A Hybrid Neuro-Symbolic approach for Text-Based Games using Inductive Logic Programming

Anonymous Authors

Anonymous Affiliations

Abstract

Text-based games (TBGs) have emerged as an important testbed, requiring reinforcement learning (RL) agents to combine natural language understanding with reasoning. A key challenge for agents solving this task is to generalize across multiple games and shows good results on both seen and unseen objects. Currently, pure deep learning-based RL systems can perform well to known entities and states. They, however, perform poorly in novel situations e.g., when handling out-of-vocabulary (OOV) objects. In the perspective of generalization, recent efforts in infusing external commonsense knowledge into an RL agent show better results than pure deep-learning systems. However, the policies learned by these systems are not interpretable or easily transferable. To tackle these issues, we have designed a hybrid neuro-symbolic framework for TBGs that uses symbolic reasoning along with the neural RL model. It employs inductive logic programming (ILP) to learn the symbolic rules (policies) as default theory with exceptions and is represented in the form of an answer-set-program (ASP) that allows performing non-monotonic reasoning in the partially observable game environment. We use WordNet as an external knowledge source to lift the learned rules to their generalized versions. These rules are learned in an online manner and applied with an ASP solver to predict an action for the agent. We show that the agents that incorporate the neuro-symbolic hybrid approach with the generalized rules outperform the baseline agents.

Introduction

Natural language plays a crucial job in human intelligence and cognition. TBGs become appropriate simulation environments for studying the language-informed sequential decision making process as the states and actions in these games are described in natural language. So, to solve these games an agent needs the skill of both natural language processing (NLP) and reinforcement learning (RL). At a high level, the existing agents can be classified into two classes - (a) rule-based agents, and (b) neural agents. Rule-based agents such as NAIL (Hausknecht et al. 2019) rely heavily on the prior pre-defined knowledge. This makes them less flexible and adaptable. To overcome the challenges of rule-based agents, in recent years, with the advent of the new deep learning techniques, significant progress has been made to the neural RL agents for TBGs (Adhikari et al. 2020; Narasimhan, Kulkarni, and Barzilay 2015). However, these frameworks suffer from a number of shortcomings. First, from deep-learning, they inherit the need for very large training sets, which entails that they learn slowly. Second, they are brittle in the sense that a trained network that may show good performance with the seen entities in the training environments, however, it performs very poorly in the similar environment with the unseen entities. Also, the policies learned by these neural RL agents are not interpretable, which makes them less traceable.

In this paper, we introduce a hybrid neuro-symbolic (HNS) architecture for TBGs that utilizes the positive features from both the neural and the symbolic agents. Instead of using pre-defined prior knowledge, the symbolic agent in HNS learns the symbolic policies by leveraging the reward and action pairs while playing the game. This allows the policies to be interpretable and very natural. Importantly, the rules are learned as default theories so that the agent can do non-monotonic reasoning. The non-monotonic reasoning is crucial in the partially observable world as the agent's belief can be changed in the presence of new examples. Also, we lift the rules using WordNet and that gives more generalization capabilities to the rules. The neural part of an HNS agent is responsible for doing the exploration in the environment and is used in the scenarios where symbolic agent fails to provide an action (due to lack of learned rules). We have tested our HNS agent on TextWorld-Commonsense framework (Murugesan et al. 2021) for TBGs that has received much attention among the TBGs researchers. Our results depict that the collaboration between the neural agent and the symbolic agent along with WordNet based generalization outperforms the current state-of-the-art results.

The main contributions of this paper are the following: (1) we show a hybrid neuro-symbolic architecture for TBGs that outperforms existing models in terms of steps and scores; (2) we discuss the importance of non-monotonic reasoning in the partially observable world; (3) we demonstrate how default theories can be learned with exceptions in an online manner for the TBGs; (4) we provide a novel information-gain based rule generalization algorithm that leverages WordNet.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Background

Text-based reinforcement learning: TBGs provide a challenging environment where an agent can observe the current state of the game and act in the world using only the modality of text. The agent perceives the state of the game only through natural language observations. Hence, TBGs can be modeled as a Partially Observable Markov Decision Process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r)$, where $\mathcal{S}$ is the set of states of the game, $\mathcal{A}$ is the natural language action space, $\mathcal{O}$ is the set of textual observations describing the current state, $\mathcal{T}$ are the conditional transition probabilities from one state to another, $\mathcal{E}$ are the conditional observation probabilities, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a scalar reward function, which maps a state-action pair to the reward received by the agent.

Inductive Logic Programming (ILP): ILP is one Machine Learning technique where the learned model is in the form of logic programming rules (Horn Clauses) that are comprehensible to humans. It allows the background knowledge to be incrementally extended without requiring the entire model to be re-learned. Meanwhile, the comprehensibility of symbolic rules makes it easier for users to understand and verify induced models and even edit them. Details can be found elsewhere (Muggleton and De Raedt 1994).

