Learning Generalizable Behavior via Visual Rewrite Rules

Yiheng Xie[*1], **Mingxuan Li**[*2], **Shangqun Yu**[*1], **Michael L. Littman**[1]

[1]Department of Computer Science, Brown University,
[2]Department of Computer Science, Columbia University
{yiheng_xie, shangqun_yu, michael_littman}@brown.edu, ml@cs.columbia.edu

Abstract

Though deep reinforcement learning agents have achieved unprecedented success in recent years, their learned policies can be brittle, failing to generalize to even slight modifications of their environments or unfamiliar situations. The black-box nature of the neural network learning dynamics makes it impossible to audit trained deep agents and recover from such failures. In this paper, we propose a novel representation and learning approach to capture environment dynamics without using neural networks. It originates from the observation that, in games designed for people, the effect of an action can often be perceived in the form of local changes in consecutive visual observations. Our algorithm is designed to extract such vision-based changes and condense them into a set of action-dependent descriptive rules, which we call "visual rewrite rules" (VRRs). We also present preliminary results from a VRR agent that can explore, expand its rule set, and solve a game via planning with its learned VRR world model. In several classical games, our non-deep agent demonstrates superior performance, extreme sample efficiency, and robust generalization ability compared with several mainstream deep agents.

Introduction

While deep reinforcement-learning agents have achieved impressive performance in Atari games (Espeholt et al. 2018; Badia et al. 2020; Schrittwieser et al. 2020; Hafner et al. 2021), the vast amount of data they require for training and their limited generalization ability preclude us from extending these approaches to more meaningful and challenging real world tasks. The sample efficiency problem has been long existed in the reinforcement-learning (RL) literature due to the challenges of the "curse of dimensionality" (Bellman 1966). Even deep RL agents must contend with the hardness of learning good representations (Glorot and Bengio 2010; Laskin et al. 2020). For the generalization issue, studies (Cobbe et al. 2020, 2019; Witty et al. 2018) have demonstrated that deep RL agents can easily memorize and overfit to training environments instead of truly understanding the dynamics that underpin the task. As a consequence, even simple variations in an environment that human learners barely even notice can undermine the performance of a deep RL agent.

Though there are extensive studies trying to address the aforementioned problems (Lee et al. 2020; Cobbe et al. 2020; Laskin et al. 2020), most continue to adopt an end-to-end learning framework. We show that by disentangling representation learning from policy learning, problems can successfully be solved in parts.

Specifically, human game playing priors (Dubey et al. 2018; Tsividis et al. 2017) support the learning of high-level factored rules built upon the concept of an object (Diuk, Cohen, and Littman 2008), which support planning in a wide variety of related tasks. Motivated by the observation that it is possible to reason (Furnas 1991) and compute (Ackley 2018) with picture-to-picture rules, we examine using such an approach to learn environmental dynamics in RL. We present *visual rewrite learning*, a simple learning algorithm that captures local action-dependent visual rewrite rules (VRRs) of the environment. We also present a model-based agent framework integrating VRR and planning.

We demonstrate the effectiveness of the VRR agent in a series of publicly available environments. Our VRR agent outperforms state-of-the-art model-free and model-based deep RL agents in tests of generalization and uses significantly fewer training samples. To the best of our knowledge, we are the first to beat deep networks with a non-deep approach in these environments.

Background

In this work, we use Markov Decision Processes (MDPs) to model an agent interacting with its environment while solving a task. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ where state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, reward $r \in \mathcal{R}$, state transition described by $\mathcal{P}(s_{t+1}, r_t | s_t, a_t)$ and discount factor $\gamma \in [0, 1)$. The RL agent's goal is to maximize discounted reward $\sum_{t=0}^{\infty} \gamma^t r_t$.

In model-based reinforcement learning, an agent learns an environment model composed of a state-transition model and a reward model. It then uses the learned model to navigate through the environment, gathering reward. Dyna (Sutton 1991), which integrates model-free learning with generated data from a learned model, is an example of such an approach. Deep RL, with its focus on visual observations, was slow to incorporate model-based learning. But,

[*]These authors contributed equally.

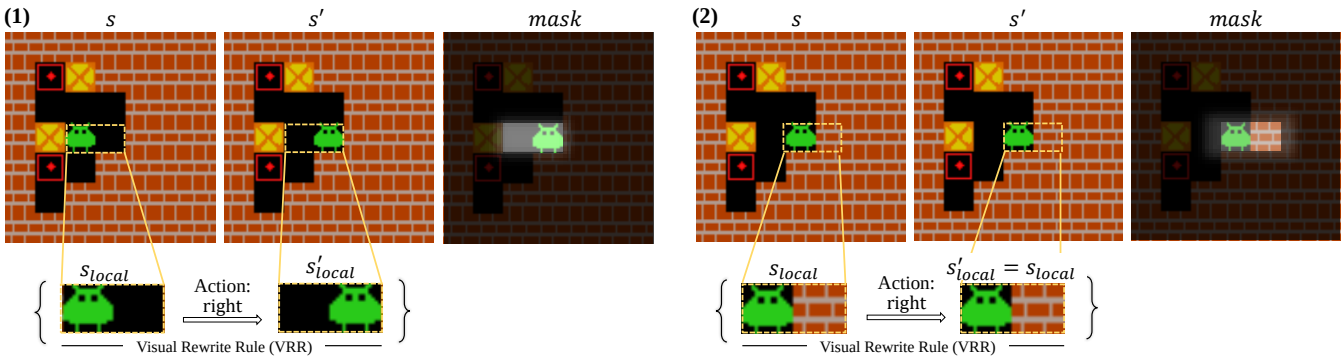

Figure 1: Example Visual Rewrite Rules (VRRs) in Sokoban. State change can be decomposed into a static non-local component and a dynamic local component near the agent. The local region is highlighted by mask. (1) The Sokoban agent (green) moves right onto the empty space, in response to action right. (2) Agent cannot move into the brick wall with action right, instead no state change occurs ($s^t = s^{t+1}$).

by now, deep world models are being used to generate agent-simulated trajectories for training model-free agents (Ha and Schmidhuber 2018; Racanière et al. 2017; Hamrick et al. 2017; Kaiser et al. 2020; Hafner et al. 2021) or can be integrated with lookahead planning (Schrittwieser et al. 2020; Goldwaser and Thielscher 2020; Hamrick et al. 2021). A major challenge is that the quality of predicted trajectories typically degrades quickly as small errors compound as trajectories are rolled out (Janner et al. 2019; Asadi et al. 2019). In addition, overfitting hampers the application of a deep-learned world model to predictions on out-of-distribution (training) states, even when they share dynamics. We argue that without an organized, structural way of learning and using the learned environment dynamics, it is nearly impossible to verify if the world model has grasped the essential knowledge for solving the task and therefore applying to novel situations. We investigate a more structured non-deep-network-based approach for learning a world model.

Our approach can be viewed as a form of abstraction (Li, Walsh, and Littman 2006; Konidaris 2019) in which observations are decomposed into patches of pixels observed to change together. Two main forms of abstraction in RL are state abstraction and action abstraction. The former maps the original larger state space into a smaller state space while preserving the essential properties for solving the task (Dean and Givan 1997; Jong and Stone 2005; Jiang, Kulesza, and Singh 2015; Abel et al. 2018). The latter approach aggregates the atomic actions into new units enabling high level planning and skill reuse (Sutton, Precup, and Singh 1999; Konidaris and Barto 2009; Sharma et al. 2020).

Visual Rewrite Rules

We introduce Visual Rewrite Rules (VRRs) as a new representation for describing game-environment dynamics. In contrast to deep neural networks, VRRs are an intuitive, elegant, yet robust method to model environments. In this section, we first offer an intuitive explanation of how VRRs describe state transitions. Then, we introduce the formal definition of visual rewrite rules, along with their strengths and limitations. We then explain how to learn VRRs from experience data. Finally, we present subtleties to implementing VRRs with pure-vision observations.

