INTERPRETABLE REINFORCEMENT LEARNING WITH NEURAL SYMBOLIC LOGIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent progress in deep reinforcement learning (DRL) can be largely attributed to the use of neural networks. However, this black-box approach fails to explain the learned policy in a human understandable way. To address this challenge and improve the transparency, we introduce symbolic logic into DRL and propose a Neural Symbolic Reinforcement Learning framework, in which states and actions are represented in an interpretable way using first-order logic. This framework features a relational reasoning module, which performs on task-level in Hierarchical Reinforcement Learning, enabling end-to-end learning with prior symbolic knowledge. Moreover, interpretability is enabled by extracting the logical rules learned by the reasoning module in a symbolic rule space, providing explainability on task level. Experimental results demonstrate better interpretability of subtasks, along with competing performance compared with existing approaches.

1 INTRODUCTION

In recent years, deep reinforcement learning (DRL) has achieved great success in sequential decision-making problems like playing Atari Games (Mnih & et al, 2015) or the game of Go (Silver et al., 2017). However, it is hard to apply DRL to practical problems due notably to its lack of interpretability. The ability to explain decisions is important in earning people's trust and developing a robust and responsible system, especially in the area like autonomous driving. Moreover, an interpretable system makes problems traceable and debugging of systems easier. Therefore, interpretability has attracted increasingly attention in the DRL community recently. Interpretability can be either intrinsic or post-hoc, depending on how it is obtained. In the latter case, the black-box model is explained after training by visualizing for instance t-SNE and saliency maps (Zahavy et al.) or attention masks (Shi et al.). In the former case, interpretability is entailed by the inherent transparent property of the model. Our current work falls in that category.

To improve the interpretability of DRL policies, we investigate an approach that represents states and actions using first-order logic (FOL) and makes decision via neural-logic reasoning. In this setting, interpretability is enabled by inspecting the FOL rules used in the action selection, which can easily be understood and examined by a human. A number of algorithms (Jiang & Luo, 2019; Dong et al., 2019; Payani & Fekri, 2020) involving FOL take advantage of neural networks to induce a policy that performs the action selection via approximate reasoning on existing symbolic states and possibly additional prior knowledge. In this context, an action is selected if after performing some reasoning steps, an action predicate becomes true. The rules used in a policy can be learned using a differentiable version of inductive logic programming (ILP) whose goal is to learn FOL rules to explain observed data. When a neural network implements the policy, it can be trained to learn the rules and perform reasoning over those rules by having forward chaining implemented in the neural network architecture. The main issues with those approaches are their potential high-memory requirements and their computational costs, which limit their applicability. Alternatively, Lyu et al. (2019) propose a hierarchical reinforcement learning (HRL) approach where a high-level (task level) policy selects tasks to be solved by low-level (action level) policies. The latter policies interacts directly with the environment through potential high-dimensional inputs, while the high-level policy makes decisions via classical planning. While this approach can scale to larger problems, it requires the specification by an expert of the planning problem of the high-level policy.

To alleviate the problems discussed above, in this paper, we propose a novel framework named Neural Symbolic Reinforcement Learning (NSRL). Similarly to (Lyu et al., 2019), it is an HRL framework. However, in NSRL, the high-level policy makes decisions via neuro-logic reasoning. Thus, in contrast to (Lyu et al., 2019), NSRL does not need the definition of any oracle rules or transition model in advance. In contrast to differentiable ILP methods, NSRL performs reasoning by selecting relational paths over the knowledge graph induced by the known predicates. To the best of our knowledge, this is the first work introducing reasoning into RL that can can succeed in complex domains while remaining interpretable. More specifically, this framework features a reasoning module based on neural attention network, which performs relational reasoning on symbolic states and induces the RL policy. We further embed this module on the task level in HRL and evaluate the framework on Montezuma's Revenge. Leveraging the power of symbolic representation, the results demonstrate competing performance with existing RL approaches while providing improved interpretability by extracting the most relevant relational paths.

2 Related Work

2.1 Inductive Logic Programming

Traditional inductive logic programming (ILP) approaches require the search in a discrete space of rules and are not robust to noise (Evans & Grefenstette, 2018). To address those issues, many recent works have proposed various differentiable versions of ILP (Evans & Grefenstette, 2018; Dong et al., 2019; Payani & Fekri, 2020). They are all based on simulating forward chaining and suffer from some form of scalability issues (Yang & Song, 2020). In contrast, multi-hop reasoning methods (Gardner & Mitchell, 2015; Das et al., 2017; Lao & Cohen, 2010; Yang & Song, 2020) allow answering queries involving two entities over a knowledge graph (KG) by searching a relational path between them. In the ILP context, such paths can be interpreted as grounded first order rules. Interestingly, they can be computed via matrix multiplication (Yang et al., 2017). Compared to differentiable ILP, multi-hop reasoning methods have demonstrated better scalability. Our work can be seen as the extension of the work by Yang & Song (2020) to the RL setting.