Answer Set Programming (ASP): An answer set program is a collection of rules of the form -

$$l_0 \leftarrow l_1, \ \ldots, \ l_m, \ not\, l_{m+1}, \ \ldots, \ not\, l_n.$$

Classical logic denotes each l_i is a literal (Gelfond and Kahl 2014). In an ASP rule, the left hand side is called the *head* and the right-hand side is the *body*. Constraints are ASP rules without *head*, whereas facts are without *body*. The variables start with an uppercase letter, while the predicates and the constants begin with a lowercase. We will follow this convention throughout the paper. The semantics of ASP is based on the stable model semantics of logic programming (Gelfond and Lifschitz 1988). ASP supports *negation as failure* (Gelfond and Kahl 2014), allowing it to elegantly model common sense reasoning, default rules with exceptions, etc.

s(CASP) Engine: s(CASP) (Arias et al. 2018) is a query-driven, goal-directed implementation of ASP that includes constraint solving over reals. Goal-directed execution of s(CASP) is indispensable for automating commonsense reasoning, as traditional grounding and SAT-solver based implementations of ASP may not be scalable. There are three major advantages of using the s(CASP) system: (i) s(CASP) does not ground the program, which makes our framework scalable, (ii) it only explores the parts of the knowledge base that are needed to answer a query, and (iii) it provides natural language justification (proof tree) for an answer (Arias et al. 2020).

Neuro-Symbolic RL Framework for Text-Based Games

The goal of this paper is to show how a neural and a symbolic agent can work together in an RL environment for the TBGs. The neural agents are good at exploration whereas the symbolic agents are good at learning interpretable policies that offer rewards and apply them to select a candidate set of actions. Keeping it as a motivation, in this work, we try to capitalize the power of both the agents to get better results. The main idea is to use the symbolic agent to learn the policies in the form of logic rules and apply them using an ASP solver. When the symbolic agent fails to provide a good action, then neural agent takes care of it as a fallback. In other words, the action selector gives priority to the symbolic agent over the neural. Figure 1 illustrates the components of our HNS architecture and shows an overview of the decision making process.

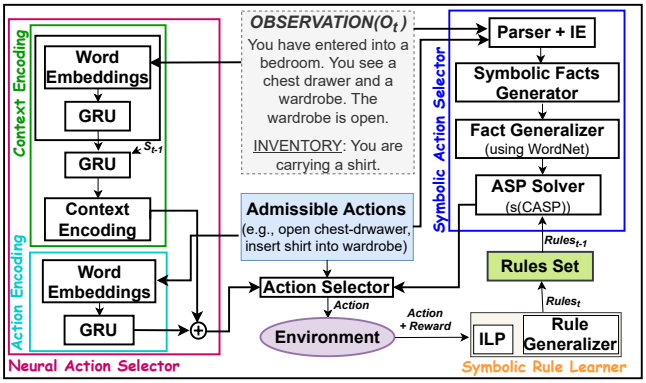

Figure 1: Overview of the HNS agent's decision making at any given time step. The Hybrid Neuro-Symbolic architecture mainly consist of 5 modules - (a) Context Encoder encodes the observation to dynamic context, (b) Action Encoder encodes the admissible actions, (c) Neural Action Selector combines (a) and (b) with $\bigoplus$ operator, (d) Symbolic Action Selector returns a set of candidate actions, and (e) Symbolic Rule Learner uses ILP and WordNet based rule generalization to generate symbolic rules.

Symbolic Policy Learner with Generalization

Deep reinforcement learning (DRL) has gained great success by learning directly from high-dimensional sensory inputs, yet it suffers from lack of interpretability. Interpretability of an agent's action is of utmost importance in sequential decision-making as it increases the transparency of the black-box-style DRL agents. Also, it helps the RL researchers to understand the high-level behavior of the system better. To make a system interpretable, one of the most vastly used approaches is learning the agent's policies symbolically. In our work, the HNS agent learns these symbolic policies in the form of logical rules represented in ASP language. For the HNS agent these learned logical ASP rules not only provide a better understanding of the system's functionality but also can be used to predict the agent's action using an ASP solver. HNS agent learns the rules iteratively and applies the rules to predict an action in collaboration with the neural agent. Our result shows, this hybrid approach is very effective in terms of performance.

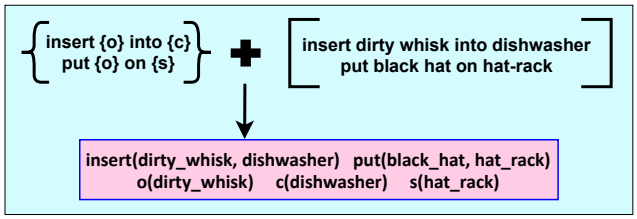

Figure 2: Entity extraction using Action Template

HNS agent works in a partially observable environment, where it needs to predict an action based on its prior knowledge. If it fails, then it learns a new thing which will be applied in the next episode. This is very similar to human thought process as we take decisions based on the knowledge we currently possess and the knowledge increases with experience. So, similar to a human, the reasoning approach of the HNS agent is non-monotonic in nature, which means *"what it believes currently may become false in the future with new evidence"*. We can model this using a non-monotonic logic programming paradigm which supports *default rules* and *exception to defaults* (Gelfond and Kahl 2014). In our work, the belief of an agent is represented as an Answer Set Program in the form of *default rules* with *exception*. With the help of ILP, these rules are learned by the HNS agent after each episode and then apply them in the following episode. Based on the outcome after applying the default rules, it updates the learned rules with the exception (if needed) and also learned new rules. In this way, the HNS agent incrementally learns these rules along with the exceptions.

