Neural Logic Reinforcement Learning

Zhengyao Jiang [1] **Shan Luo** [1]

Abstract

Deep reinforcement learning (DRL) has achieved significant breakthroughs in various tasks. However, most DRL algorithms suffer a problem of generalising the learned policy which makes the learning performance largely affected even by minor modifications of the training environment. Except that, the use of deep neural networks makes the learned policies hard to be interpretable. To address these two challenges, we propose a novel algorithm named Neural Logic Reinforcement Learning (NLRL) to represent the policies in the reinforcement learning by first-order logic. NLRL is based on policy gradient methods and differentiable inductive logic programming that have demonstrated significant advantages in terms of interpretability and generalisability in supervised tasks. Extensive experiments conducted on cliff-walking and blocks manipulation tasks demonstrate that NLRL can induce interpretable policies achieving near-optimal performance, while demonstrating good generalisability to environments of different initial states and problem sizes.

1. Introduction

In recent years, deep reinforcement learning (DRL) algorithms have achieved stunning achievements in vairous tasks, e.g., video game playing (Mnih et al., 2015) and the game of Go (Silver et al., 2017). However, similar to traditional reinforcement learning algorithms such as tabular TD-learning (Sutton & Barto, 1998), DRL algorithms can only learn policies that are hard to interpret (Montavon et al.) and cannot be generalized from one environment to another similar one (Wulfmeier et al., 2017).

The interpretability is a critical capability of reinforcement learning algorithms for system evaluation and improvement. A common practice to analyse the learned policy of a DRL agent is to observe the behaviours of the agent in different circumstances and then model how the agent make decisions by characterising the observed behaviours. Such a practice of induction-based interpretation is straightforward but the obtained decisions made by the agent in such systems might just be caused by coincidence. In addition, by simply observing the input-output pairs, it lacks rigorous procedures to determine the beneath reasoning of a neural network. The interpretable reinforcement learning, e.g., relational reinforcement learning (Džeroski et al., 2001), has the potential to improve the interpretability of the decisions made by the reinforcement learning algorithms and the entire learning process. The interpretability of such algorithms also makes it convenient for a human to get involved in the system improvement iteration as interpretable reinforcement learning is easier to understand, debug and control.

The generalizability is also an essential capability of the reinforcement learning algorithm. In the real world, it is not common that the training and test environments are exactly the same. However, most DRL algorithms have the assumption that these two environments are identical, which makes the robustness of DRL remains a critical issue in real-world deployments. An example is the reality gap in the robotics applications that often makes agents trained in simulation inefficient once transferred in the real world. In addition, the problem of sparse rewards is common in the agent systems. A DRL system of good generalizability can train the agent in easier and smaller scale problems and use the learned policies to solve larger problems where rewards cannot be easily acquired by random moves.

In contrast to the neural network based DRL algorithms, the interpretability and generalization are the advantages of symbolic AI (Džeroski et al., 2001). However, symbolic methods are not differentiable that make them not applicable to advanced DRL algorithms. To address this challenge, recently Differentiable Inductive Logic Programming (DILP) has been proposed in which a learning model expressed by logic states can be trained by gradient-based optimization methods (Evans & Grefenstette, 2018; Rocktäschel & Riedel, 2017; Cohen et al., 2017). Compared with traditional symbolic logic induction methods, with the use of gradients for optimising the learning model, DILP has significant advantages in dealing with stochastic-

[1]Department of Computer Science, University of Liverpool, Liverpool, United Kingdom.Correspondence to: Zhengyao Jiang<Z.Jiang22@student.liverpool.ac.uk>, Shan Luo<Shan.Luo@.liverpool.ac.uk>

ity (caused by mislabeled data or ambiguous input) (Evans & Grefenstette, 2018). On the other side, thanks to the strong relational inductive bias, DILP shows superior interpretability and generalization ability than neural networks (Evans & Grefenstette, 2018). However, to the authors' best knowledge, all current DILP algorithms are only tested in supervised tasks such as hand-crafted concept learning (Evans & Grefenstette, 2018) and knowledge base completion (Rocktäschel & Riedel, 2017; Cohen et al., 2017). To make a step further, in this work we propose a novel framework named as Neural Logic Reinforcement Learning (NLRL) to enable the DILP work on sequential decision-making tasks. The extensive experiments on block manipulation and cliff-walking have shown the great potential of the proposed NLRL algorithm in improving the interpretation and generalization of the reinforcement learning in decision making. Furthermore, the proposed NLRL framework is of great significance for advancing the DILP research. By applying DILP in sequential decision making, we investigate how intelligent agents learn new concepts without human supervision, instead of describing a concept already known to the human in supervised learning tasks.

The rest of the paper is organized as follows: In Section 2, related works are reviewed and discussed; In Section 3, an introduction to the preliminary knowledge is presented, including the first-order logic programming ∂ILP and Markov Decision Processes; In Section 4, the NLRL model is introduced, both the DILP architecture and a general NLRL framework modeled with Markov Decision Processes; In Section 5, the experiments of NLRL on block manipulation and cliff-walking are presented; In the last section, the paper is concluded and future directions are directed.