2.2 Interpretable Reinforcement Learning

Recent work on interpretable DRL can be classified into two types of approaches, focusing either on (i) intrinsic interpretability or (ii) post-hoc explanation. Intrinsic interpretability requires the learned model to be self-understandable by nature, which is achieved by using a transparent class of models, whereas post-hoc explanation entails learning a second model to explain an already-trained black-box model. In approach (i), a (more) interpretable policy can be learned directly online by considering a specific class of interpretable policies (e.g., (Lyu et al., 2019)), or by enforcing interpretability via architectural inductive bias (e.g., (Zambaldi et al., 2018), (Jiang & Luo, 2019; Dong et al., 2019)). Alternatively, an interpretable policy can be obtained from a trained one via imitation learning (e.g., (Bastani et al., 2018; Vasic et al., 2019), or (Verma et al., 2018; Verma et al.)). In approach (ii), various techniques have been proposed to explain the decision-making of DRL agents (e.g., (Zahavy et al.; Greydanus et al., 2018; Gupta et al., 2020), (Shi et al.), (Sequeira & Gervasio), (Juozapaitis et al.), (Madumal et al., 2020), or (Topin & Veloso, 2019)). More related to interpretable policies, some work in approach (ii) also tries to obtain a more understandable policy (e.g., (Coppens et al.)) in order to explain a trained RL agent.

Both our framework and that of Lyu et al. (2019) are formulated in a hierarchical reinforcement learning (HRL) setting. However, in their work, the meta-controller (i.e., high-level policy) is a symbolic planner, while in ours, its action selection is realized by neural logic reasoning. Thus, their proposition requires an expert to describe a task as a symbolic planning problem, whereas our work only needs the definition of a high-level reward function, which is arguably easier to provide. In contrast to the extensions of differentiable ILP to RL (Jiang & Luo, 2019; Dong et al., 2019), our framework is formulated in an HRL setting and only the meta-controller is based on neural-logic reasoning, which is moreover realized via multi-hop reasoning. We believe that our architectural choices explain why our method can scale to a complex problem such as Montezuma's revenge, while preserving some level of interpretability.

3 PRELIMINARY

In this section, we give a brief introduction to the background knowledge necessary for the proposed framework. Interpretable rules described by First-Order Logic are first introduced, then the basics about Hierarchical Reinforcement Learning are briefly described.

3.1 FIRST ORDER LOGIC

A typical First-Order Logic(FOL) system consists of three components: **Entity, Predicate, Formula**. Entities are constants (e.g., objects) while a predicate can be seen as a relation between entities. An **atom** α=$p(t_1, t_2, .., t_n)$ is composed with a n-nary predicate p and n terms $\{t_1, t_2, ..., t_n\}$, where a term can be a constant or variable. An atom is grounded if all terms in this atom are constants. A **formula** is an expression formed with atoms, logical connectives, and possibly existential and universal quantifiers. In the context of ILP, one is interested to learn formulas of restricted forms called rules. A rule is called a **clause** in the form of $\alpha \leftarrow \alpha_1 \wedge \alpha_2, ..., \wedge\alpha_n$ where α is called head atom and $\alpha_1, \alpha_2, ..., \alpha_n$ are called body atoms. A clause is grounded with all the associated atoms grounded. The head atom is believed to be true only if all the body atoms are true. For example, $Connected(X, Z) \leftarrow Edge(X, Y) \wedge Edge(Y, Z)$ is a clause where X, Y, Z are variables and $Connected, Edge$ are predicates. If we substitute X, Y, Z with constants a, b, c, then a and c are considered connected if $Edge(a, b)$ and $Edge(b, c)$ hold. Embedded with prior knowledge, clauses described by FOL is highly understandable and interpretable.