Learning default theories using ILP

In this section, we briefly describe the default theory learning procedure using ILP in an RL environment. As mentioned earlier, the model learned by ILP is in the form of a logic program. We use ASP as the logic programming paradigm as it has not only the efficiency to represent the knowledge in default theories with exception using negation-as-failure (NAF) but also capable to do reasoning on the represented knowledge. So, the symbolic component of an HNS agent learns the default rules using ILP and uses ASP solver - s(CASP) to predict an action.

To apply an ILP algorithm, at first an HNS agent needs to collect the *State*, *Action*, and *Reward* pairs while exploring the text based environment. The main two components of the State description are the inventory information of the agent and the details of the entities in the environment. The inventory information, which is present explicitly in the HNS agent, can be easily extracted. To get the entity description, we utilize the admissible actions of an agent. This set of admissible actions are also provided by the game environment at each step and among these actions one action is the best to take in the perspective of direct/future reward. Now, the action templates, which are predefined to the agent, can be applied on the admissible actions to extract the entities that are present in the current environment. The TWC environment includes three main kinds of entities: *objects (O)*, *supporters (S)*, and *containers (C)* and using them the action templates are made. An example of an action template is — *"take {O} from {S}"*. Figure 2 illustrates an instance of a first-order predicate generation process by applying the action templates to the actions. Along with this *State* description, the HNS agent also stores the taken *Action* and the *Reward* information at each step.

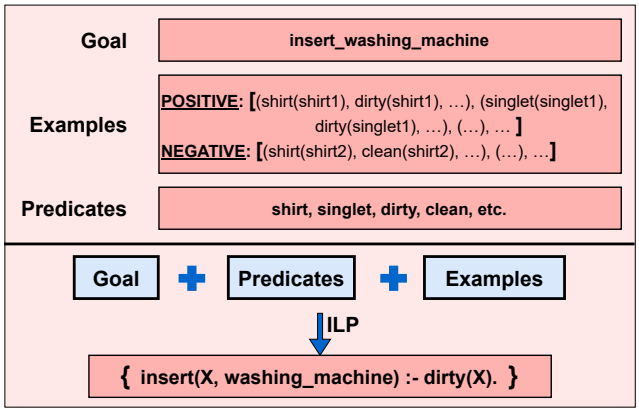

Figure 3: ILP Rule Learning Example

To learn the rules, an ILP algorithm requires mainly three types of information - *goal*, *predicate list*, and *examples*. The *goal* is the concept that the ILP algorithm is going to learn by exploring the examples. The *predicates* give the explanation to a concept. In the learned theory formulated as logical rules, *goal* is the head and the *predicate list* gives the domain space for the body clauses. The *examples* are the set of positive and negative scenarios that are collected by the agent while playing. In our work, we have mainly focused on learning the hypothesis for the rewarded actions. In TWC, the rewarded actions are location oriented or depend on the *storage* (S) and *containers* (C) entities. So, the agent first splits the gathered information (i.e., state, action, and reward) based on the action and the location of an object in the form of <*action_location*> (e.g., *insert_washing_machine, put_shelf,* etc.). This <*action_location*> becomes the *goal* for the ILP algorithm. Next, using the rewards (i.e., positive and negative) the agent splits the corresponding information of a goal in two separate mutually exclusive sets. These two sets become the positive and negative example sets for the ILP. Each example (structured as first-order predicate) in these sets are presented in the form of <*feature(object_id)*>, where the *object_id* represents a unique instance of the object type. It is important to distinguish two or more different objects of the same type in the same environment with distinct features. Also, the agent creates the predicate list by extracting the predicate names from the examples. Now, after having the goal, predicate list, and the example, the HNS agent runs the ILP algorithm to learn the hypothesis and then do simple string post-processing to get the hypothesis in the below form —

```
action( X , location) <- feature(X).
```

Figure 3 elaborates the ILP data preparation procedure along with an example of a learned rule.