2. Related work

We place our work in the development of relational reinforcement learning (Džeroski et al., 2001) that represent states, actions and policies in Markov Decision Processes (MDPs) using the first order logic where transitions and rewards structures of MDPs are unknown to the agent. To this end, in this section we review the evolvement of relational reinforcement learning and highlight the differences of our proposed NLRL framework with other algorithms in relational reinforcement learning.

Early attempts that represent states by first-order logics in MDPs appeared at the beginning of this century (Boutilier et al., 2001; Yoon et al., 2002; Guestrin et al., 2003), however, these works focused on the situation that transitions and reward structures are known to the agent. In such cases with environment models known, variations of traditional MDP solvers such as dynamic programming (Boutilier et al., 2001), linear programming (Guestrin et al., 2003) and heuristic greedy searching (Yoon et al., 2002) were employed to optimise policies in training tasks that can be generalized to large problems. In these works, the transition and reward functions are also represented in logic forms. The setting limits their application to complex tasks whose transition and reward functions are hard to be modeled using the first order logic.

The concept of relational reinforcement learning was first proposed by (Džeroski et al., 2001) in which the first order logic was first used in reinforcement learning. There are variants of this work (Driessens & Ramon, 2003; Driessens & Džeroski, 2004) that extend the work, however, all these algorithms employ non-differential operations which makes it hard to apply new breakthroughs happened in DRL community. In contrast, in our work using differentiable inductive logic programming, once given the logic interpretations of states and actions, any type of MDPs can be solved with policy gradient methods compatible with DRL algorithms. Furthermore, most such algorithms represent the induced policy in a single clause. Some auxiliary predicates, for example, the predicates that count the number of blocks, are given to the agent. In our work, the DILP algorithms have the ability to learn the auxiliary invented predicates by themselves, which not only enables stronger expressive ability but also gives possibilities for knowledge transfer.

One previous work close to ours is (Gretton, 2007) that also trains the parameterised rule-based policy using policy gradient. An approach was proposed to pre-construct a set of potential policies in a brutal force manner and train the weights assigned to them using policy gradient. In contrast, in our work weights are not assigned directly to the whole policy and the parameters to be trained are involved in the deduction process whose number is significantly smaller than the enumeration of all policies, especially for larger problems. This gives our method better scalability. In addition, in (Gretton, 2007), expert domain knowledge is needed to specify the potential rules for the exact task that the agent is dealing with. However, in our work, we stick to use the same rules templates for all tasks we test on, which means all the potential rules have the same format across tasks.

A recent work on the topic (Zambaldi et al., 2018) proposes deep reinforcement learning with relational inductive bias that applies neural network mixed with self-attention to reinforcement learning tasks and achieves the state-of-the-art performance on the StarCraftII mini-games. The proposed methods show some level of generalization ability on the constructed block world problems and StarCraft mini-games, showing the potential of relation inductive bias in larger problems. However, as a graph-based rela-

tional model was used (Zambaldi et al., 2018), the learned policy is not fully explainable and the rules expression is limited, different from the interpretable logic-represented policies learned in ours using DILP.

3. Preliminary

In this section, we give a brief introduction to the necessary background knowledge of the proposed NLRL framework. Basic concepts of the first-order logic are first introduced. ∂ILP, a DILP model that our work is based on, is then described. The Markov Decision Process (MDP) and reinforcement learning are also briefly introduced.

3.1. First-Order Logic Programming

Logic programming languages are a class of programming languages using logic rules rather than imperative commands. One of the most famous logic programming languages is ProLog, which expresses rules using the first-order logic. In this paper, we use the subset of ProLog, i.e., DataLog (Getoor & Taskar, 2007).

Predicate names (or for short, predicates), **constants** and **variables** are three primitives in DataLog. In the language of relational learning, a predicate name is also called a relation name, and a constant is also termed as an entity (Getoor & Taskar, 2007). An **atom** α is a predicate followed by a tuple $p(t_1, ..., t_n)$, where p is a n-ary predicate and $t_1, ..., t_n$ are **terms**, either variables or constants. For example, in the atom *father(cart, Y)*, *father* is the predicate name, *cart* is a constant and *Y* is a variable. If all terms in an atom are constants, this atom is called a **ground atom**. We denote the set of all ground atoms as G. A predicate can be defined by a set of ground atoms, in which case the predicate is called an **extensional predicate**. Another way to define a predicate is to use a set of **clauses**. A clause is a rule in the form $\alpha \leftarrow \alpha_1, ..., \alpha_n$, where α is the **head** atom and $\alpha_1, ..., \alpha_n$ are body atoms. The predicates defined by rules are termed as **intensional predicates**.

3.2. ∂ILP

Inductive logic programming (ILP) is a task to find a definition (set of clauses) of some intensional predicates, given some positive examples and negative examples (Getoor & Taskar, 2007). The attempts that combine ILP with differentiable programming are presented in (Evans & Grefenstette, 2018; Rocktäschel & Riedel, 2017) and ∂ILP (Evans & Grefenstette, 2018) is introduced here that our work is based on.