3.2 HIERARCHICAL REINFORCEMENT LEARNING

Consider a Markov Decision Process defined by a tuple $(S, A, P^a_{ss'}, r^a_s, \gamma)$ where S and A denote the state space and action space respectively, $P^a_{ss'}$ provides the transition probability of moving from state $s \in S$ to state $s' \in S$ after taking action $a \in A$, r^a_s is the immediate reward obtained after performing action a in state s and $\gamma \in [0, 1]$ is a discount factor. The objective of an RL algorithm is to find a policy $\pi : S \rightarrow A$ that maximizes the expected return $V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{1+t} | s_0 = s]$ where r_t is the reward at time step t received by following π from state s_0. In Hierarchical Reinforcement Learning (HRL), the agent learns the high-level policy (meta-controller) and low-level policy (controller) jointly when interacting with the environment. Sutton et al. (1999) formalize the idea of option as temporally extended actions. In a two-level structure, an option chosen by the high-level policy is achieved by the lower-level policy. Markov property exists at different levels. Thus, we can utilize standard RL algorithms to learn policy separately for each level.

4 PROPOSED METHOD

In this section, we firstly illustrate the whole structure of Neural Symbolic Reinforcement Learning (NSRL) and how it performs on the task level in Hierarchical Reinforcement Learning and demonstrate the cross-fertilization of three components of NSRL. Then, we separately introduce these three components : reasoning module, attention module and the policy module. In the end of this section, we present the training process of NSRL.

4.1 NEURAL SYMBOLIC REINFORCEMENT LEARNING

Hierarchical Reinforcement Learning (HRL) is an effective method to solve sparse and delayed reward by integrating temporal abstraction and intrinsic motivation. However, it still suffers from the lack of interpretability of each abstracted level. In this section, we abstract the Markov Decision Making Process into two levels: task-level and action level. Then, we put forward the framework of Neural Symbolic Logic based HRL, providing a high level insight on the decision-making process. For convenience, we reuse the notation of standard RL. We use $(\hat{S}, \hat{A}, \hat{P}^{\hat{a}}_{\hat{s}\hat{s}'}, \hat{r}^a_s, \hat{\gamma})$ to denote the higher level where $\hat{S}$ represents the symbolic state. $(S, A, P^a_{ss'}, r^a_s, \gamma)$ denotes the lower level where S represents the high dimensional state.

As shown in Figure 1, the left part illustrates the whole training procedure of the algorithms. Firstly, a detector of objects and relations extracts objects from high-dimensional state s and outputs a symbolic state $\hat{s}$ to the meta controller, which allocates tasks and provides intrinsic rewards to the

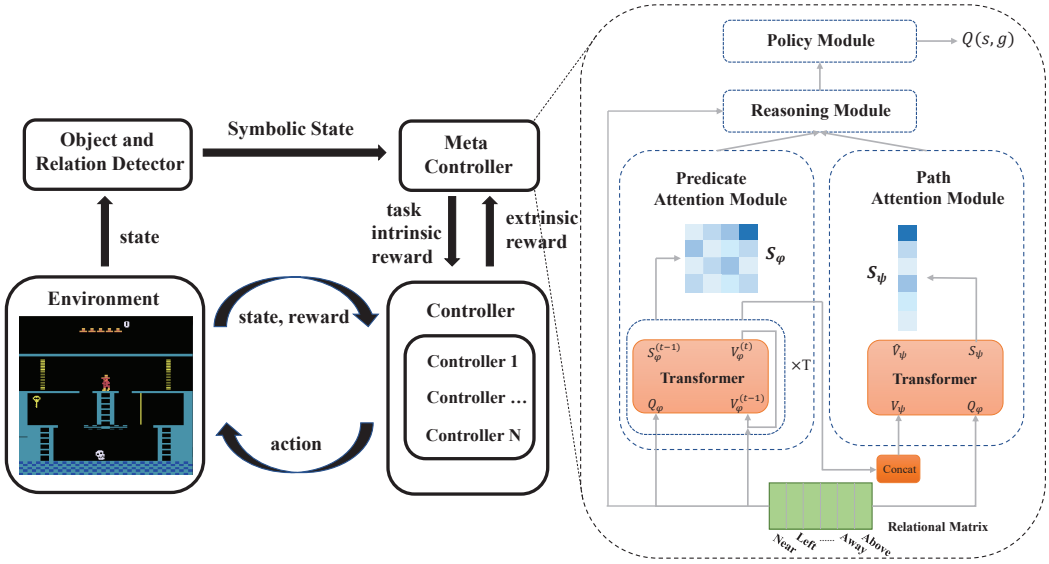


Figure 1: Architecture illustration

controller. Then, the controller interacts with the environment iteratively until achieving the task or reaching maximum steps. At the same time, the controller collects extrinsic reward to the meta controller for training. Instead, intrinsic reward is used to train the controller.