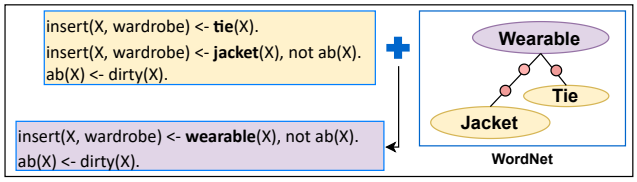

Figure 4: Example of Rule Generalization

HNS agent learns these rules after each episode iteratively and applies the rules in the next step. For applying these rules, HNS agent uses goal-directed ASP solver - s(CASP) (Arias et al. 2018). Similar to the ILP data preparation procedure (described above), HNS agent first collects the state and the admissible actions set and represents them as ASP facts. Then, employ the s(CASP) engine to get the possible set of action pairs from the learned rules and the current state information. As we learn after each episode, because of very little exploration, the learned rules in the initial stages are not great and it gradually improves with the no. of training episodes. One of the key components of this improvement is the exception learning. The exception is a clause in the rule's body that uses NAF to capture the exceptional scenarios. Classical logic rules are very strict in nature, whereas using NAF and exception the ASP rules become flexible and can be applied in the absence of information. An HNS agent learns these exceptions after applying the rules and failing to get rewards. For an example, HNS agent learns the rule saying that "*apple* goes to the *fridge*", however, the agent fails when it tries the rule in the next episode for the "*rotten apple*". Then it learns that the feature - '*rotten*' is the exception to the above learned rule. This can be represented as below:

```
insert(X , fridge) <- apple(X), not ab(X).
ab(X) <- rotten(X).
```

Also, note that the no. of examples covered by the exception are always less than the no. of examples covered by the defaults and we have incorporated this constraint in the HNS agent's exception learning module.

WordNet based Generalization

An ideal RL agent should not only work great with the seen entities but also supposed to work well with the unseen entities or out-of-distribution (OOD) data. To achieve this, policy generalization is the utmost important feature that an ideal RL agent should possess. In our work, the TWC games are designed in such a way that the agents are tested on OOD entities that are not seen before, however, they are similar to the training data. So, the learned policies as logic rules will not work on the unseen objects. For an example, the rule *"insert(X, fridge) <- apple(X)"* can not work on another fruit such as *"orange"*. To tackle this, we lift the learned policies using WordNet's (Miller 1995) hypernym-hyponym relations to get the generalized rules. Figure 4 illustrates a rule generalization example from WordNet. The motivation behind this comes from the way humans do the tasks in their day-to-day life. For an example, if we know a *dirty shirt* goes to the *washing machine* and we have seen a *dirty pant*.

Algorithm 1: Generalized Rule Generation using WordNet

Input: E: Examples (*States, Actions, and Rewards*)
Output: R_G: Generalized Rules Set

procedure GETGENERALIZEDRULES(E)
  $R_g \leftarrow \{\}$ ▷ initialization
  $Goals \leftarrow$ *getGoals(E)* ▷ get the list of goals similar to the ILP data preparation (described above)
  for each $g \in Goals$ **do**
    $E_g \leftarrow$ *getExamples(E, g)*
    $(E_g^+, E_g^-) \leftarrow$ *splitByRewards*(E_g)
    $(Hyp_g^+, Hyp_g^-) \leftarrow$ *extractHypernyms*(E_g^+, E_g^-) ▷ get the hypernyms from WordNet
    $r_g \leftarrow$ *getBestGeneralization*$(E_g^+, E_g^-, Hyp_g^+, Hyp_g^-)$ ▷ uses entropy based information gain formula
    $R_g \leftarrow R_g \cup r_g$
  end for
  return R_g
end procedure

Then, normally we put the *dirty pant* into the *washing machine* as both are the type of *clothes* and *dirty*. The human mind is capable of doing the generalization in no time and that works really great.

On one hand, generalization gives better policies to work with the unseen entities and on the other hand, too much generalization leads to a drastic increment in false-positive results. To keep the balance, the agent should know how much generalization is good. For an example, "*apple* is a *fruit*", "*fruits* are *part of a plant*", and "*plants* are *living thing*". Now, if we apply the same rule that explains a property of an *apple* to all the *living things*, then it will become too erroneous. In this paper, we introduce a novel algorithm to dynamically generate the generalized rules exploring the hypernym relations from the WordNet. The algorithm is based on information gain calculated using the entropy of the positive and negative set of examples (collected by the agent). The illustration of the process is given in the Algorithm 1. The algorithm takes the collected set of examples and returns the generalized rules set. First, similar to the ILP data preparation procedure, the *goals* are extracted from the examples. For each *goal*, examples are split into two sets - E^+ and E^-. Next, the hypernyms are extracted using the hypernym-hyponym relations of the WordNet ontology. The combined set of hypernyms from (E^+, E^-) gives the *body* predicates for the generalized rules. Similar to the ILP (discussed above) the *goal* will be the *head* of a generalized rule. Next, the best generalized rules are generated by calculating the max information gain between the hypernyms. Information gain for a given clause is calculated using the below formula (Mitchell 1997) —

$$IG(R,h) = total * (log_2 \frac{p_1}{p_1+n_1} - log_2 \frac{p_0}{p_0+n_0}) \quad (1)$$

where h is the candidate hypernym predicate to add to the rule R, p_0 is the number of positive examples implied by the rule R, n_0 is the number of negative examples implied by the rule R, p_1 is the number of positive examples implied by the rule $R+h$, n_1 is the number of negative examples implied