The major component of ∂ILP operates on the valuation vectors whose space is $[0, 1]^{|G|}$, where each element of a valuation vector represents the confidence that a related ground atom is true. The logical deduction of each step of the ∂ILP is applied to the valuation vector. The new facts are derived from the facts provided by the valuation vector in the last step. For each predicate, ∂ILP generates a series of potential clauses combinations in advance based on rules templates. Trainable weights are assigned to clauses combinations, and the sum of weights for a predicate is constrained to be summed up to 1 using a softmax function.

With the differentiable deduction, the system can be trained with gradient-based methods. The loss value is defined as the cross-entropy between the output confidence of atoms and the labels. Compared with traditional inductive logic programming methods, ∂ILP has advantages in terms of robustness against noise/uncertainty and ability to deal with fuzzy data (Evans & Grefenstette, 2018).

4. Neural Logic Reinforcement Learning

In this section, the details of the proposed NLRL framework are presented. A new DILP architecture termed as Differentiable Recurrent Logic Machine (DRLM), an improved version of ∂ILP, is first introduced. The MDP with logic interpretation is then proposed to train the DILP architecture.

4.1. Differentiable Recurrent Logic Machine

Recall that ∂ILP operates on the valuation vectors whose space is $E = [0, 1]^{|G|}$, each element of which represents the confidence that a related ground atom is true. A DRLM is a mapping $f_{\boldsymbol{\theta}} : E \to E$, which performs the deduction of the facts $\boldsymbol{e}_0$ using weights $\boldsymbol{w}$ associated with possible clauses. $f_{\boldsymbol{\theta}}$ can then be decomposed into repeated application of single step deduction functions $g_{\boldsymbol{\theta}}$, namely,

$$f^t_{\boldsymbol{\theta}}(\boldsymbol{e}_0) = \begin{cases} g_{\boldsymbol{\theta}}(f^{t-1}_{\boldsymbol{\theta}}(\boldsymbol{e}_0)), & \text{if } t > 0. \\ \boldsymbol{e}_0, & \text{if } t = 0. \end{cases}, \tag{1}$$

where t is the deduction step. $g_{\boldsymbol{\theta}}$ implements one step deduction of all the possible clauses weighted by their confidences. We denote the probabilistic sum as $\oplus$ and

$$(\boldsymbol{a} \oplus \boldsymbol{b})_i = a_i + b_i - a_i b_i, \tag{2}$$

where $\boldsymbol{a} \in E, \boldsymbol{b} \in E$. $g_{\boldsymbol{\theta}}$ can then be expressed as

$$g_{\boldsymbol{\theta}}(\boldsymbol{e}) = \left(\bigoplus_n \sum_j w_{n,j} h_{n,j}(\boldsymbol{e}) \right) + \boldsymbol{e}_0, \tag{3}$$

where $h_{n,j}(\boldsymbol{e})$ implements one-step deduction using jth possible definition of nth clause.[1] For every single clause

[1] Computational optimization is to replace $\oplus$ with typical + when combining valuations of two different predicates. For further details on the computation of $h_{n,j}(\boldsymbol{e})$ (F_c in the original paper), readers are referred to Section 4.5 in (Evans & Grefenstette, 2018).

c, we can constraint the sum of its weights to be 1 by letting $\boldsymbol{w}_c = \text{softmax}(\boldsymbol{\theta}_c)$, where $\boldsymbol{w}_c$ is the vector of weights associated with the predicate c and $\boldsymbol{\theta}_c$ are related parameters to be trained.

Compared to ∂ILP, in DRLM the number of clauses used to define a predicate is more flexible; it needs less memory to construct a model (less than 10 GB in all our experiments); it also enables learning longer logic chaining of different intensional predicates. All these benefits make the architecture be able to work in larger problems. Detailed discussions on the modifications and their effects can be found in the appendix.

4.2. Markov Decision Process with Logic Interpretation

In this section, we present a formulation of MDPs with logic interpretation and show how to solve the MDP with the combination of policy gradient and DILP.

An MDP with logic interpretation is a triple (M, p_S, p_A):

- $M = (S, A, T, R)$ is a finite-horizon MDP;
- $p_S : S \to 2^G$ is the state interpretation that maps each state to a set of atoms including both information of the current state and background knowledge;
- $p_A : [0,1]^{|D|} \to [0,1]^{|A|}$ is the action interpretation that maps the valuation (or score) of a set of atoms D to the probability of actions.

For a DILP system $f_{\boldsymbol{\theta}} : 2^G \to [0,1]^{|D|}$, the policy $\pi : S \to [0,1]^{|D|}$ can be expressed as $\pi(s) = p_A(f_{\boldsymbol{\theta}}(p_S(s)))$. Thus any policy-gradient methods applied to DRL can also work for DILP. The p_S and p_A can either be hand-crafted or represented by neural architectures. The action selection mechanism in this work is to add a set of action predicates into the architecture, which depends on the valuation of these action atoms. Therefore, the action atoms should be a subset of D. As for ∂ILP, valuations of all the atoms will be deduced, i.e., $D = G$. If p_S and p_A are neural architectures, they can be trained together with the DILP architectures. p_S extracts entities and their relations from the raw sensory data. In addition, the use of a neural network to represent p_A enables agents to make decisions in a more flexible manner. For instance, the output actions can be deterministic and the final choice of action may depend on more atoms rather than only action atoms if the optimal policy cannot be easily expressed as first-order logic. For simplicity, in this work, we will only use the hand-crafted p_S and p_A. Notably, p_A is required to be differentiable so that we can train the system with policy gradient methods operating on discrete, stochastic action spaces, such as vanilla policy gradient (Willia, 1992), A3C (Mnih et al., 2016), TRPO(Schulman et al., 2015a) or PPO (Schulman et al., 2017). We will use the following schema to represent the p_A in all experiments. Let $p_A(a|\boldsymbol{e})$ be the probability of choosing action a given the valuations $\boldsymbol{e} \in [0,1]^{|D|}$. The probability of choosing an action a is proportional to its valuation if the sum of the valuation of all action atoms is larger than 1; otherwise, the difference between 1 and the total valuation will be evenly distributed to all actions, i.e.,