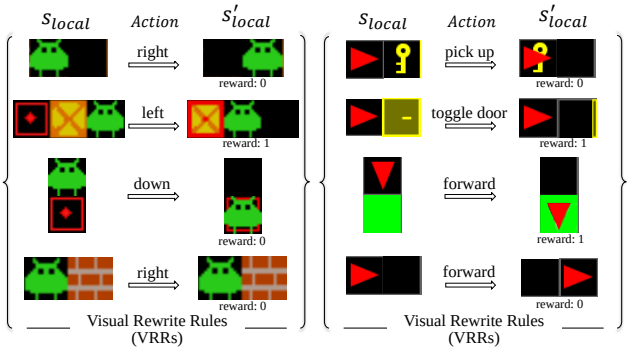

Figure 2: Example Visual Rewrite Rules in Sokoban (left) and MiniGrid (right).

State Transitions as VRRs

Intuitively, one may think of a VRR state transition as updating a subset of the state vector representing the state according to the action, while the rest of the state vector remains the same. The dynamic, changing component is "rewritten" based on the outcome of the action, hence the name "visual rewrite". Fig. 1 shows an example of modeling the state transitions in Sokoban using VRRs. In the first graph of Fig. 1, the agent moves from one blank cell to another blank cell under action "move right". All other grid cells remain the same between the two time steps. The two cells are "rewritten" with the outcome of the action. In the second graph of Fig. 1, the agent tries to move right but bumps into a wall. There is no state change. The reason that the same action results in different transitions is explained by the two cells located around the agent, the difference being the presence and absence of a wall.

In many games, the effects of agent actions is localized near the agent. Hence, VRRs decompose the state transition

into a *local* component near the agent, and a static non-local component. We use **Visual Rewrite Rules (VRRs)** to model such state transitions. Each VRR describes the shape of the local component of the state vector, what the local component looks like, how the local component changes under the influence of actions, and the reward. Fig 2 shows a selection of VRRs from the Sokoban and MiniGrid games.

Formally, each VRR is defined as $\mathcal{F} : (s[\text{mask}], a) \rightarrow (s'[\text{mask}], \text{reward})$ mapping the current local component of s and action a to the local component after the transition and reward. We define mask as an indicator function for the shape of the local component of the state vector, specifically $s_local = s[\text{mask}]$.

When a state transition occurs (as in Figure 1, left), mask indicates the position where the pixel values have changed between s and s': $\text{mask} = where(s \neq s')$. The set of VRRs are stored as a **dictionary**, where the key-value pair is $((s[\text{mask}], a), (s'[\text{mask}], \text{reward}))$.

VRR World Model

VRRs form a vision-based world model. At inference time, a VRR can be "stamped" onto a new state vector to predict state transitions. Algo 1 describes the VRR world model.

Algorithm 1: *world_model*. VRRs as a world model.

Input : State vector s. Action a. Rule set $\text{VRR}s$. Agent position agent.pos.
Output: State vector s'. reward. status.

$\text{mask} = where(s \neq s')$;
// Compute local component.
$s_{\text{local}} = s[\text{mask}]$;
for $s_{rule} \in \text{VRR}s[a]$ **do**
  $tmpS_{rule} = s_{rule}$ shifted to agent.pos;
  Compare $tmpS_{rule}$ with current s_{local};
  // Found a match.
  if $tmpS_{rule} = s_{\text{local}}$ **then**
    $(s'_{\text{local}}, \text{reward}) = \text{VRR}s[a, s_{rule}]$;
    Update s with s'_{local} to obtain s';
    $\text{status} =$ known rule;
    return $s', \text{reward}, \text{status}$;
  end
end
// No matching local states found.
$\text{status} =$ unknown rule;
return status;

Given a state vector s and agent position agent.pos, we compare the local region centered around the agent s_{local} to each local component s_{rule} stored in the $\text{VRR}s$ dictionary keys. If there is a match, we use the tuple of agent action a and s_{local} to find the resulting state transition s'_{local}, and reward r. If there's no matching dictionary entry, this indicates a previously unseen local component of the state vector.

By the nature of most games, the number of VRRs needed for constructing a perfect VRR world model is small, since the game dynamics is usually composed of basic components such as movements, pick up, and toggle. We leverage this sparsity of game rules to learn an efficient set of VRRs. VRR is also invariant to rotation and translation, since we only focus on the local action effect while ignoring the static components. This leads to robust and generalizable world model, which we show in the experiments section.