The right part depicts the architecture of the meta controller or NSRL. There are three major components: reasoning module, attention module and the policy module, which are detailly described in Section 4.2, Section 4.3, Section 4.4 separately. The symbolic state $\hat{s}$ would be firstly transformed to relational matrix, which is sent to two attention modules to generate the attention weights on predicate and length of relational paths. Afterwards, the relational matrix, as well as the attention weights would be sent into reasoning module, performing reasoning on existing symbolic knowledge. The reasoning module can take consideration of all predicates at one reasoning step and possible relational paths of varying length. After that, the policy module would receive the reasoning output and compute the Q value of action atoms, thus induce the DRL policy.

4.2 Reasoning Module

Consider a knowledge graph, where objects are represented as nodes and relations are edges. Multi-hop reasoning on such a graph mainly focuses on searching chain-like logical rules of the following form,

$$query(x', x) \leftarrow R_1(x', z_1) \wedge \cdots \wedge R_n(z_n, x) \tag{1}$$

The task of multi-hop reasoning for a given query corresponds to finding find a relational path from x to x' with multi-steps $x \xrightarrow{R_1} \cdots \xrightarrow{R_n} x'$ with given query. Based on Yang et al. (2017), the inference of this logical path could be seen as a process of matrix multiplication. Every predicate or relation P_k is represented as a binary matrix $\boldsymbol{M}_k$ in $\{0,1\}^{|\mathcal{X}|\times|\mathcal{X}|}$, whose entry (i,j) of $\boldsymbol{M}_k$ is 1 if $P_k(x_i, x_j)$ is in the knowledge graph with entity x_i and x_j are connected by edge P_k. $\mathcal{X}$ is the total numbers of objects. Let $\boldsymbol{v}_x$ be the one-hot encoding of object x. Then, the $t-$th hop of the reasoning along the path can be computed as

$$\boldsymbol{v}^{(0)} = \boldsymbol{v}_x \tag{2}$$

$$\boldsymbol{v}^{(t)} = \boldsymbol{M}^{(t)}\boldsymbol{v}^{(t-1)} \tag{3}$$

where $\boldsymbol{M}^{(t)}$ is the matrix used in $t-$th hop and $\boldsymbol{v}^{(t-1)}$ is the path features vector. After T steps reasoning, the score of the query is computed as

$$\text{score}(\text{x}, \text{x}') = \boldsymbol{v}_x{}^\intercal \prod_{t=1}^{T} \boldsymbol{M}^{(t)} \cdot \boldsymbol{v}_{x'} \tag{4}$$

Considering all the predicate matrices at each step and relational paths of different lengths, the final score could be rewritten with soft attention as below:

$$\kappa(s_\psi, S_\varphi) = \sum_{t'=1}^{T} s_\psi^{(t')} \left(\prod_{t=1}^{t'} \sum_{k=1}^{K} s_{\varphi,k}^{(t)} \boldsymbol{M}_k \right) \tag{5}$$

$$\text{score}(\text{x}, \text{x}') = \boldsymbol{v}_x{}^\intercal \kappa(s_\psi, S_\varphi) \boldsymbol{v}_{x'} \tag{6}$$

where T is the maximum reasoning steps, $s_\psi^{t'}$ corresponds to attention weights on all the relational paths of length t' and $s_{\varphi,k}^{(t)}$ to another attention weights on predicate matrix $\boldsymbol{M}_k$ used in the t-th step, and K denotes the total number of predefined predicates.

4.3 Attention Module

In this section, we introduce the architecture of attention network, a hierarchical stack of transformers, to generate the dynamic attention weights. Consider a basic multi-head dot-product attention module (MHDPA) in transformer (Vaswani et al., 2017), the input of MHDPA is the query, key and value representations: Q, K, V. MHDPA firstly computes the similarity or attention weights S between the query and the key, and then calculates the weighted value as output V':

$$S = \text{softmax}\left(\frac{QK^\intercal}{\sqrt{d}}\right) \text{ and } V' = SV \tag{7}$$

where d is the dimension of K. We utilize this module to generate the attention weights s_ψ and $s_{\varphi,k}$. In fact, the symbolic states can be represented as a 3-dimensional tensor $\boldsymbol{M} \in [0,1]^{|\mathcal{X}|\times|\mathcal{X}|\times N}$, where $\mathcal{X}$ denotes the numbers of extracted objects and N represents the numbers of predefined predicates.