$$p_A(a|\boldsymbol{e}) = \begin{cases} \frac{l(e,a)}{\sigma}, & \sigma \geq 1 \\ l(e,a) + \frac{\sigma}{|A|}, & \sigma < 1 \end{cases} \tag{4}$$

where $l : [0,1]^{|D|} \times A \to [0,1]$ maps from valuation vector and action to the valuation of that action atom, σ is the sum of all action valuations $\sigma = \sum_{a'} p_A(a'|\boldsymbol{e})$. Empirically, this design is crucial for inducing an interpretable and generalizable policy. If we replace it with a trivial normalization, it is not necessary for NLRL agent to increase rule weights to 1 for sake of exploitation. The agent instead only need to keep the relative valuation advantages of desired actions over other actions, which in practice leads to tricky policies. We will train all the agents with vanilla policy gradient (Willia, 1992) in this work.

5. Experiments and Analysis

In general, the experiment is going to act as empirical investigations of the following hypothesis:

1. NLRL can learn policies that are comparable to neural networks in terms of expected return;
2. To induce these policies, we only need to inject minimal background knowledge;
3. The induced policies are explainable;
4. The induced policies can generalize to environments that are different from the training environments in terms of scale or initial state.

Four sets of experiments, which are cliff-walking, *STACK*, *UNSTACK* and *ON*, have been conducted and the benchmark model is a fully-connected neural network. The induced policy will be evaluated in terms of expected returns, generalizability and interpretability.

5.1. Experiment Setup

In the experiments, to test the robustness of the proposed NLRL framework, we only provide minimal atoms describing the background and states while the auxiliary predicates are not provided. The agent must learn auxiliary invented predicates by themselves as well, together with the action predicates.

5.1.1. BLOCK MANIPULATION

In this environment, the agent will learn how to stack the blocks into certain styles, that are widely used as a benchmark problem in the relational reinforcement learning research. We examine the performance of the agent on three subtasks: *STACK*, *UNSTACK* and *ON*. In the *STACK* task, the agent needs stack the scattered blocks into a single column. In the *UNSTACK* task, the agent needs to do the opposite operation, i.e., spread the blocks on the floor. In the *ON* task, it is required to put a specific block onto another one. In all three tasks, the agent can only move the topmost block in a pile of blocks. When the agent finishes its goal it will get a reward of 1. Before that, the agent keeps receiving a small penalty of -0.02. We will terminate the training if the agent didn't reach the goal within 50 steps.

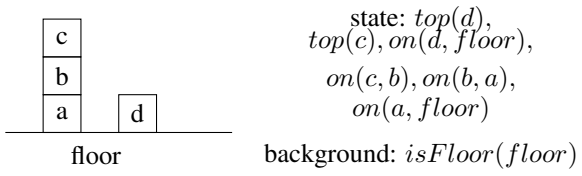


Figure 1. A Blocks Manipulation state noted as $((a, b, c), (d))$.

There are 5 different entities, 4 blocks labeled as *a, b, c, d* and *floor*. The state predicates are *on(X,Y)* and *top(X)*. *on(X,Y)* means the block *X* is on the entity *Y* (either blocks or floor). $top(X)$ means the block X is on top of an column of blocks. Notably, $top(X)$ cannot be expressed using on here as in DataLog there is no expression of negation, i.e., it cannot have "$top(X)$ means there is no $on(Y, X)$ for all Y". For all tasks, a common background knowledge is *isFloor(floor)*, and for the *ON* task, there is one more background knowledge predicate *goalOn(a,b)*, which indicates the target is to move block a onto the block b. The action predicate is *move(X,Y)* and there are 25 actions atoms in this task. The action is valid only if both Y and X are on the top of a pile or Y is $floor$ and X is on the top of a pile. If the agent chooses an invalid action, e.g., *move(floor, a)*, the action will not make any changes to the state. We use a tuple of tuples to represent the states, where each inner tuple represents a column of blocks, from bottom to top. For instance, Figure 1 shows the state $((a, b, c), (d))$ and its logic representation.

The training environment of the *UNSTACK* task starts from a single column of blocks $((a, b, c, d))$. To test the generalizability of the induced policy, we construct the test environment by modifying its initial state by swapping the top 2 blocks or dividing the blocks into 2 columns. The agent is also tested in the environments with more blocks stacking in one column. Therefore, the initial states of all the generalization test of *UNSTACK* are: $((a, b, d, c))$, $((a, b), (c, d))$, $((a, b, c, d, e))$, $((a, b, c, d, e, f))$ and $((a, b, c, d, e, f, g))$. For the *STACK* task, the initial state is $((a), (b), (c), (d))$ in training environment. Similar to the *UNSTACK* task, we swap the right two blocks, divide them into 2 columns and increase the number of blocks as generalization tests. The initial states of all the generalization test of *STACK* are: $((a), (b), (d), (c))$, $((a, b), (d, c))$, $((a), (b), (c), (d), (e))$, $((a), (b), (c), (d), (e), (f))$, $((a), (b), (c), (d), (e), (f), (g))$. For *ON*, the initial state is $((a, b, c, d))$. We swap either the top two or middle two blocks in this case, and also increase the total number of blocks. The initial states of all the generalization test of *ON* are thus: $((a, b, d, c))$, $((a, c, b, d))$, $((a, b, c, d, e))$, $((a, b, c, d, e, f))$ and $((a, b, c, d, e, f, g))$.

5.1.2. CLIFF-WALKING

Cliff-walking is a commonly used toy task for reinforcement learning. We modify the version in (Sutton & Barto, 1998) to a 5 by 5 field, as shown in Figure 2. When the agent reaches the cliff position it gets a reward of -1, and if the agent arrives the goal position, it gets a reward of 1. Before reaching these absorbing positions, the agent keeps receiving a small penalty of -0.02, encouraged to reach the goal as soon as possible. If the agent fails to reach the absorbing states within 50 steps, the game will be terminated. This problem can be modelled as a finite-horizon MDP.

The constants in this experiment are integers from 0 to 4. We inject basic knowledge about natural numbers including the smallest number (*zero(0)*), largest number (*last(4)*), and the order of the numbers (*succ(0,1), succ(1,2), ...*). The symbolic representation of the state is *current(X,Y)*, which specifies the current position of the agent. There are four action atoms *up(), down(), left(), right()*.

In the training environment of cliff-walking, the agent starts from the bottom left corner, labelled as S in Figure 2. In the generalization test, we first move the initial position to the top right, top left and centre of the field, labelled as S_1, S_2, S_3 respectively. Then we increase the size of the whole field to 6 by 6 and 7 by 7 without retraining.

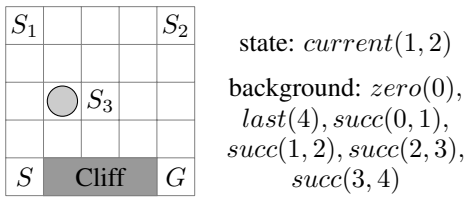


Figure 2. Cliff-walking, circle represents location of the agent

5.1.3. HYPERPARAMETERS

Similar to ∂ILP, we use RMSProp to train the agent, whose learning rate is set as 0.001. The generalized advantages ($\lambda = 0.95$) (Schulman et al., 2015b) are applied to the value network where the value is estimated by a neural network with one 20-units hidden layer.

Just like the architecture design of the neural network, the rules templates are important hyperparameters for the DILP algorithms. The rules template of a clause indicates the arity of the predicate (can be 0, 1, or 2) and the number of existential variables (usually pick from $\{0, 1, 2\}$). It also specifies whether the body of a clause can contain other invented predicates. We represent a rule template as a tuple of its three parameters, such as $(2, 1, True)$, for the simplicity of expression. The rules templates of the DRLM are quite general and the optimal setting and can be searched automatically. In this work, however, we stick the same rules templates for invented predicates across all the tasks, each with only 1 clause, i.e., $(1, 1, True), (1, 2, False), (2, 1, True), (2, 1, True)$. The templates of action predicates vary in different tasks but it is easy to find a good one by exhaustive search, therefore, little domain knowledge is needed. For *UNSTACK* and *STACK* task, the action predicate template is $(2, 1, True)$. The template of *ON* allows the action predicate defines two clauses that are specified as $(2, 1, True), (2, 0, True)$. There are four action predicates in the cliff-walking task, we give all these predicates the same template $(3, 1, True)$.

5.1.4. BENCHMARK NEURAL NETWORK AGENT

In all the tasks, we use a DRL agent as one of the benchmarks that have two hidden layers with 20 units and 10 units respectively. All the units in hidden layer use a ReLU (Nair & Hinton, 2010) activation function. For the cliff-walking task, the input is the coordinates of the current position of the agent. For block stacking tasks, the input is a $7 \times 7 \times 7$ tensor X, where $X_{x,y,i} = 1$ if the block indexed as i is in position x, y. We set each dimension of the tensor as 7 that is the maximum number of blocks used in the generalization test.

5.2. Results and Analysis

The performance of policy deduced by NLRL is stable against different random seeds once all the hyperparameters are fixed, therefore, we only present the evaluation results of the policy trained in the first run for NLRL here. For the neural network agent, we pick the agent that performs best in the training environment out of 5 runs.

5.2.1. PERFORMANCE AND GENERALIZATION TEST

The NLRL agent succeeds to find near-optimal policies on all the tasks. For generalization tests, we apply the learned policies on similar tasks, either with different initial states or problem sizes. The neural network agents and random agents are used as benchmarks.

