far (Garnelo et al., 2016). The authors combine deep learning with classical symbolic AI, integrating symbolic elements such as constants, functions, and predicates. Garnele et al. note the limitation of declarative bias in instantiating symbolic semantics for an RL agent to interpret. The back-end works similarly to an auto-encoder to compress raw perceptional data to be fed as input to the symbolic front-end which maps the inputs to actions. The authors also propose a set of principles, which are relevant to all research with regard to symbolic RL. (1) Conceptual abstraction which determines whether a new situation is similar to past experiences and can a connection be drawn. (2) Compositional structure which is a representational medium that has a set of elements that can be recompiled in an open-ended manner such as probabilistic FOL. (3) Common sense priors are the minimal assumptions and expectations that can be built into the learning procedure and (4) Causal reasoning which attempts to discover the causal structure of the domain and express them through the common sense ontology provided. This work was further extended by Garcez et al. (d'Avila Garcez et al., 2018) by providing heuristics to the common sense principle, so-called

principle one which only updates agent and object interaction states based on Q values only when the rewards are non-zero and *principle two* which takes into account the relational position of an object to an agent as a factor for an action selection.

In (Martires et al., 2020), real-world perception in an RL setting was researched where a semantic world modelling approach via detecting, segmentation, and processing perceived objects from visual input and then applying an anchoring system to maintain consistent representations (anchor) of perceived objects. The pipeline ends with an inference system to self-check the anchoring system and for logically tracking objects in a dynamic setting. Predicate logic is used as the language for relational representation.

In (Li et al., 2019), the graph network approach to real-world robotics tasks is also interesting to note. Graph networks provide an alternative to ILP since graph structures are suitable for an object-oriented environment and can model relational relationships between objects. Li et al. have an agent train with an attention-based graph neural network (GNN) to handle curriculum learning with multi-object environments and have the number of objects increase as the agent learns.

Other works have provided a testing bed for RRL research such as the works by Silver et al. (Silver and Chitnis, 2020). The authors bridge the research environment Open AI Gym (Brockman et al., 2016) with planning domains in Planning Domain Definition Language (PDDL) which allows for testing of RL interpretability.

Zhang et al. combine high-level symbolic rules and deep Q-learning (DQN), treating rules defined in a knowledge background as intuitive heuristics such as "slow down when you approach curve" (Zhang et al., 2019). The authors proposed so-called, Rule-interposing Learning (RIL), which has a deep learning component. An agent queries the KB for appropriate actions in addition to Q values and decides what action is appropriate. The KB aids the DQN in pruning the actions that are unsafe or unnecessary. The KB acts as an independent module, as opposed to (Garnelo et al., 2016), which relies on the end-to-end RL architecture with neural back-end and symbolic front-end.

Saebi et al. address the issue of knowledge graph (KG) completion, and problems of inferring missing relations in their work (Saebi et al., 2020). A real-world KG can contain exponential numbers of entity relations, overburdening an RL agent. The authors propose a representation of the state space which includes entity-type information. They then use a pruning algorithm to limit searches made by an agent in the action space. Similarly (Li et al., 2019), the authors use GNNs to encode neighbourhood information.

Sreedharan et al. research into RRL explore so-called, Expectation-aware planning, which includes a 'human in the loop' for agent learning. The authors propose self-explaining plans which contain actions that are responsible for explaining the plan in an interpretable manner and seek to include agent behaviour acting in accordance with human expectation (Sreedharan et al., 2019).

Pacheo et al. seek to combine GNNs and end-to-end learning of deep learning in their seminal work (Pacheco et al., 2018). The authors present DRail, a declarative modelling language which consists of rules sets with factor templates over variables. Rules in the framework have an associated neural architecture to learn a scoring function and feature definition.

A unique application, Lederman et al. rely on graph networks and relational RL, primarily for solving quantified Boolean logics and returning proofs. Here they utilise a GNN to predict the quality of each variable as a decision variable to select an action and use end-to-end learning to improve reasoning on given sets of formulas (Lederman et al., 2018).

4 CONCLUSIONS

In this survey paper, we studied the various dimensions and some technical foundations for logic + reinforcement learning + deep learning. There is a growing body of work on logic and deep learning (Garcez et al., 2015), and this survey pushes the area further by looking at dynamic domains. In an extended report, we also discuss advances in inverse RL (Ng and Russell, 2000) and its integration with ILP.

ACKNOWLEDGEMENTS

This research was partly supported by a Royal Society University Research Fellowship, UK, and partly supported by a grant from the UKRI Strategic Priorities Fund, UK to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020–2024).

REFERENCES

Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J. M. F., and Eckersley,

P. (2020). Explainable machine learning in deployment. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Chen, J., Kasihmuddin, M. S. M., Gao, Y., Guo, Y., Asyraf Mansor, M., Romli, N. A., Chen, W., and Zheng, C. (2023). Pro2sat: Systematic probabilistic satisfiability logic in discrete hopfield neural network. *Advances in Engineering Software*, 175:103355.

d'Avila Garcez, A. S., Dutra, A. R. R., and Alonso, E. (2018). Towards symbolic reinforcement learning with common sense. *CoRR*, abs/1804.08597.