We transform tensor $\boldsymbol{M}$ into a matrix $\boldsymbol{M}_f \in [0,1]^{N\times 2|\mathcal{X}|}$ at each time step. Each row of matrix $\boldsymbol{M}_f$ represents a part of the symbolic state, which can be seen as an embedding of predicate varying with time. In this way, the attention module can generate weights on predicates at different time steps, taking consideration of the symbolic information of current RL state. We firstly generate the query, key and value representation by multi-perception layers with $\boldsymbol{M}_f$ as input initially. For convenience, we introduce $V_\phi = \boldsymbol{M}_f$. Then, we repeatedly use the output value from last step to generate the attention weights. For predicate attention module.

$$Q_\varphi^{(t-1)}, K_\varphi^{(t-1)}, V_\varphi^{(t-1)} = \text{FeedForward}(\text{V}_\phi) \tag{8}$$

$$S_\varphi^{(t)}, V_\varphi^{(t)} = \text{MHDPA}(Q_\varphi^{(t-1)}, K_\varphi^{(t-1)}, V_\varphi^{(t-1)}), \quad V_\phi = V_\varphi^{(t)} \tag{9}$$

Here, $S_\varphi^{(t)}$ represents the attention weights on predicates in the t-th hop reasoning.

For path attention module, we reuse the output value of each time step in predicate attention module. During the iterative processing, the output value at each step embeds the information of paths of different lengths. We simply use another transformer to generate the path attention weights S_ψ. Let $V_\varphi = [V_\varphi^{(0)}, V_\varphi^{(1)}, \cdots, V_\varphi^{(t)}]^\intercal$.

$$Q_\varphi, K_\varphi, V_\varphi = \text{FeedForward}(\text{V}_\varphi) \tag{10}$$

$$S_\psi, V_\psi = \text{MHDPA}(Q_\varphi, V_\varphi) \tag{11}$$

4.4 POLICY MODULE

In this section, we put forward the policy module to induce DRL policy. Suppose there exists an entity set $\boldsymbol{S_e}$ and a predicate set $\boldsymbol{S_p}$ consisting of predefined predicates $\boldsymbol{P_b}$ and auxiliary predicates $\boldsymbol{P_a}$ similar to (Evans & Grefenstette, 2018). With the predicate and entity set, we can obtain symbolic state at each time step. The symbolic state can be represented by a 3-dimensional tensor $\boldsymbol{M}$ as discussed in Section 4.3. Since a policy is a mapping from states to actions, the predicate at the last hop needs to be constrained to be an action predicate. Instead of introducing a predicate matrix, we can instead introduce multi-perceptron layers to the output of reasoning to induce a policy.

$$Q_a(x, x') = (\boldsymbol{v}_x^\intercal \mathrm{MLP}_a(\kappa(s_\psi, S_\varphi))\boldsymbol{v}_{x'}) \tag{12}$$

Algorithm 1: Neural Symbolic Reinforcement Learning

Initialize Replay Buffer $D_m, D_{s1}, ..., D_{sj}$;
Initialize Attention Network $\mathcal{L}_m$ and Controller $\mathcal{L}_{s1}, \mathcal{L}_{s2}, ..., \mathcal{L}_{sj}$ respectively;
for $i = 1...num\_episodes$ **do**
  extract objects and obtain symbolic state $\hat{s}$;
  while *$\hat{s}$ is not Terminal and maximal step is not reached* **do**
    perfrom reasoning on symbolic state $\hat{s}$ based on (12);
    choose task $g \leftarrow \mathrm{EPSGreedy}(\mathrm{Q_m}, \hat{\mathrm{s}})$;
    obtain current state s;
    while *s is not Terminal and maximal step is not reached* **do**
      $a \leftarrow \mathrm{EPSGreedy}(\mathrm{Q_g}, \mathrm{s})$ where Q_g is the output of Controller $\mathcal{L}_g$;
      obtrain current state s' and reward r;
      store (s, a, s', r) in D_g;
      $s \leftarrow s'$;
      if *g is reached* **then**
        break
    obtain current symbolic state $\hat{s}'$ and reward $\hat{r}$;
    store $(\hat{s}, g, \hat{s}', \hat{r})$ in D_m;
    $\hat{s} \leftarrow \hat{s}'$;
  Update Q_j for $j = 1, 2, ..., num\_tasks$;
  Update Meta Controller Q_m

5 EXPERIMENTS

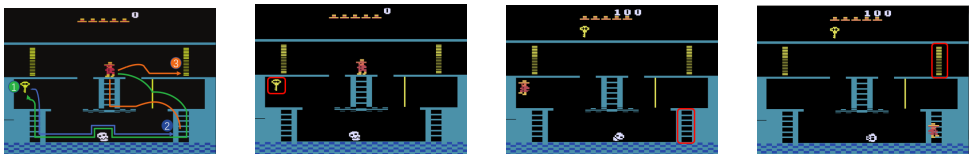
We conduct experiments on Montezuma's Revenge, an **ATARI** Game with sparse and delayed reward. The player is required to navigate through several rooms while collecting treasures. Our experiments are based on the first room. In this room, the player needs to first fetch the key, receiving a reward(+100) after taking a long sequence of actions without any reward information. Then, the player needs to navigate to the door (left or right) and pass through it, resulting in another reward(+300).