We present the average and standard deviation of 500 repeats of evaluations in different environments in Figure 3 and Table 1. The highest average return of the three agents are marked in bold in each row of the table and the optimal performance of each task is also given. Each left group of bars in Figure 3 shows that the NLRL not only achieve a near-optimal performance in all the training environments but also successfully adapts to all new environments we designed in all the experiments. In most generalization tests, the agents manage to keep the performance in the near optimal level even if they never experience these new environments before. For instance, we can observe in Table 1 that in the *UNSTACK* task the NLRL agent achieves 0.937 average return, close to the optimal policy, and can achieve 0.94 final return. The minor difference between induced policy and the optimal one is caused by the stochasticity of the induced rules since the rule confidence is close but not exactly 1, which will be seen in the rules interpretations. When the top 2 blocks are swapped, the performance of NLRL agent is not affected. When the initial blocks are divided into 2 columns, it can still achieve 0.958 average return, very close to the optimal performance (0.96). The increase in the number of blocks gradually brings a larger difference between the return of the induced policy and the optimal one, whereas the difference is still less than 0.02.

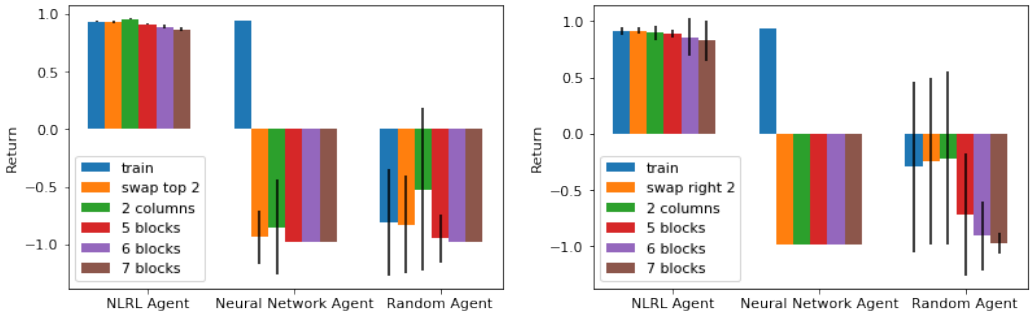
The neural network agents learn optimal policy in the training environment of 3 block manipulation tasks and learn near-optimal policy in cliff-walking. However, the neural network agent seems only remembers the best routes in the training environment rather than learns the general approaches to solving the problems. The overwhelming trend is, in varied environments, the neural networks perform even worse than a random player.

5.2.2. INTERPRET THE POLICIES

UNSTACK induced policy: The policy induced by NLRL in *UNSTACK* task is:

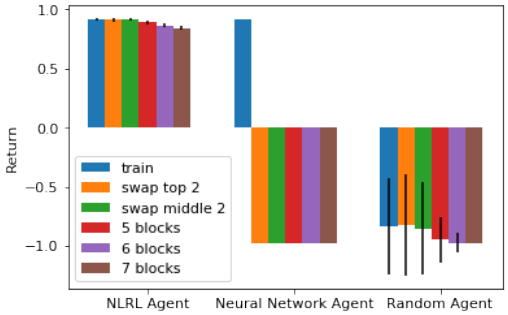
$$\begin{aligned} &0.972 : move(X, Y) \leftarrow isFloor(Y), pred(X) \\ &0.987 : pred(X) \leftarrow pred2(X), top(X) \\ &0.997 : pred2(X) \leftarrow on(X, Y), on(Y, Z) \end{aligned} \tag{5}$$

We only show the invented predicates that are used by the action predicate and the definition clause with high confidence (larger than 0.3) here. The $pred2(X)$ means the block X is on top of another block (the block is not directly

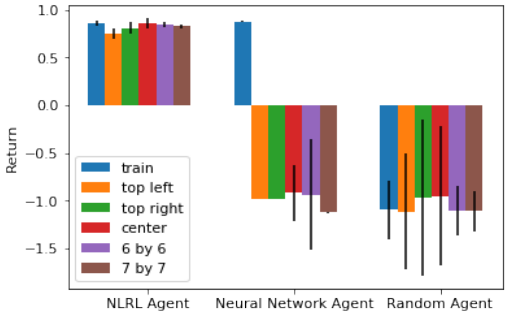


(a) *UNSTACK*

(b) *STACK*



(c) *ON*



(d) Cliffwalking

Figure 3. Performance on Train and Test Environments. Each sub-figure shows the performance of the three agents in a taks. The performance of each agent is divided into a group. In each group, the blue bar shows the performance in the training environment while other show the performance in the test environments.

Table 1. Performance on Train and Test Environments. The first three columns demonstrate the return of the three agents. The last column shows the return of the optimal policy.