Denil, M., Agrawal, P., Kulkarni, T. D., Erez, T., Battaglia, P., and de Freitas, N. (2016). Learning to perform physics experiments via deep reinforcement learning.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. (2019). Neural logic machines. *CoRR*, abs/1904.11694.

Driessens, K. (2010). *Relational Reinforcement Learning*, pages 857–862. Springer US, Boston, MA.

Evans, R. and Grefenstette, E. (2017). Learning explanatory rules from noisy data. *CoRR*, abs/1711.04574.

Foerster, J. N., Assael, Y. M., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning.

Garcez, A., Besold, T., De Raedt, L., Földiák, P., Hitzler, P., Icard, T., Kühnberger, K.-U., Lamb, L., Miikkulainen, R., and Silver, D. (2015). Neural-symbolic learning and reasoning: Contributions and challenges.

Garnelo, M., Arulkumaran, K., and Shanahan, M. (2016). Towards deep symbolic reinforcement learning. *CoRR*, abs/1609.05518.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines.

Guo, Y., Kasihmuddin, M. S. M., Gao, Y., Mansor, M. A., Wahab, H. A., Zamri, N. E., and Chen, J. (2022). Yran2sat: A novel flexible random satisfiability logical rule in discrete hopfield neural network. *Advances in Engineering Software*, 171:103169.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc.

Hu, Z., Ma, X., Liu, Z., Hovy, E. H., and Xing, E. P. (2016). Harnessing deep neural networks with logic rules. *CoRR*, abs/1603.06318.

Icarte, R. T., Klassen, T., Valenzano, R., and McIlraith, S. (2018). Using reward machines for high-level task specification and decomposition in reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116, Stockholmsmässan, Stockholm Sweden. PMLR.

Jiang, Z. and Luo, S. (2019). Neural logic reinforcement learning.

Kowalski, R. A. (1974). Predicate logic as programming language. In *IFIP Congress*, pages 569–574.

Lederman, G., Rabe, M. N., Lee, E. A., and Seshia, S. A. (2018). Learning heuristics for quantified boolean formulas through deep reinforcement learning.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., and Legg, S. (2018). Scalable agent alignment via reward modeling: a research direction. *CoRR*, abs/1811.07871.

Li, R., Jabri, A., Darrell, T., and Agrawal, P. (2019). Towards practical multi-object manipulation using relational reinforcement learning.

Martires, P. Z. D., Kumar, N., Persson, A., Loutfi, A., and Raedt, L. D. (2020). Symbolic learning and reasoning with noisy data for probabilistic anchoring.

Milch, B., Marthi, B., Russell, S. J., Sontag, D., Ong, D. L., and Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *Proc. IJCAI*, pages 1352–1359.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Muggleton, S. and de Raedt, L. (1994). Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629 – 679. Special Issue: Ten Years of Logic Programming.

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

Narendra, T., Sankaran, A., Vijaykeerthy, D., and Mani, S. (2018). Explaining deep learning models using causal inference. *CoRR*, abs/1811.04376.

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Pacheco, M. L., Dalal, I., and Goldwasser, D. (2018). Leveraging representation and inference through deep relational learning. *NeurIPS Workshop on Relational Representation Learning*.

Payani, A. and Fekri, F. (2019). Inductive logic programming via differentiable deep neural logic networks. *CoRR*, abs/1906.03523.

Payani, A. and Fekri, F. (2020). Incorporating relational background knowledge into reinforcement learning via differentiable inductive logic programming.

Raedt, L. D., Kersting, K., Natarajan, S., and Poole, D. (2016). Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189.

Roderick, M., Grimm, C., and Tellex, S. (2017). Deep abstract q-networks.

Rodriguez, I. D. J., Killian, T. W., Son, S., and Gombolay, M. C. (2019). Interpretable reinforcement learning via differentiable decision trees. *CoRR*, abs/1903.09338.

Saebi, M., Krieg, S., Zhang, C., Jiang, M., and Chawla, N. (2020). Heterogeneous relational reasoning in knowledge graphs with reinforcement learning.

Sanner, S. and Kersting, K. (2010). Symbolic dynamic programming for first-order pomdps. In *Proc. AAAI*, pages 1140–1146.

Silver, T. and Chitnis, R. (2020). Pddlgym: Gym environments from pddl problems.

Sreedharan, S., Chakraborti, T., Muise, C., and Kambhampati, S. (2019). Planning with explanatory actions: A joint approach to plan explicability and explanations in human-aware planning. *CoRR*, abs/1903.07269.

Sutton, R. S. and Barto, A. G. (2012). *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. (2018). Programmatically interpretable reinforcement learning. *CoRR*, abs/1804.02477.

Yang, Y. and Song, L. (2019). Learn to explain efficiently via neural logic inductive learning.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., and Battaglia, P. (2019). Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*.

Zamri, N. E., Azhar, S. A., Sidik, S. S. M., Mansor, M. A., Kasihmuddin, M. S. M., Pakruddin, S. P. A., Pauzi, N. A., and Nawi, S. N. M. (2022). Multi-discrete genetic algorithm in hopfield neural network with weighted random k satisfiability. *Neural Computing and Applications*, 34(21):19283–19311.

Zhang, H., Gao, Z., Zhou, Y., Zhang, H., Wu, K., and Lin, F. (2019). Faster and safer training by embedding high-level knowledge into deep reinforcement learning.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS