Learning Neuro-Symbolic World Models with Logical Neural Networks

Don Joven Agravante, Daiki Kimura, Michiaki Tatsubori

[1] IBM Research

Abstract

Model-based reinforcement learning has shown great results when using deep neural networks for learning world models. However, these results are not directly applicable to many real-world problems that require explainable models and where training data is limited. A more suitable problem setting that can address these issues is relational model-based reinforcement learning where a logical world model is learned. In this setting, we propose to use Logical Neural Networks (LNN) which enable the scalable learning of logical rules. Our method builds around the LNN by creating a framework for learning lifted logical operator models. This is used together with object-centric perception modules and AI planners that reason about the learned logical world model. We first test our agent by comparing the LNN-learned models against the existing handcrafted models which are available in the PDDLGym environments. For these tests, we show that our agent performs optimally and is on-par with planning on expert-crafted models. We then further test our agent in a text-based game domain called TextWorld-Commonsense where expert-crafted models are not available. For this domain, deep reinforcement learning agents are the state-of-the-art and we showed that we significantly outperform all of the existing agents.

Introduction

Reinforcement learning (RL) with deep neural networks (NN) is an effective method for solving complex sequential-decision-making problems as evidenced by great results in Atari (Mnih et al. 2015) and Go (Silver et al. 2016) to name a few. In recent years, a lot of research was focused on model-based RL (MBRL), wherein an explicit world model is learned for planning (Schrittwieser et al. 2020; Łukasz Kaiser et al. 2020; Hafner et al. 2021; Ye et al. 2021). These methods showed significant improvements over the previous model-free systems and they all leverage deep neural networks for their world model. Although these are very effective for the end results, these methods also inherit the issues of deep NNs such as a lack of explainability and a need for a huge amount of training data. These are required for several real-world applications so we seek to address these issues by instead learning world models which are explicitly represented by logic.

Relational RL (Džeroski, De Raedt, and Driessens 2001) is a subset of RL where the states and actions of the environment are in a relational logic form. It uses relational learning or inductive logic programming (ILP) resulting in explainable logic rules. Relational MBRL (Croonenborghs et al. 2007) is the MBRL form of this where the world model being learned also has a logic form. Although relational RL was demonstrated well in these works, it has two major drawbacks - the need for logical representations of the environment and the *brittleness* and difficulty in scaling ILP to large datasets. We believe it is a good time to revisit these issues because we can leverage the recent advances of NNs directly relating to both of these.

We envision an agent consisting of two modules: First, an object-centric perception module which converts raw observations into relational logic. Ideally, this module perfectly converts an RL environment into a relational RL environment. Recently, deep NNs have shown to be very effective at this task which is named differently in different literature sources according to modality (e.g. semantic parsing for text; scene graph prediction for images). The second module is the relational MBRL agent which is the focus of this paper. We concentrate on the second problem of learning the logical world model but we incorporate into our design a consideration about the first issue. In particular, we can assume that logical states are very accurate but imperfect. This requires learning from noisy data which classical ILP methods cannot provide.

In recent years, ILP has also advanced by using NNs which has increased its scalability and allowed it to learn with noisy data (Evans and Grefenstette 2018). This opens new research directions, one of which is relational RL agents. In this direction, we propose to use the Logical Neural Networks (LNN) (Riegel et al. 2020) as the core building block because it provides a solid foundation in the theory of real-valued logics. We then leverage the work of (Sen et al. 2021) which provides ILP functionality with the LNN. This Neuro-Symbolic ILP (NS-ILP) method will be used to learn the logical world model in our agent. Moreover, we design our framework such that our logical world model can be converted directly into STRIPS-like operators or a basic PDDL (Planning Domain Definition Language). Doing this allows us to use AI-planning systems compatible with this standard.

The contributions of our work are as follows:

- We design a novel framework using the LNN and NS-ILP for relational MBRL. The design is such that the learned

model can be converted into the standard format used by classical planning (PDDL).

- We extend our classical planning framework for partial observable environments by adding novelty-guided exploration and replanning.
- We designed experiments and evaluated our framework. In the experiments on the TextWorld-Commonsense domain we also show that we significantly outperform all of the existing agents.

Background

This section outlines the necessary theoretical foundations to describe our work in relational model-based RL. We formally describe it starting from the building blocks required to describe MDPs and ILP. These subsections serve as an overview to recall important topics.

Markov Decision Processes

RL environments are often formalized as Markov Decision Processes (MDP) which are defined as $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$. An agent interacts with an MDP by perceiving the state, $s \in \mathcal{S}$, and then taking action, $a \in \mathcal{A}$. By doing this at a certain time, t, a transition of the state occurs according to the conditional transition probabilities, $T(s_{t+1}|s_t, a_t)$. This satisfies the Markov property of being conditionally independent from the full history of states and actions. During the interaction, the agent also receives scalar rewards, r, according to a reward function, R, such that $r_t = R(s_t, a_t)$.

The Partially Observable Markov Decision Process (POMDP) is an extension of the MDP defined as $\langle \mathcal{S}, \mathcal{A}, T, \mathcal{O}, R \rangle$. In a POMDP, the state is hidden from the agent. Instead, the agent perceives observations, o, according to the conditional obervation probabilities, $\mathcal{O}(o_t|s_t)$. To solve POMDPs, it is necessary to keep track of the history of states and actions in some way. This may be done for example by keeping track over a *belief* over the set of states.

Relational Logic

Relational logic expands upon propositional logic. Here, we are mainly interested in variables and the abstraction it allows. To recall these concepts, let us use the example `on(book,table)`, which means that a `book` is `on` a `table`. The predicate `on` describes the relation between the objects `book` and `table`. The example proposition can be abstracted or *lifted* into `on(x,y)` with the variables, (x, y) replacing the *object groundings*. Since our predicate `on` carries a consistent *meaning*, reasoning about lifted statements allows an abstraction of concepts that is useful for generalization.

Relational Markov Decision Processes

A relational MDP further specifies an MDP by using relational logic. A state is decomposed as a set of the logical atoms which are true, f, such that $s = \{f_0, \ldots, f_n\}$. Each atom is composed of a predicate, p, and one or more objects, σ, such that $f = p(\sigma_0, \ldots, \sigma_m)$. The arity, m is fixed according to each predicate. Actions are also represented with this same predicate calculus but only as a single atom. Using the predicates and objects, the relational MDP can be defined as $\langle \mathcal{S}_p, \mathcal{A}_p, \Sigma, T, R \rangle$ where the state predicates, $\mathcal{S}_p$, action predicates $\mathcal{A}_p$ and set of objects, Σ, replace the state and action space of the MDP.

Inductive Logic Programming

An ILP problem can be defined formally as $\langle \mathcal{B}, \mathcal{P}, \mathcal{N}, \mathcal{T} \rangle$ where $\mathcal{B}$ is a background knowledge, $\mathcal{P}$ is a set of positive examples, $\mathcal{N}$ is a set of negative examples. The task is to find a rule that entails all elements of $\mathcal{P}$ and none of the elements in $\mathcal{N}$. A rule (or clause) is defined as $h \leftarrow b_0 \wedge b_1 \ldots \wedge b_n$, where the head atom h is true if the conjunction of all atoms in the body b is true. For practical applications, the search space of the rule is limited by a given rule template, $\mathcal{T}$.

Relational Model-Based RL

The main objective in relational MBRL is to allow an agent to learn and solve a relational MDP through interaction without knowing the model of the world beforehand. At each time step, an agent can explore the MDP by taking an action to obtain a data tuple (s_t, a_t, s_{t+1}, r_t). This is repeated several times after which an agent then uses a batch of data to try to estimate the world model $\langle T, R \rangle$. This is usually done iteratively but can possibly be done in a single batch within the offline RL framework. We assume that our world model has some underlying *structure* which can be learned. The general problem then becomes that of formulating this learning problem into the ILP form. This means that the world model needs to be broken down into rules and the data needs to be classified into positive and negative examples. Doing these requires design choices which will be specific to the methods. After the learning process, a world model is available which we can use to plan on the environment to maximize the reward or to reach a given goal.

Learning World Models with LNN

In this section we outline the core of our algorithm and the design choices we make throughout. We assume that we are in the relational MBRL setting already. Here, the main design choice pertains to how the world model will be broken down into a set of rules and how these rules will be learned.

To start with, we concentrate on the transition model, T, where the general learning problem can be written as:

$$\begin{aligned} T = &\operatorname*{argmin}_{T} \mathcal{L}(T, s_t, s_{t+1}, a_t) \\ \text{s.t.} \quad & s_{t+1} = T(s_t, a_t). \end{aligned} \tag{1}$$

The most common instantiation of this for continuous states is to define the loss, $\mathcal{L}$, as:

$$\mathcal{L} \coloneqq \sum_t \left\| s_{t+1} - T(s_t, a_t) \right\|.$$

However, we cannot use this as we have logical states, actions and transition models. Instead of using Eq.(1) directly, we break it down into finding components of T. First, we decompose T into smaller logical components. For this, we

propose to use the most well-known action model definition which is the STRIPS operator. This has an added benefit later on with the ease of converting it to PDDL which gives us access to powerful planning systems. In STRIPS, each action operator is a quadruple of $\langle \alpha, \beta, \gamma, \delta \rangle$. Each element is a set of logical conditions where α are conditions that must be true for the action to be executable, β are ones that must be false, γ are ones made true by the action and δ are ones made false. Collectively, α and β make up the preconditions while γ and δ are the postconditions/effects. So, with n actions the transition function is:

$$T = \{\langle \alpha, \beta, \gamma, \delta \rangle_0, \langle \alpha, \beta, \gamma, \delta \rangle_1, \ldots, \langle \alpha, \beta, \gamma, \delta \rangle_n\}. \quad (2)$$

Each of the elements of the operator are clauses in lifted logic statements. So, for example, with m different conditions, α is:

$$\alpha \leftarrow b_0 \wedge b_1 \ldots \wedge b_m, \quad (3)$$

where β, γ and δ are similarly defined so we only write out α for the next few steps. All the conditions are lifted logic statements which means that we need to abstract our data into the lifted logic before the ILP process. This means that we remove the association of the atoms to objects and operate on abstracted variables. The differentiable ILP process can be expressed by first expanding Eq.(3) with weights, $\mathbf{w} = \{w_0, w_1, \ldots, w_m\}$, such that:

$$\alpha \leftarrow w_0 b_0 \wedge w_1 b_1 \ldots \wedge w_m b_m, \quad (4)$$

which can then be used in an optimization problem to find the weights:

$$\begin{aligned} \mathbf{w} = & \underset{\mathbf{w}}{\operatorname{argmin}} \ \mathcal{L}(\alpha, \mathbf{b}) \\ \text{s.t.} \quad & \mathbf{w} \in \mathcal{W}, \end{aligned} \quad (5)$$

where the loss, $\mathcal{L}$, is defined to minimize the discrepancy in the dataset containing both positive and negative examples, $\mathcal{P}$ and $\mathcal{N}$. A loss function here can simply be a count of misclassified rules compared to the dataset. For classical ILP, $\mathcal{W}$ is defined such that the weights are constrained to be boolean (either zero or one). However, in this case, Eq.(5) is a difficult combinatorial optimization problem. In our case, $\mathcal{W}$ is relaxed such that individual weights are continuous valued between zero and one. This specification allows us to use LNNs as outlined in (Sen et al. 2021).

With this formulation, we will have 4 rules to be learned for n action predicates and each head of the rule corresponds to a part of the STRIPS operator. The body of the rules all have an exhaustive list of m lifted atoms. This means that we need to learn $4 \times n \times m$ weights to fully reconstruct the transition function, T. An example of this is shown in Figure 1. Each of these rules can be constructed using a combination of the LNN$-pred$ and LNN-$\wedge$ as described in section 4 of (Sen et al. 2021). Negations are also added for β and γ.

Although the rules are structurally similar, the training is done differently for preconditions (α and β) and effects (γ and δ). The difference lies in transforming the data samples (containing s_t, a_t, s_{t+1}) into the positive examples, $\mathcal{P}$, and negative examples, $\mathcal{N}$ of the rules to be learned which are defined by the input conditions, $\mathbf{b}$. For the LNNs of α and β, the inputs are given the corresponding logical values of the atoms in the state, s, that is, $\mathbf{b} \coloneqq s$. This is a slight abuse of notation for clarity but to be precise, the logical values of $\mathbf{b}$ are set to be true if (with a value 1.0) if they are present in s otherwise it is set to be false (with a value 0.0). The output (head of the rule) is true when two conditions are met. First if action, a, in the data sample corresponds to the action predicate being modeled by α and β, and second if $s \neq s'$ (a change in state occurred). Otherwise, it is false. For the LNNs of γ and δ, the inputs are given the logical values corresponding to the difference in the atoms of s and s' such that γ are the the conditions made true and δ those that are made false. Again, with a slight notation abuse we can write that for γ, we have $\mathbf{b} \coloneqq s_{t+1} - s$ and for δ we have $\mathbf{b} \coloneqq s - s_{t+1}$. Similarly, the output is true when action, a, corresponds otherwise it is false.

Once the LNN structure and data/rule examples are set, we can take advantage of the machinery of gradient-based optimization in a supervised learning setting (Riegel et al. 2020; Sen et al. 2021). When learning converges, we have a set of weights for each of the corresponding elements that correspond to real-valued logic. These may be interpreted as probabilistic transitions but we can also threshold these when we are expecting a deterministic structure from the environment (Sen et al. 2021). The pseudocode for training the LNN such that it will learn STRIPS operators is shown in Algorithm 1. This assumes that from from RL exploration there exists a dataset, D, of interactions with the environment.

Algorithm 1: LNN training for learning STRIPS operators

Input:
Dataset, $D = \{(s, a, s')_1, \ldots, (s, a, s')_t\}$
Parameters:
LNN parameter for logic thresholding, LNN$_{th}$
LNN paramater for maximum training epochs, LNN$_{ep}$
Outputs:
STRIPS Operator in real-valued logic, $\langle \alpha, \beta, \gamma, \delta \rangle$
PDDL_actions

D = variable_abstraction(D)
S_p = get_unique_state_predicates(D)
A_p = get_unique_action_predicates(D)
$\langle \mathcal{P}, \mathcal{N} \rangle$ = transform_dataset(D)
for a_p **in** A_p
    S_{var} = permute_variables(S_p, get_arity(a_p))
    S_{pos} = LNN-PRED(S_{var})
    S_{neg} = LNN-PRED(LNN-NOT(S_{var}))
    αa_p = LNN-AND(S_{pos})
    βa_p = LNN-AND(S_{neg})
    γa_p = LNN-AND(S_{pos})
    δa_p = LNN-AND(S_{neg})
    $\langle \alpha a_p, \beta a_p \rangle$ = train_precond($\mathcal{P}$, $\mathcal{N}$, LNN$_{th}$, LNN$_{ep}$)
    $\langle \gamma a_p, \delta a_p \rangle$ = train_postcond($\mathcal{P}$, $\mathcal{N}$, LNN$_{th}$, LNN$_{ep}$)
    $\langle \alpha, \beta, \gamma, \delta \rangle$.append($\alpha a_p, \beta a_p, \gamma a_p, \delta a_p$)
end for
PDDL_actions = threshold($\langle \alpha, \beta, \gamma, \delta \rangle$, LNN$_{th}$)
return $\langle \alpha, \beta, \gamma, \delta \rangle$, PDDL_actions

The remaining design choice pertains to the reward func-

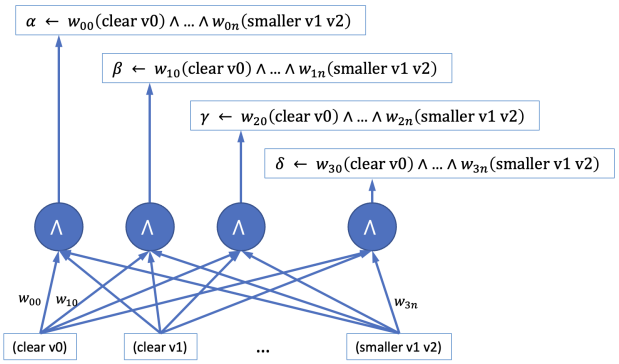

Figure 1: An example of an LNN operator. This corresponds to the move operator for the tower of hanoi game. This is a simplified diagram where negations and pred operators are not shown

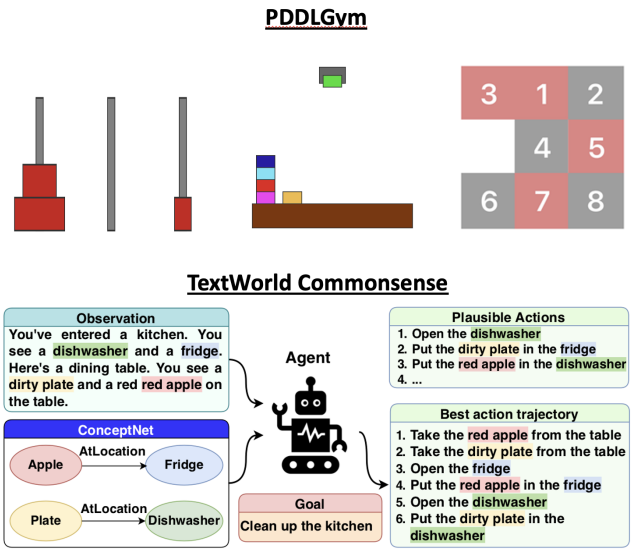

Figure 2: Illustrations of the domains for our experiments: PDDLGym (Silver and Chitnis 2020) and TextWorld Commonsense (Murugesan et al. 2021)

tion model. It is possible to model it into logic in a similar manner as the transition model. However, we think that being given logical goals is a good feature for our agent. In fact, this is already a common and well-accepted assumption in several RL works whether the goal appears as a separate parameter in *goal-conditioned RL* or is incorporated as part of the state. Because of this, we consider this to be a weak assumption that is already used in current RL research. This has even been expanded in recent research with more explicit forms of the reward function being given as an input such as the *reward machines* (Icarte et al. 2018).

Enhancements

This section details some important components surrounding our RL agent which gives significant performance increase or new functionality.

Exploration for Relational RL

Exploration is a key part of any RL agent and it is no different even in the relational RL setting (Lang, Toussaint, and Kersting 2012). For our agent, we implemented a standard count-based exploration method but we implemented changes that take into account the nature of the relational logic state. Specifically, our exploration counts are based on the abstracted lifted state-action pairs instead of the grounded ones. This simple change gives a significant boost in data-efficiency and takes advantage of the generalization inherent in relational representations. A more nuanced and detailed implementation of this idea was presented in (Lang, Toussaint, and Kersting 2012).

Enabling our agent for POMDP Settings

Planning in the most general POMDP settings is intractable. However, it has been shown that if some assumptions can be made about the environment then classical replanning will be effective (Bonet and Geffner 2011). This is the main assumption. It can be further extended by considering information gathering actions as in (Yang et al. 2021). This is another exploration-related enhancement. Specifically, we propose to designate as information gathering actions those which reveal more information about the world in the form of new atoms in the state. We also implement a belief-state-tracker that is standard with POMDP planning.

Experimental Setup

To evaluate our agent, we chose two different sets of domains as shown in Figure 2. The first one uses classical planning domains and is aimed at comparing and contrasting our results against models of well-studied domains crafted by human experts. The second is a text-based game and it was chosen to be able to compare and contrast our approach against contemporary deep RL methods. Another major consideration for choosing both types of domains is that these both have underlying relational representations that can be accessed. This allows us to test and experiment with our relational RL agent before tackling complex interactions with the object-centric perception modules.

In either domain, once we have a logical world model in the form of STRIPS operators, we can use this with a classical planner to complete our game-playing agent. For our convenience, we convert the STRIPS operators into PDDL operators by combining (α, β) into the preconditions and (γ, δ) into the effects. The PDDL allows us to take advantage of modern AI planning systems such as the Fast-Downward Planner (Helmert 2006) which we used throughout the experiments.

Classical Planning Environments

Relational RL methods have historically been tested and targeted towards classical planning environments (Croonenborghs et al. 2007; Džeroski, De Raedt, and Driessens 2001; Lang, Toussaint, and Kersting 2012). A modern implementation of such domains is collected in the PDDLGym (Sil-

Domain	Source	Solved %	# Rules
Hanoi	Handcrafted	100 %	8
	Learned	100 %	17
Blocks	Handcrafted	100 %	37
	Learned	100 %	49
Slidetile	Handcrafted	100 %	48
	Learned	100 %	288

Table 1: Results on the classical planning environments and a comparison of the number of rules for modeling the world.

ver and Chitnis 2020). This enables the use of many different domains from the classical planning literature within the modern *gym* framework of RL environments.

Among the available options, we chose 3 well-known domains: the towers of hanoi, blocksworld and slidetile/slide puzzle. These were chosen for their popularity and diversity in the size and complexity of the expert-crafted PDDL models which are available in PDDLGym.

Text-Based Games

Text-Based Games began as a form of entertainment in the 1980s where players could read a narrative and imagine the state of the game world from text. They can then interact with this world through text input as well. In recent years, text-based games have become an interesting benchmark in the intersection of natural language processing and sequential decision making. Recently, several sets of benchmarks and game environments were proposed such as TextWorld (TW) (Côté et al. 2018), Jericho (Hausknecht et al. 2020) and TextWorld Commonsense (TWC) (Murugesan et al. 2021). Among these, we first chose the TextWorld-based games due to the availability of the relational logic state (Côté et al. 2018). In this paper, we specifically chose TWC among the different TW datasets as it had another neuro-symbolic method that we can compare against.

Text-based games are inherently POMDPs (Côté et al. 2018). As such all the enhancements we described in section 4.2 are required for these tests.

Results and Discussions

This section shows the results of our experiments. The classical planning environments are meant to compare against expert-crafted models and text-based games are meant to compare against other agents, in particular deep RL methods.

Classical Planning Environments

Our results are summarized in Table 1. This compares the human-handcrafted PDDL domain to the operators learned by the LNN in 3 well-known planning domains - the towers of hanoi, blocksworld and slidetile. These are well-studied games so the handcrafted PDDL works on 100% of the problem instances. Our results show that the LNN-learned model can do the same, verifying the correctness of our approach.

The more interesting result is shown in the last column where our method always learns more rules. Most of these

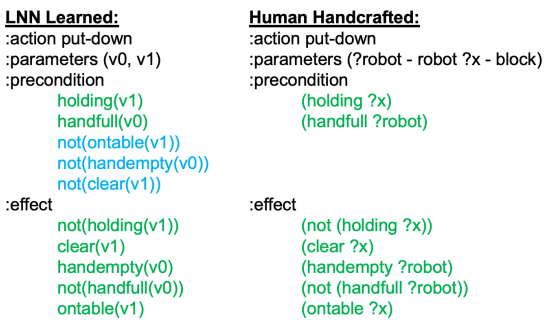

Figure 3: Comparison between the LNN-learned model and expert-crafted model for one of the actions of the blocksworld domain. Green indicates rules that correspond on both sides. Blue indicates rules that are unique to the LNN-learned model

are in the preconditions which indicates a smaller state-transition-graph, which is better. On inspecting the *extra* rules, we see that a lot of these are *mutex conditions*. For example, in Hanoi, one of the handcrafted preconditions is *on(disc, from)*. This was also learned by the LNN along with a related *extra* precondition: *not(on(from, disc))*. Another example can be seen in Figure 3 which shows one of the actions in the blocksworld domain. We can see in this example the matching rules in green and the extra rules in blue. Using this, we can appreciate the level of inherent explanability that our model possessess.

One limitation of our work can be found in the result for the slidetile domain. Here, there is a significant difference in the number of rules learned. One can imagine that this becomes a hindrance to the explanability - although all the rules may be correct and can be followed, it becomes tedious to check each of the 288 rules. In this case, perhaps more rules are not necessarily better. Indeed, our current method has no mechanism that pushes it towards parsimonious models. For applications requiring such, this may be the subject of future research.

Text-Based Games

For our results, we first show some examples of the learned rules in our logical world model in Figure 4. This is in the converted PDDL form. Here, we can visually inspect the validity of the rules. For example, for the *take* action the effect would be that the object $v0$ is no longer $at(v1)$ but now it is in the inventory $v3$. This level of *explanability* is inherent in logical models although it requires careful inspection.

It would be tedious to inspect the rules individually, but what would be more interesting is if taken altogether can these rules allow us to plan optimal actions in the world. To answer this, we present our results in Table 2. Here, each row corresponds to the results of a different agent for this benchmark environment. Our method appears on the bottom row and the top row shows the best possible metrics (labeled optimal). Of note, we show the two best performing deep RL agents in (Murugesan et al. 2021) which is the

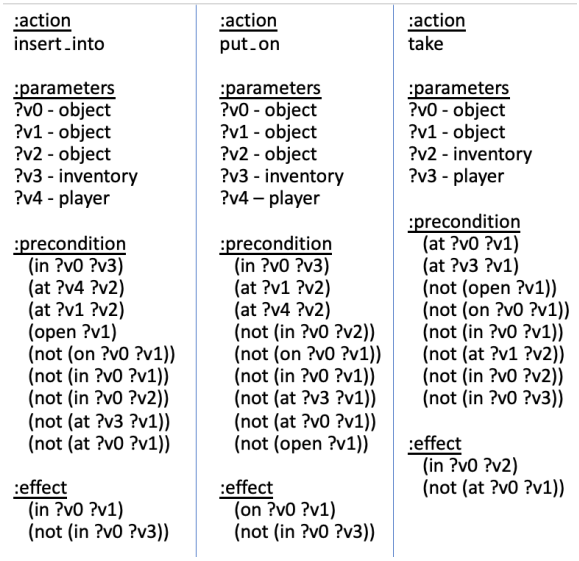

Figure 4: Examples of the learned action models for TextWorld

KG-A2C and DRRN. We also show a model-free neuro-symbolic agent (NeSA). The TWC games are categorized into Easy-Medium-Hard with a validation and testing set for each as shown in the columns. Our results show that planning on our learned model can produce the same optimal actions for all the Easy and Medium games and for the validation set of the Hard games. An interesting limitation appears in the test set of the Hard games wherein novel predicates appear in the test set that do not appear in any of the training or validation set. This is a current limitation of our system which does not have any mechanism for handling such novel predicates. These results show that our method clearly outperformed the 2 best deepRL agents and even clearly outperformed the Note however that we have additional assumptions differing from the plain deep RL setting of the original setup in (Murugesan et al. 2021). We give this comparison as a reference on the potential improvement our overall approach might provide. The comparison to the model-free NeuroSymbolic approach is a bit fairer as they also look to use relational logic. in terms of the number of steps.

Related Work

The most pertinent works were cited and discussed throughout the previous sections. This section collects other relevant works that concentrate on specific but important topics that were not mentioned in enough detail.

Model-Free RL for Text-based Games

Text-based games (Côté et al. 2018; Murugesan et al. 2021; Hausknecht et al. 2020) are a testbed for tackling challenges in reinforcement learning and natural language processing. LSTM-DQN (Narasimhan, Kulkarni, and Barzilay 2015) is a study on an LSTM-based encoder for feature extraction from observations and Q-learning for action policies. LSTM-DQN++ (Yuan et al. 2018) extended exploration, and LSTM-DRQN was proposed for adding memory units in the action scorer. KG-DQN (Ammanabrolu and Riedl 2019) and GATA (Adhikari et al. 2020) extended the language understanding. LeDeepChef (Adolphs and Hofmann 2020) uses recurrent feature extraction along with A2C (Mnih et al. 2016). CREST (Chaudhury et al. 2020) was proposed for pruning observation information. These methods tackled the solving of games with deep methods, not neuro-symbolic method.

Since these methods do not have interpretability in trained rules, recent studies (Kimura et al. 2021b; Chaudhury et al. 2021; Kimura et al. 2021a) introduced neuro-symbolic methods. FOL-LNN (Kimura et al. 2021b) introduced the training of first-order logic for increasing the training speed and interpretability in LNN (Riegel et al. 2020). SLATE (Chaudhury et al. 2021) trains interpretable action policy rules from symbolic abstractions of textual observations for improving generalization. LOA (Kimura et al. 2021a) is a demo for a neuro-symbolic method with simple rules without showing/editing an actual network from the training.

Other approaches to learning planning models

In this paper, we advocated for the formalism of relational RL but many other works also propose methods to learn models for AI planning. For example, a representative work is (Pasula, Zettlemoyer, and Kaelbling 2007) which uses greedy search for learning the rule sets. Older literature also exists such as (Benson 1995) and much more recent works such as (Silver et al. 2021) follows a similar procedure. These works are along the same line as using classical ILP approaches for relational RL. Although these are shown to work well for smaller to medium-sized problems, we believe that a fundamental limitation exists due to the discrete and combinatorial nature of the problem formulation. In contrast, we showed in our method how differentiable logic can be used. Furthermore, we showed in this paper that this branching out from classical ILP can be cleanly described as the choice of constraint in the optimization problem which then leads to the choice of available optimization methods. Becasue of these differences, we believe our work is orthogonal and worth investigating in parallel.

Another related line of work is presented in (Asai and Fukunaga 2018) which leveraged deep NNs for the model such that their method plans in the latent space. Such works are also orthogonal to our main concern of bringing explicit logical models which are explainable and grounded in the theory of logics.

Conclusion

We outlined and proposed a framework that centers around the relational model-based RL setting and advocates for the using a neuro-symbolic ILP module for learning logical world models. This can be used with classical planning systems to produce optimal actions in several different game worlds. We believe this approach shows promise and we presented results and experiments on the key component which

	Easy		Medium		Hard	
	Valid	Test	Valid	Test	Valid	Test
Optimal (Upper Bound)	2.4 steps 100% score	2.4 steps 100% score	4.4 steps 100% score	3.6 steps 100% score	13.6 steps 100% score	14.0 steps 100% score
KG-A2C [1]	17.65 ± 3.62 $85\% \pm 7\%$	18.00 ± 3.24 $87\% \pm 5\%$	37.18 ± 4.86 $72\% \pm 7\%$	43.08 ± 4.13 $54\% \pm 17\%$	49.36 ± 7.5 $46\% \pm 10\%$	49.96 ± 0.0 $22\% \pm 0\%$
DRRN [1]	18.88 ± 2.69 $81\% \pm 8\%$	19.49 ± 4.89 $84\% \pm 8\%$	33.41 ± 2.81 $73\% \pm 6\%$	40.49 ± 4.41 $56\% \pm 7\%$	46.20 ± 4.86 $44\% \pm 1\%$	50.00 ± 0.0 $18\% \pm 10\%$
Model-free NeSA [2]	- -	15.0 ± 0.0 100%	- -	28.6 ± 0.0 100%	- -	- -
LNN-MBRL (Ours)	2.4 ± 0.0 100%	2.4 ± 0.0 100%	4.4 ± 0.0 100%	3.6 ± 0.0 100%	13.6 ± 0.0 100%	28.4 ± 0.0 60.6%

Table 2: Scores on the TextWorld Commonsense(TWC) set of games. The top row numbers are the number of steps an agent takes (lower is better) and the bottom row is the normalized score (higher is better) [1] indicates the results are taken directly from (Murugesan et al. 2021) [2] indicates work that was extended from (Kimura et al. 2021a)

learns the logical world models. The natural progression of this work is to use real object-centric perception modules. Although our method is designed with noisy logical states in mind, the extent of the tolerable noise might differ in practice and new innovations may be required.

References

Adhikari, A.; Yuan, X.; Côté, M.-A.; Zelinka, M.; Rondeau, M.-A.; Laroche, R.; Poupart, P.; Tang, J.; Trischler, A.; and Hamilton, W. 2020. Learning Dynamic Belief Graphs to Generalize on Text-Based Games. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 3045–3057. Curran Associates, Inc.

Adolphs, L.; and Hofmann, T. 2020. Ledeepchef deep reinforcement learning agent for families of text-based games. In *AAAI*, 7342–7349.

Ammanabrolu, P.; and Riedl, M. 2019. Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3557–3565. Minneapolis, Minnesota: Association for Computational Linguistics.

Asai, M.; and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Benson, S. 1995. Inductive learning of reactive action models. In *Machine Learning Proceedings 1995*, 47–54. Elsevier.

Bonet, B.; and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Chaudhury, S.; Kimura, D.; Talamadupula, K.; Tatsubori, M.; Munawar, A.; and Tachibana, R. 2020. Bootstrapped Q-learning with Context Relevant Observation Pruning to Generalize in Text-based Games. In *EMNLP*, 3002–3008.

Chaudhury, S.; Sen, P.; Ono, M.; Kimura, D.; Tatsubori, M.; and Munawar, A. 2021. Neuro-Symbolic Approaches for Text-Based Policy Learning. In *EMNLP*, 3073–3078.

Côté, M.-A.; Kádár, A.; Yuan, X.; Kybartas, B.; Barnes, T.; Fine, E.; Moore, J.; Tao, R. Y.; Hausknecht, M.; Asri, L. E.; Adada, M.; Tay, W.; and Trischler, A. 2018. TextWorld: A Learning Environment for Text-based Games. *CoRR*, abs/1806.11532.

Croonenborghs, T.; Ramon, J.; Blockeel, H.; and Bruynooghe, M. 2007. Online Learning and Exploiting Relational Models in Reinforcement Learning. In *IJCAI*, 726–731.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational Reinforcement Learning. *Machine Learning*, 43(1): 7–52.

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

Hafner, D.; Lillicrap, T. P.; Norouzi, M.; and Ba, J. 2021. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*.

Hausknecht, M.; Ammanabrolu, P.; Côté, M.-A.; and Yuan, X. 2020. Interactive Fiction Games: A Colossal Adventure. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05): 7903–7910.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

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 *International Conference on Machine Learning*, 2107–2116. PMLR.

Kimura, D.; Chaudhury, S.; Ono, M.; Tatsubori, M.; Agravante, D. J.; Munawar, A.; Wachi, A.; Kohita, R.; and Gray, A. 2021a. LOA: Logical Optimal Actions for Text-based Interaction Games. In *ACL*, 227–231.

Kimura, D.; Ono, M.; Chaudhury, S.; Kohita, R.; Wachi, A.; Agravante, D. J.; Tatsubori, M.; Munawar, A.; and Gray, A.

2021b. Neuro-Symbolic Reinforcement Learning with First-Order Logic. In *EMNLP*, 3505–3511.

Lang, T.; Toussaint, M.; and Kersting, K. 2012. Exploration in Relational Domains for Model-based Reinforcement Learning. *Journal of Machine Learning Research*, 13(119): 3725–3768.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

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. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Murugesan, K.; Atzeni, M.; Kapanipathi, P.; Shukla, P.; Kumaravel, S.; Tesauro, G.; Talamadupula, K.; Sachan, M.; and Campbell, M. 2021. Text-based RL Agents with Commonsense Knowledge: New Challenges, Environments and Baselines. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10): 9018–9027.

Narasimhan, K.; Kulkarni, T.; and Barzilay, R. 2015. Language Understanding for Text-based Games using Deep Reinforcement Learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1–11. Lisbon, Portugal: Association for Computational Linguistics.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29: 309–352.

Riegel, R.; Gray, A. G.; Luus, F. P. S.; Khan, N.; Makondo, N.; Akhalwaya, I. Y.; Qian, H.; Fagin, R.; Barahona, F.; Sharma, U.; Ikbal, S.; Karanam, H.; Neelam, S.; Likhyani, A.; and Srivastava, S. K. 2020. Logical Neural Networks. *CoRR*, abs/2006.13155.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Sen, P.; de Carvalho, B. W. S. R.; Riegel, R.; and Gray, A. G. 2021. Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks. *CoRR*, abs/2112.03324.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. *CoRR*, abs/2002.06432.

Silver, T.; Chitnis, R.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3182–3189. IEEE.

Yang, S.; Mao, X.; Wang, Q.; and Huang, Y. 2021. A Hybrid Planning Approach for Accompanying Information-gathering in Plan Execution Monitoring. *Journal of Intelligent & Robotic Systems*, 103(1): 19.

Ye, W.; Liu, S.; Kurutach, T.; Abbeel, P.; and Gao, Y. 2021. Mastering Atari Games with Limited Data. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 25476–25488. Curran Associates, Inc.

Yuan, X.; Côté, M.; Sordoni, A.; Laroche, R.; des Combes, R. T.; Hausknecht, M. J.; and Trischler, A. 2018. Counting to Explore and Generalize in Text-based Games. *CoRR*, abs/1806.11525.

Łukasz Kaiser; Babaeizadeh, M.; Miłos, P.; Osiński, B.; Campbell, R. H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; Mohiuddin, A.; Sepassi, R.; Tucker, G.; and Michalewski, H. 2020. Model Based Reinforcement Learning for Atari. In *International Conference on Learning Representations*.