		NLRL	NN	Random	Optimal
UNSTACK	train	0.937 ± 0.008	$\mathbf{0.94} \pm 0.0$	-0.807 ± 0.466	0.94
	swap top 2	$\mathbf{0.936} \pm 0.009$	-0.94 ± 0.232	-0.827 ± 0.428	0.94
	2 columns	$\mathbf{0.958} \pm 0.006$	-0.852 ± 0.414	-0.522 ± 0.71	0.96
	5 blocks	$\mathbf{0.915} \pm 0.01$	-0.98 ± 0.0	-0.948 ± 0.208	0.92
	6 blocks	$\mathbf{0.891} \pm 0.014$	-0.98 ± 0.0	-0.98 ± 0.0	0.9
	7 blocks	$\mathbf{0.868} \pm 0.016$	-0.98 ± 0.0	-0.98 ± 0.0	0.88
STACK	train	0.91 ± 0.033	$\mathbf{0.94} \pm 0.0$	-0.292 ± 0.759	0.94
	swap right 2	$\mathbf{0.913} \pm 0.029$	-0.98 ± 0.0	-0.24 ± 0.739	0.94
	2 columns	$\mathbf{0.897} \pm 0.064$	-0.98 ± 0.0	-0.215 ± 0.772	0.94
	5 blocks	$\mathbf{0.891} \pm 0.032$	-0.98 ± 0.0	-0.718 ± 0.542	0.92
	6 blocks	$\mathbf{0.856} \pm 0.169$	-0.98 ± 0.0	-0.905 ± 0.307	0.90
	7 blocks	$\mathbf{0.828} \pm 0.179$	-0.98 ± 0.0	-0.973 ± 0.097	0.88
ON	train	0.915 ± 0.01	$\mathbf{0.92} \pm 0.0$	-0.837 ± 0.405	0.92
	swap top 2	$\mathbf{0.912} \pm 0.013$	-0.98 ± 0.0	-0.821 ± 0.432	0.92
	swap middle 2	$\mathbf{0.914} \pm 0.011$	-0.98 ± 0.0	-0.853 ± 0.394	0.92
	5 blocks	$\mathbf{0.89} \pm 0.016$	-0.98 ± 0.0	-0.949 ± 0.195	0.90
	6 blocks	$\mathbf{0.865} \pm 0.018$	-0.98 ± 0.0	-0.975 ± 0.081	0.88
	7 blocks	$\mathbf{0.844} \pm 0.017$	-0.98 ± 0.0	-0.98 ± 0.0	0.86
Cliff-walking	train	0.862 ± 0.026	$\mathbf{0.877} \pm 0.008$	-1.096 ± 0.307	0.88
	top left	$\mathbf{0.749} \pm 0.057$	-0.98 ± 0.0	-1.115 ± 0.606	0.84
	top right	$\mathbf{0.809} \pm 0.064$	-0.98 ± 0.0	-0.966 ± 0.817	0.92
	center	$\mathbf{0.859} \pm 0.05$	-0.917 ± 0.296	-0.952 ± 0.73	0.92
	6 by 6	$\mathbf{0.841} \pm 0.024$	-0.934 ± 0.578	-1.101 ± 0.26	0.86
	7 by 7	$\mathbf{0.824} \pm 0.024$	-1.122 ± 0.006	-1.107 ± 0.209	0.84

on the floor). The $pred(X)$ means the block X is in the top position of a column of blocks and it is not directly on the floor, which basically indicates the block to be moved. The action predicate $move(X, Y)$ simply move the top block in any column with more than 1 block to the floor.

STACK induced policy: The policy induced by NLRL in *STACK* task is:

$$\begin{aligned}
&0.903 : move(X,Y) \leftarrow pred3(Y), pred4(X,Y)\\
&0.923 : pred4(X,Y) \leftarrow pred2(X), pred(Y,X)\\
&0.964 : pred(X,Y) \leftarrow on(X,Z), top(Y)\\
&0.970 : pred2(X) \leftarrow on(X,Y), isFloor(Y)\\
&0.960 : pred3(X) \leftarrow on(X,Y), pred(Y,X)
\end{aligned} \tag{6}$$

The $pred2(X)$ means X is a block that directly on the floor. The $pred(X, Y)$ means X is a block and Y is the top block in a column, where no meaningful interpretation exists. We consider $pred$ here is just used to help other predicates to express longer statement. The $pred4(X, Y)$ means X is a block that directly on the floor and there is no other blocks above it, and Y is a block. The main functionality of $pred4$ is to label the block to be moved, therefore, this definition is not the most concise one. In principle, we just need $pred4(X, Y) \leftarrow pred2(X), top(X)$ but the pruning rule of ∂ILP prevent this definition when constructing potential definitions because the variable Y in the head atom does not appear in the body. The $pred3(X)$ has the same meaning of $pred$ in $UNSTACK$ task, as it labels the top block in a column that is at least two blocks in height, which in this tasks tells where the block on the floor should be moved to. The meaning of $move(X, Y)$ is then clear: it moves the movable blocks on the floor to the top of a column that is at least two blocks high.

ON induced policy: The induced policy of the *ON* task is:

$$\begin{aligned}
&1.000 : move(X,Y) \leftarrow top(X), pred(X,Y)\\
&1.000 : move(X,Y) \leftarrow top(X), goalOn(X,Y)\\
&0.947 : pred(X,Y) \leftarrow isFloor(Y), pred2(X)\\
&1.000 : pred2(X) \leftarrow on(X,Y), on(Y,Z)
\end{aligned} \tag{7}$$

The goal of *ON* is to move block a onto b, while in the training environment the block a is at the bottom of the whole column of blocks. The strategy NLRL agent learned is to first unstack all the blocks and then move a onto b. The first clause of move $move(X, Y) \leftarrow top(X), pred(X, Y)$ implements the unstack procedures, where the logics are similar to the *UNSTACK* task. The second clause $move(X, Y) \leftarrow top(X), goalOn(X, Y)$ tells if the block X is already movable (there is no blocks above), just move X on Y. This strategy can deal with most of the circumstances and is optimal in the training environment. Whereas, we can also construct non-optimal case where unstacking all the blocks are not necessary or if the block b is below the block a, e.g., $((b, c, a, d))$.

Cliff-walking induced policy: The policy induced in the cliff-walking experiment is:

$$\begin{aligned}
&0.990 : right() \leftarrow current(X,Y), succ(Z,Y)\\
&0.561 : down() \leftarrow pred(X), last(X)\\
&0.411 : down() \leftarrow current(X,Y), last(X)\\
&0.988 : pred(X) \leftarrow zero(Y), current(X,Z)\\
&0.653 : left() \leftarrow current(X,Y), succ(X,X)\\
&0.982 : up() \leftarrow current(X,Y), zero(Y)
\end{aligned} \tag{8}$$

We can see that the agent will move to right if the Y coordinate has a predecessor, i.e., it is larger than 0. The rules about going down is a bit complex in the sense it uses an invented predicate that is actually not necessary. The rules of going down it deduced can be simplified as $down() : -current(X, Y), last(X)$, which means the current position is in the rightmost edge. The clause associated to predicate $left()$ will never be met since there will not be a number if the successor of itself, which is sensible since we never want the agent to move left in this game. There are many other definitions with lower confidence which basically will never be activated. Finally, the agent will go upwards if it is at the bottom row of the whole field. In fact, the only position the agent need to move up in the optimal route is the bottom left corner, while it does not matter here because all other positions in the bottom row are absorbing states. Such a policy is a sub-optimal one because it has the chance to bump into the right wall of the field. Although such a flaw is not serious in the training environment, shifting the initial position of the agent to the top left or top right makes it deviate from the optimal obviously.

6. Conclusion and Future work

In this paper, we propose a novel reinforcement learning method named Neural Logic Reinforcement Learning (NLRL) that is compatible with policy gradient algorithms in deep reinforcement learning. Empirical evaluations show NLRL can learn near-optimal policies in training environments while having superior interpretability and generalizability. In the future work, we will investigate knowledge transfer in the NLRL framework that may be helpful when the optimal policy is quite complex and cannot be learned in one shot. Another direction is to use a hybrid architecture of DILP and neural networks, i.e., to replace p_S with neural networks thus the agent can make decisions based on raw sensory data.

Acknowledgements

We want to thank thank Tim Rocktäschel and Frans A. Oliehoek for the discussions about the project; thank the reviwers for the useful comments; and thank Neng Zhang for the proofreading of the paper.

References

Boutilier, C., Reiter, R., and Price, B. Symbolic dynamic programming for first-order mdps. In *International Joint Conference on Artificial Intelligence*, IJCAI'17, 2001.

Cohen, W. W., Yang, F., and Mazaitis, K. R. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint*, abs/1707.05390, 2017.

Driessens, K. and Džeroski, S. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57 (3):271–304, Dec 2004. ISSN 1573-0565.

Driessens, K. and Ramon, J. Relational instance based regression for relational reinforcement learning. In *International Conference on Machine Learning*, ICML'03, 2003.

Džeroski, S., De Raedt, L., and Driessens, K. Relational reinforcement learning. *Mach. Learn.*, 43(1-2):7–52, April 2001. ISSN 0885-6125.

Evans, R. and Grefenstette, E. Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research*, 61:1–64, Jan 2018. ISSN 1076-9757. doi: 10.1613/jair.5714.

Getoor, L. and Taskar, B. *Introduction to Statistical Relational Learning*. The MIT Press, 2007. ISBN 0262072882.

Gretton, C. Gradient-based relational reinforcement learning of temporally extended policies. In *International Conference on Automated Planning and Scheduling*, ICAPS'07, 2007.

Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. Generalizing plans to new environments in relational mdps. In *International Joint Conference on Artificial Intelligence*, IJCAI'03, 2003.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 0028-0836. doi: 10.1038/nature14236.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on International Conference on Machine Learning*, ICML'16, 2016.

Montavon, G., Samek, W., and Müller, K.-R. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *International Conference on International Conference on Machine Learning*, ICML'10, 2010.

Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*. 2017.

Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. Trust region policy optimization. In *International Conference on on Machine Learning*, ICML'15, 2015a.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint*, abs/1506.02438, 2015b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint*, abs/1707.06347, 2017.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. ISSN 14764687. doi: 10.1038/nature24270.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Willia, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 15730565. doi: 10.1023/A:1022672621406.

Wulfmeier, M., Posner, I., and Abbeel, P. Mutual alignment transfer learning. In *Conference on Robot Learning*, 2017.

Yoon, S., Fern, A., and Givan, R. Inductive policy selection for first-order mdps. In *Conference on Uncertainty in Artificial Intelligence*, 2002.

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. Relational Deep Reinforcement Learning. *arXiv preprint*, abs/1806.01830, June 2018.