			Easy		Medium		Hard	
			#steps	N. Score	#steps	N. Score	#steps	N. Score
IN	Text Only		15.12 ± 1.95	0.91 ± 0.03	33.17 ± 2.76	0.83 ± 0.04	47.68 ± 2.43	0.6 ± 0.05
	Best Rules		17.72 ± 3.18	0.92 ± 0.02	46.45 ± 1.85	0.48 ± 0.12	38.33 ± 1.7	0.84 ± 0.03
	All-Inclusive Rules		17.39 ± 3.01	0.93 ± 0.04	46.7 ± 2.14	0.42 ± 0.12	37.66 ± 0.93	0.88 ± 0.01
	Generalized Rules	Exhaustive	12.86 ± 3.04	0.91 ± 0.04	29.9 ± 3.16	0.65 ± 0.06	30.44 ± 0.87	0.95 ± 0.03
		IG (Hyp. Lvl. 2)	10.59 ± 1.3	0.95 ± 0.02	22.57 ± 1.04	0.77 ± 0.07	30.46 ± 0.74	0.87 ± 0.01
		IG (Hyp. Lvl. 3)	9.55 ± 2.34	0.96 ± 0.02	25.34 ± 2.86	0.76 ± 0.03	33.54 ± 1.47	0.91 ± 0.03
OUT	Text Only		16.66 ± 1.74	0.92 ± 0.03	37.3 ± 3.45	0.73 ± 0.06	50.00 ± 0.0	0.3 ± 0.04
	Best Rules		21.17 ± 2.3	0.86 ± 0.05	44.26 ± 4.09	0.47 ± 0.1	45.32 ± 2.36	0.57 ± 0.05
	All-Inclusive Rules		21.19 ± 0.87	0.84 ± 0.06	46.36 ± 1.52	0.42 ± 0.08	44.25 ± 0.42	0.63 ± 0.01
	Generalized Rules	Exhaustive	14.65 ± 2.18	0.91 ± 0.05	37.07 ± 2.09	0.63 ± 0.06	41.52 ± 1.12	0.83 ± 0.02
		IG (Hyp. Lvl. 2)	15.08 ± 1.2	0.91 ± 0.02	40.63 ± 3.03	0.57 ± 0.06	42.18 ± 0.66	0.79 ± 0.01
		IG (Hyp. Lvl. 3)	12.72 ± 1.22	0.92 ± 0.02	37.38 ± 3.09	0.64 ± 0.09	43.16 ± 2.83	0.78 ± 0.03

Table 1: Generalization results for within distribution (IN) and out-of-distribution (OUT) games

by the rule $R + h$, $total$ is the number of positive examples implied by R also covered by $R + h$. Finally, it collects all the generalized rules set and returns. Please note that this algorithm only learns the generalized rules which are used in addition to the rules learned by ILP and exception learning (discussed earlier) are the same for both cases.

Experiments

With the help of TWC framework (Murugesan et al. 2021), we generate a set of games with 3 different difficulty levels - (i) *easy level:* that contains 1 room with 1 to 3 objects; (ii) *medium level:* that contains 1 or 2 rooms with 4 or 5 objects; and (iii) *hard level:* a mix of games with a high number of objects (6 or 7 objects in 1 or 2 rooms) or high number of rooms (3 or 4 rooms containing 4 or 5 objects).

In our work, we want to show that if an RL agent uses symbolic and neural reasoning in tandem, then the performance of that agent increases drastically in the text-based games. So as a baseline model, we choose the text-only neural agent (Murugesan et al. 2021) that uses encoded history of observation to select the best action. For this paper, as our main focus is on the symbolic reasoning, we kept the text-only agent same in all of our experiments and tested on five different symbolic settings. Please note that out of these five settings two uses only ILP - *best* and *all-inclusive* rules, and other three uses ILP + WordNet based generalization. Below are the details of all the settings -

Best Rules: A feature of any ILP algorithm is to learn minimum amount of rules that covers maximum examples. To achieve this, whenever there are two or more rules having same information gain, the ILP selects one of them randomly. This is our best rule settings.

All-Inclusive Rules: Here, the ILP return all the rules that have same information gain instead of choosing randomly.

Exhaustive Rule Generalization: This setting lifts the rules exhaustively with all the hypernyms up to WordNet level 3 from an object or in other words select those hypernyms of an object whose path-distance with the object is ≤ 3.

IG based generalization (Hypernym Level 2): Here, the agent uses the rule generalization algorithm (algorithm 1). It takes WordNet hypernyms up to level 2 from an object.

IG based generalization (Hypernym Level 3): Similar to the above setting, here we provide hypernyms upto level 3.

Table 1 shows the comparison results of all the 5 settings along with the baseline model (text-only agent). Following (Murugesan et al. 2021), we compared our agents in two different test sets - (i) *IN distribution:* that has the same entities as the training dataset, and (ii) *OUT distribution:* that has new entities, which have not been included in the training set. In the table, we report the number steps taken by the agent (lower is better) and the normalized scores (higher is better). Note that, in all the settings, agents are trained using 100 episodes with 50-steps maximum.