5.1 SYMBOLIC REPRESENTATION

Similar to HDQN, every task is associated with an extracted object. Logical representation is based on 8 predefined locations: middle laddder, top-right door, top-left door, bottom-left ladder, bottom-right ladder, conveyer, rope and the key and 2 mobile objects : man and skull. We introduce predicates like **OnSpot, WithObject, SmallerX, SameX, SmallerY, SameY, Near, PathExist, Conditional**. The atom Conditional(x,y) means we need to fetch object x firstly and then reach object y. Firstly, we use logic primitive statements not on these predicates to generate other predicates like **LargerX, LargerY, Away**. Since some logical rules may combine the negation of relation of entities. It's easy to accomplish negation operation by fuzzy logic, which enables to extend the rule search space as described in Yang & Song (2020). We introduce one auxiliary and binary action predicate **MoveTo(agent,** x), which means the agent needs to reach the object x.

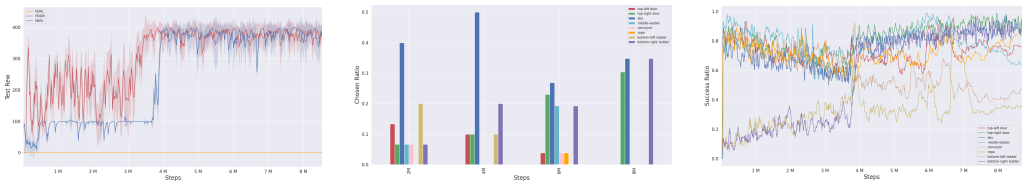
5.2 EXPERIMENTAL RESULTS

In this section, we analyze the experimental results. Fig2 illustrates a sample gameplay by our agent on Montezuma's Revenge. Fig 3 depicts the performance of algorithms as training proceeds. Finally, several relational paths learned by the agent are visualized to validate the interpretablity embedded in NSRL.



(a) Optimal Policy (b) Choose Key (c) Choose Right Ladder (d) Choose Right Door

Figure 2: Sample gameplay on Montezuma's Revenge : **(a)** illustrates the optimal policy learned by the agent. It firstly chooses the key as a task, then turn to reach the bottom-right ladder after obtaining the key. Finally, the agent chooses top-right door and takes a set of low-level actions to reach the door. **(b-d)** presents the process of navigating through the first room. It is easy to see the key, bottom-right ladder and the top-right door are the key tasks to choose.



(a) Learning Curve (b) Choosen ratio of different tasks over time (c) Success ratio of different tasks over time

Figure 3: Training performance on Montezuma's Revenge: **(a)** illustrates the training phase of NSRL and HDQN, of which the results are collected and averaged by 8 runs. **(b)** depicts the distribution of tasks chosen by the meta controller at different time steps. **(c)** presents the success ratio of tasks over time.

We compare our approach with HDQN (Kulkarni et al., 2016a), SDRL (Lyu et al., 2019) as baselines. Both HDQN and NSRL coverage to achieve +400 reward but SDRL fails. To learn the optimal policy, it nearly takes NSRL and HDQN another 4M steps. Different from HDQN, at the beginning of first 3M steps, NSRL fails to learn the way back to the door (+300 reward). Besides of this, the performance of NSRL is still competing compared to HDQN since they nearly start to coverage almost at the same time and the variance of NSRL is smaller than that of HDQN. Moreover, with embedded reasoning module, NSRL can provide more interpretability as discussed below. As shown in Fig.3b, all tasks are nearly preferred at the beginning of training expect the key. As training proceeds, the agent gradually learns to select the key tasks, the probabilities of which gradually increase. Specially, at 8M steps, the agent only chooses the key, bottom-right ladder and top-right door. We also visualize the relational paths to explain the important relations when choosing pivotal tasks as below. At each reasoning step, we consider the predicates with highest attention weights at each reasoning step and the connective constraints to formalize a specific path since we use softmax operation to consider all paths of different lengths. For convenience, we use RightDoor to represent top-right door, LeftDoor to represent top-left door, RightLadder to represent bottom-right ladder, LeftLadder to represent bottom-left ladder and Man to agent.

	Relational Paths
1	MoveTo(Man, Key) ← WithOutObject(Man, Key)
2	MoveTo(Man, Key) ← WithOutObject(Man, Key) ∧ SmallerY(Key, Man) ∧ LargerY(Man, Key)
3	MoveTo(Man, Key) ← WithOutObject(Man, Key) ∧ SmallerY(Key, LeftDoor) ∧ LargerY(LeftDoor,Key)
4	MoveTo(Man, Key) ← WithOutObject(Man, Key) ∧ SmallerY(Key, RightDoor) ∧ LargerY(RightDoor,Key)
5	MoveTo(Man, Key) ← WithOutObject(Man, Key) ∧ SmallerY(Key, RightDoor) ∧ LargerY(RightDoor,Man) ∧ LargerY(Man, Key)

Table 1: Choose Key

	Relational Paths
1	MoveTo(Man, RightLadder) ← WithObject(Man, Key) ∧ Conditional(Key, LeftDoor) ∧ LargerY(LeftDoor, Man) ∧ LargerY(Man, Conveyer) ∧ SmallerX(Conveyer, RightLadder)
2	MoveTo(Man, RightLadder) ← WithObject(Man, Key) ∧ Conditional(Key, LeftDoor) ∧ LargerY(LeftDoor, Man) ∧ LargerY(Man, LeftLadder) ∧ SmallerX(LeftLadder, RightLadder)
3	MoveTo(Man, RightLadder) ← WithObject(Man, Key) ∧ Conditional(Key, RightDoor) ∧ LargerY(RightDoor, Man) ∧ LargerY(Man, Conveyer) ∧ SmallerX(Conveyer, RightLadder)

Table 2: Choose RightLadder

	Relational Paths
1	MoveTo(Man, RightDoor) ← Near(Man, RightLadder) ∧ Away(RightLadder, RightDoor)
2	MoveTo(Man, RightDoor) ← Near(Man, RightLadder) ∧ Away(RightLadder, RightDoor) ∧ SameX(RightDoor, RightLadder) ∧ LargerX(RightLadder, Man) ∧ LargerX(Man, MiddleLadder) ∧ Near(MiddleLadder, Conveyer) ∧ Away(Conveyer, RightDoor)
3	MoveTo(Man, RightDoor) ← Near(Man, RightLadder) ∧ Away(RightLadder, RightDoor) ∧ SameX(RightDoor, RightLadder) ∧ LargerX(RightLadder, Rope) ∧ LargerX(Rope, Conveyer) ∧ Near(Conveyer, MiddleLadder) ∧ Away(MiddleLadder, RightDoor)

Table 3: Choose RightDoor

Tables 1-3 present the relational paths the agent puts most attention on when choosing different tasks. NSRL realizes the fact that the agent does not possess the key by putting highest attention on predicate **WithOutObject**. As shown in Table 2, we can see that NSRL pays close attention on predicate **WithObject** and **Conditional** at consecutive steps when choosing bottom-right ladder as a task, indicating that NSRL gradually learns the importance of possessing the key to the door. After that, the agent considers the relevant location information to choose bottom-right ladder. In Table 3, the agent considers more distance information when choosing RightDoor. It first recognizes the current location by atom Near(Man, RightLadder) and then considers the reachbility from RightLadder to RightDoor. All of these relational paths are a form of logical rules, providing interpretability on choosing tasks.

6 CONCLUSION

In this paper, we propose a novel framework performing neural-logic reasoning to enable interpretability by visualizing the relational paths to tasks. Exploiting multi-hop reasoning and hierarchical reinforcement learning, our approach can solve large-sized complex problem like Montezuma's Revenge, in contrast to other recent neuro-symbolic approaches. Compared to other black-box style methods, the learning process of our approach naturally integrates with symbolic knowledge while achieving comparable performance and preserving interpretability. The introduced symbolic predicates could easily be reused in other domains.

7 ACKNOWLEDGEMENT

REFERENCES

Osbert Bastani, Yewen Pu, and Armando Solarlezama. Verifiable reinforcement learning via policy extraction. *Neural Information Processing Systems*, pp. 2494–2504, 2018.

Youri Coppens, Kyriakos Efthymiadis, Tom Lenaerts, and Ann Now. Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, pp. 1–6. URL https://cris.vub.be/files/46718934/IJCAI_2019_XAI_WS_paper.pdf. 00000.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew Mccallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. *Conference of The European Chapter of The Association for Computational Linguistics*, 1:132–141, 2017.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Dengyong Zhou. Neural logic machines. *International Conference on Learning Representations*, 2019.

R Evans and E Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61(1):1–64, 2018.

Matt Gardner and Tom M Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1488–1498, 2015.

Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. *International Conference On Machine Learning*, pp. 1787–1796, 2018.

Piyush Gupta, Nikaash Puri, Sukriti Verma, Dhruv Kayastha, Shripad Deshmukh, Balaji Krishnamurthy, and Sameer Singh. Explain your move: Understanding agent actions using focused feature saliency. *International Conference on Learning Representations*, 2020.

Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. *International Conference on Machine Learning*, pp. 3110–3119, 2019.

Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*. URL https://web.engr.oregonstate.edu/~afern/papers/reward_decomposition__workshop_final.pdf. 00000.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Neural Information Processing Systems*, 2016a.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. *Neural Information Processing System*, pp. 3682–3690, 2016b.

Ni Lao and William W Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1), 2010.

Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Silver David Mnih, Kavukcuoglu and Rusu et al. Human-level control through deep reinforcement learning. *Nature*, 518(740):529–533, 2015.

Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. *arXiv: Learning*, 2019.

Ali Payani and Faramarz Fekri. Incorporating relational background knowledge into reinforcement learning via differentiable inductive logic programming, 2020.

Pedro Sequeira and Melinda Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. 288:103367. ISSN 00043702. doi: 10.1016/j.artint.2020.103367. URL http://arxiv.org/abs/1912.09007.

Wenjie Shi, Shiji Song, Zhuoyuan Wang, and Gao Huang. Self-supervised discovering of causal features: Towards interpretable reinforcement learning. URL http://arxiv.org/abs/2003.07069. 00000.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

Nicholay Topin and Manuela Veloso. Generation of policy-level explanations for reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):2514–2521, 2019.

Marko Vasic, Andrija Petrovic, Kaiyuan Wang, Mladen Nikolic, Rishabh Singh, and Sarfraz Khurshid. Moet: Interpretable and verifiable reinforcement learning via mixture of expert trees. *arXiv: Learning*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Neural Information Processing Systems*, pp. 5998–6008, 2017.

Abhinav Verma, Hoang M. Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. URL https://papers.nips.cc/paper/9705-imitation-projected-programmatic-reinforcement-learning.pdf. 00000.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *International Conference on machine Learning*, 2018.

Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. *In Advances in Neural Information Processing Systems*, pp. 2316–2325, 2017.

Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. *International Conference on Learning Representation*, 2020.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In Maria Florina Balcan and Kilian Q Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1899–1908. PMLR. URL http://proceedings.mlr.press/v48/zahavy16.html.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv: Learning*, 2018.

A REWARD ENGINEERING

To facilitate the learning process in HRL, we use the reward engineering as below. For action level,

$$r_i(s_l, a_l) = \begin{cases} \phi & \text{achieve tasks} \\ \psi & \text{terminal} \\ c & \text{step cost} \end{cases} \tag{13}$$

The controller of lower level receives cost c at every time step and achieves a large reward ϕ when finishing tasks or negative reward ψ when failing to complete the tasks within maximum number of steps . For task level, we further modify the extrinsic reward based on Hierarchical Deep Q-Learning Framework

$$r(s_h, g_h) = \begin{cases} -\phi & \beta = 0 \\ f(s_h, g_h) & \beta = 1 \end{cases} \tag{14}$$

whereβ is a flag indicating whether the agent achieved the assigned task $(\beta = 1)$ or not $(\beta = 0)$. ϕ is a large number for punishing the agent for not finishing the tasks and $f(s_h, g_h)$ is the cumulative reward from the environment by following the task g_h under state s_h. To this end, the agent is encouraged to learn and select the right order of these tasks.

B EXPERIMENTAL SETUP

For the controller architecture, we follow the one used in Kulkarni et al. (2016b). There are 8 controllers in total, each associated with a predefined location. In terms of the meta controller, the maximum length of relational path and the numbers of head of transformer are set to 8 and 4 respectively. The length of one episode is set to be 500 and the agent is restricted to finish the game with one life. Learning rates are all set to be 1e-4. The sizes of replay buffers for meta controller and low level controller are set to be 5e4 and 6e4 respectively. For reward engineering, following (14) the agent receives reward $\phi = 10$ when achieving tasks and reward $\psi = -5$ when losing life. Cost at each step is set to be $r = 0$. Step cost r is set to be 0. Following (15) $\phi = 150$. The training procedure is divided into two phases. (1) In the first phase, the epsilon rate of meta controller is set to 1 so that all the controllers are pretrained sufficiently to solve a subset of tasks. (2) In the second phase,the meta controller is trained from scratch with epsilon rate annealing from 1 to 0.05 in 1e6 steps and the lower level controllers are fixed with epsilon rate set to be 0.05. Since we only perform reasoning on task-level, it's fair to compare the performance of algorithms with low level controllers fixed.