Qualitative Studies

The agent with *IG based generalization (hypernym Level 3)* performs better than the others in the *easy* and *medium* level games, whereas *exhaustive generalization* works well in the *hard* games. This shows that one one side, the exhaustive generalization works slightly better in the environment where the entities and rooms are more and that needs more exploration. On another side, IG-based generalization works efficiently when the agent's main task is to select appropriate locations of different objects. In the *easy* and *medium* games, the *best rules* and the *all-inclusive rules* (without generalization) perform poorly in comparison with the baseline model. This indicates - only learning rules without generalization for simple environments leads to bad action selection especially when the entities are unseen. The out-distribution results for the medium games are not up to the mark. Further studies on this show that this happens when the OOD games have different but similar locations (*clothes-line* vs. *clothes-drier*) along with different objects in the environment. Generalization on the location gives very noisy results (increases false-positive cases) as they already belong to a higher level in the WordNet ontology. The solution for this requires another way of incorporating commonsense knowledge to the agent and we addressed more on this in the future work section.

Confidence score of a learned rule is crucial as it projects the reliability of a rule. In our work, the rules are learned in two different ways - using ILP and generalizing them later with WordNet. So, HNS agent calculates the confidence

```
2.0000  put(X, shelf) :-  flour(X).                    (1)
2.0000  put(X, shelf) :-  peanut_oil(X).               (2)
1.9626  insert(X, trash_can) :-  used(X).              (3)
1.8000  put(X, hat_rack) :-  headgear(X).              (4)
1.5432  insert(X, fridge) :-  dairy_product(X).        (5)
0.9519  put(X, shelf) :-  seasoner(X).                 (6)
0.6000  insert(X, fridge) :-  structure(X).            (7)
```

Figure 5: Example of Rule's Confidence Scores (medium level games)

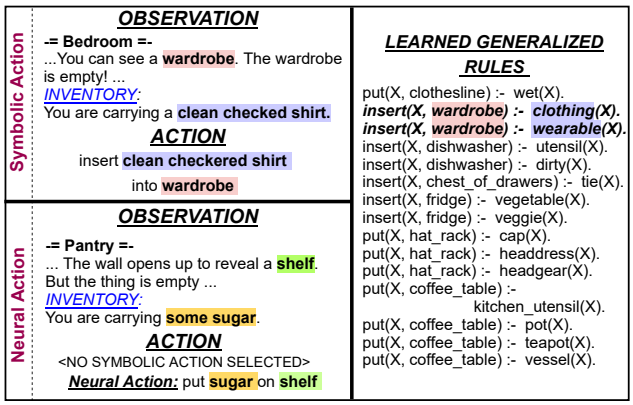

Figure 6: Examples from TWC game, showing the learned rules (right hand side) along with the *observations* and *action selection* (Symbolic vs. Neural)

score based on two separate components - (i) coverage of a rule while playing, and (ii) the distance between two nodes (entity and the hypernym) in the WordNet ontology. The coverage of a rule can be calculated by the number of times the rule gives a positive reward after applying divided by the total number of times the rule has been used. Whereas the WordNet based confidence gives a higher score to the parent of an entity than its grandparent and also the direct ILP learned rules get 1.0 (maximum score in this component). Cumulative maximum score of two components is 2.0 (1.0 from each component). Note that the *Best Rules* and *All-Inclusive Rules* do not get the second component score as they do not possess any generalization (their maximum score is 1.0). Figure 5 shows a snippet of a learned set of rules with different confidence scores. Rule (1) and (2) got the maximum score of 2 which means they were learned from the ILP and always got positive rewards when it was applied. Rule (3), (4), (5), and (6) got better to medium scores and we can clearly see why. For example, we normally put *dairy products* to the *fridge* or *head-gears* to the *hat-rack*. Rule (7) did not get a good score as it is a very generalized not-reliable rule saying - things which has a *structure* goes to the *fridge*.

Figure 6 illustrates an example, showing how a hybrid neuro-symbolic agent plays the TWC games. HNS agent learns the symbolic rules using ILP + WordNet based rule generalization. A snippet of the learned rules is given on the right-hand side of the figure. To generate action using the symbolic module, the agent first extracts the information about the entities from the observation and the inventory information. Then, this information are represented as ASP facts along with the hypernyms of the objects. Next, it runs the ASP solver - the s(CASP) engine to get a set of possible actions and select an action based on the confidence scores. The top-left section of figure 6 shows how a symbolic action that has been selected by matching the object (i.e., clean checked shirt) with the rule set (highlighted in the right). Here, the solver finds the location of the *shirt* in *wardrobe* as *'clothing'* and *'wearable'* are the hypernyms of the word - *shirt*. So, whenever the agent does not find a symbolic action or fails to get reward from the suggested symbolic action, then it moves to the neural agent to get an action (as a fallback). The bottom-left of figure 6 shows that the location of 'sugar' is not covered by any rules, so the neural agent selects an action. In a similar way, figure 7 shows two different scenarios of the game - (a) symbolic agent works and neural fails, and (b) neural agent works and symbolic fails. This is how the symbolic and neural modules of the HNS agent work in tandem.

After doing extensive studies of *exhaustive* vs. *IG based* rule generalization, we found that in exhaustive generalization, the rules are lifted without considering the reliability of the generalized rules. By adding exceptions, the exception learning module takes care of it and tunes the low reliable rules to powerful rules that work with different other objects. Oppositely, for an object, due to the presence of the exceptional scenario (negative example) for a particular hypernym, the information gain becomes lower for that hypernym in comparison with others. So, IG based generalization does not learn that rule. In another sense, the IG based generalization only learns the best hypernyms and that is much more effective when the agent has seen many objects and their correct locations (more exploration). The *easy* and *medium* game results depicts the same. Figure 8 shows two snippets of learned rules from the *exhaustive generalization* and the *IG based generalization* for the entities whose location is *shoe cabinet*. Both of the generalization methods learn almost the same set of rules, except the exhaustive one learning two more rules (rule no. 3 & 5). The reason behind is the comprehensive generalization with the exception learning. So, in exhaustive generalization first, the rules are lifted with the hypernyms without considering the negative examples and later exception learner adds that negative example as an exception to fit all the examples with the learned rules. So, 'scarf', which is a type of 'covering' but does not go to a 'shoe cabinet', learned as an exception (rule no. 5) to the rule no. 3. In IG based generalization, due to the presence of a negative example (the 'scarf'), the information gain for 'covering' goes down in comparison with the other hypernyms. This ends up in not learning the rule for 'covering' in IG based generalization.

Related Works

Text-based reinforcement learning: TBGs emerged as promising environments for studying grounded language

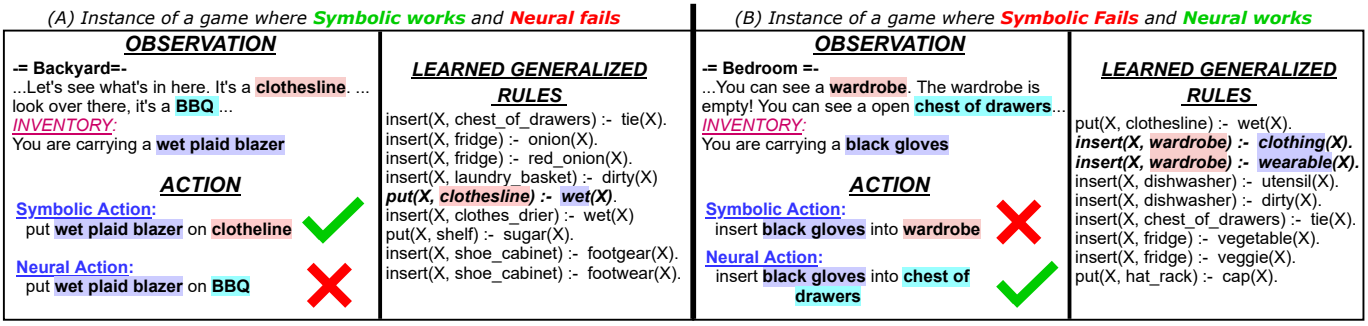

Figure 7: Two instances of TWC games, showing two different scenarios - (A) when symbolic agent works and neural agent fails, and (B) when symbolic agent fails and neural agent works

Generalized Rules: Exhaustive

1. insert(X, shoe_cabinet) :- footgear(X).
2. insert(X, shoe_cabinet) :- footwear(X).
3. insert(X, shoe_cabinet) :- covering(X), not ab1(X).
4. insert(X, shoe_cabinet) :- shoes(X).
5. ab1(X) :- scarf(X).

Generalized Rules: IG-based (Level 3)

1. insert(X, shoe_cabinet) :- footgear(X).
2. insert(X, shoe_cabinet) :- footwear(X).
3. insert(X, shoe_cabinet) :- shoes(X).

Figure 8: Generalized Rule Examples (medium level games)

understanding and have drawn significant research interest. Recently, Zahavy et al. (2018) introduced the Action-Elimination Deep Q-Network (AE-DQN), which learns to predict invalid actions in the text-adventure game *Zork*. Côté et al. (2018) designed `TextWorld`, a sandbox learning environment for training and evaluating RL agents on text-based games. On the same line, Murugesan et al. (2021) introduced *TWC*, a set of games requiring agents with commonsense knowledge. The *LeDeepChef* system (Adolphs and Hofmann 2019) achieved good results on the *First TextWord Problems* (Trischler, Côté, and Lima 2019) by supervising the model with entities from `FreeBase`, allowing the agent to generalize to unseen objects. A recent line of work learns symbolic (typically graph-structured) representations of the agent's belief. Notably, Ammanabrolu and Riedl (2019) proposed *KG-DQN* and Adhikari et al. (2020) proposed *GATA*. We also use graphs to model the state of the game.

Symbolic rule learning approaches: Symbolic rule learning using inductive logic programming has a long history of research. After the success of ASP, many works have been emerged that are capable of learning non-monotonic logic programs, such as FOLD (Shakerin, Salazar, and Gupta 2017), ILASP (Law, Russo, and Broda 2014), XHAIL (Ray 2009), ASPAL (Corapi, Russo, and Lupu 2011), etc. FOLD algorithm can learn default theories with exception from a large set of data using background knowledge. Whereas, the others resort to an exhaustive search for the hypothesis in the data and that makes them non-scalable on the real-world datasets. Also, there are not many efforts that have been taken on lifting the rules to their generalized version and then learning exceptions. Also, they do not perform well on noisy datasets. To tackle these issues, there are efforts in combining ILP with differentiable programming (Evans and Grefenstette 2018; Rocktäschel and Riedel 2017). However, it requires lots of data to be trained on. In our work, we use a simple information gain based inductive learning approach, as the HNS agent learns the rules after each episode with a very small amount of examples (sometimes with zero negative examples). And, to make the rule non-monotonic, the HNS agent learns the exception separately from the game experience and updates the previously learned rules. Also, we assume TWC games data are not noisy. In future, with more harder games that have uncertainty and noise, we will investigate more on differentiable ILP algorithms and will see how we can apply them in the RL environment along with exception learning. There are recent works in this direction (Jiang and Luo 2019), however, they do not focus on learning the non-monotonic programs, which is crucial in a partially observable world.

Future Works and Conclusion

For the TBGs, this project is a proof of concept to show how a symbolic agent can work together with a neural agent in an RL environment. So, in our future work, we plan to focus more on the *action selector* module that chooses the best action between the symbolic agent vs. the neural agent. From the current version, we have learned that too much dependency on the symbolic agent and heavy generalization is not always good. So next, we mainly want to work on the process of switching between neural agent vs. symbolic agent. We need to find a better way of choosing between the actions generated by the neural and the symbolic agents. One idea will be to use the neural-agent generated probability distributions over the admissible actions to filter out the lower confident admissible actions and use the symbolic agent to choose an action from the rest. Another approach will be calculating the rule's confidence based on external commonsense knowledge. We hope, with these updates, our

HNS agent will perform better not only on TWC but also on much harder text-adventure games like ZORK.

References

Adhikari, A.; Yuan, X.; Côté, M.-A.; Zelinka, M.; Rondeau, M.-A.; Laroche, R.; Poupart, P.; Tang, J.; Trischler, A.; and Hamilton, W. L. 2020. Learning dynamic knowledge graphs to generalize on text-based games.

Adolphs, L.; and Hofmann, T. 2019. LeDeepChef: Deep Reinforcement Learning Agent for Families of Text-Based Games. *ArXiv*, abs/1909.01646.

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.

Arias, J.; Carro, M.; Chen, Z.; and Gupta, G. 2020. Justifications for Goal-Directed Constraint Answer Set Programming. *arXiv preprint arXiv:2009.10238*.

Arias, J.; Carro, M.; Salazar, E.; Marple, K.; and Gupta, G. 2018. Constraint answer set programming without grounding. *TPLP*, 18(3-4): 337–354.

Corapi, D.; Russo, A.; and Lupu, E. 2011. Inductive logic programming in answer set programming. In *International conference on inductive logic programming*, 91–97. Springer.

Côté, M.-A.; Kádár, A.; Yuan, X.; Kybartas, B.; Barnes, T.; Fine, E.; Moore, J.; 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.

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

Gelfond, M.; and Kahl, Y. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

Gelfond, M.; and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, 1070–1080.

Hausknecht, M.; Loynd, R.; Yang, G.; Swaminathan, A.; and Williams, J. D. 2019. Nail: A general interactive fiction agent. *arXiv preprint arXiv:1902.04259*.

Jiang, Z.; and Luo, S. 2019. Neural logic reinforcement learning. In *International Conference on Machine Learning*, 3110–3119. PMLR.

Law, M.; Russo, A.; and Broda, K. 2014. Inductive learning of answer set programs. In *European Workshop on Logics in Artificial Intelligence*, 311–325. Springer.

Miller, G. A. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11): 39–41.

Mitchell, T. 1997. Machine learning. *McGraw Hill series in computer science. McGraw-Hill*.

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

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. In *Thirty Fifth AAAI Conference on Artificial Intelligence*.

Narasimhan, K.; Kulkarni, T.; and Barzilay, R. 2015. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.

Ray, O. 2009. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3): 329–340.

Rocktäschel, T.; and Riedel, S. 2017. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040*.

Shakerin, F.; Salazar, E.; and Gupta, G. 2017. A new algorithm to automate inductive learning of default theories. *Theory and Practice of Logic Programming*, 17(5-6): 1010–1026.

Trischler, A.; Côté, M.-A.; and Lima, P. 2019. First TextWorld Problems, the competition: Using text-based games to advance capabilities of AI agents.

Zahavy, T.; Haroush, M.; Merlis, N.; Mankowitz, D. J.; and Mannor, S. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 3562–3573.