Learning VRRs

From the definition of VRR, the most straightforward way of learning it is by contrasting s with s', and recording the pixels that are affected by the action. Algo 2 illustrates this basic idea of learning VRRs.

Algorithm 2: Learning VRR from state difference.

Input : Game state vector (grid) from two adjacent time steps s, s'. Action a. Global Rules set $\text{VRR}s$

// Mask out the unchanged region.
$\text{mask} = (s \neq s')$;
// Store the change in dictionary.
$\text{VRR}s[a, s[\text{mask}]] = s'[\text{mask}]$;

But there is subtlety here. Normally, one would expect that every action leads to some changes in the visual representation of the environment. However, it is very common that certain state–action pairs result in no state change at all. For example, in Sokoban, if the agent tries to push the box into a wall, s' is exactly the same as s (Fig. 1 part 2). In such cases, $s = s'$ leads to an empty mask. That is, $\text{mask} = where(s \neq s') = \emptyset$, $s'_{\text{local}} = s_{\text{local}} = \emptyset$.

A naive solution is to store the entire game state, but this is impractical, and defeats our goal of building a compact, canonical rule set. Therefore, an appropriate inductive bias is necessary for choosing *where* to look for evidence that could explain the absence of state change (for example, the wall in front of the agent).

We extend Algo 2 into Algo 3, where the "if" statement explains why a state transition occurred, while the "else" clause seeks to explain why a state didn't change under certain actions. In the latter case, the *local* region is where the VRRs *expect* changes to happen.

For example, a previously learned VRR expects the agent to move forward with action "forward", where the input of this VRR has a two-cell local component—the agent, and the empty space in front of it. When such state transition does not occur, we first mask out the same local component and notice that the empty space in s_{rule} is now a wall, which explains the absence of change. In the case when no state change is ever observed with a certain action, VRRs simply assume this action does not lead to state changes. When this is proven to be wrong, VRR will record the local component change, and proceed as normal.

With this strategy, VRR can now handle the situation when null state transition happens.

VRR in Practice

Agent Position In previous sections, we assume that the agent is not only given the observation but also knows its

Algorithm 3: *learn_vrr*. Learning VRRs with expectation correction.

Input : State vectors s, s'. reward. Action a. Rules set VRRs. Agent position agent.pos.
Output: Updated rules set VRRs.

mask $= (s \neq s')$;
if mask $\neq \emptyset$ **then**
  VRR$s[a, s[\text{mask}]] = (s'[\text{mask}], \text{reward})$;
else
  `// Given no state change occurred, we take the existing VRRs' local scope as mask.`
  mask = intersection of all s_{local} from current VRRs;
  mask$'$ = mask shifted to agent.pos;
  VRR$s[a, s[\text{mask}']] = (s[\text{mask}'], \text{reward})$;
end

own position. In other words, the controllable object, which is the agent, is differentiated from other game objects. This information is crucial because the locality of visual changes is defined relative to the agent's position. Though identifying the controllable object seems to be intuitive for us, it's not so obvious for an algorithm without any prior knowledge that human players do.

To relax this assumption, we use a principled method to identify the agent in the game. We define the agent as the object that exhibits the ability to independently move as a result of input actions. Other passive game objects, such as boxes in Sokoban, can only move as a result of the agent's actions. Hence, state changes always occur near the agent, since the object exhibiting agency is always present to produce those state changes. We leave handling more complex games with independent moving objects (Frogger, Space Invaders, etc.) for future work.

Object Centric Representation During implementation, we adopt a discrete-space and object-centric representation of the state space. Given a rendered game observation in pixels, we discretize the screen into squared grid cells. The grid size is *not* assumed to be given. Instead, we search for the minimal sprite unit by picking the grid size that yields the least number of object types while also being semantically meaningful (Fig. 3). An erroneous grid size (for example slicing between the ground truth grids) would result in an explosion of the number of object types and the number of rules. At last each distinct object type is assigned with an unique id. This approach transforms the visual observation into a compact yet expressive object-oriented state vector that also preserves the spatial topology.

Although we only show results for games whose observation space is discrete and two-dimensional, in principle, VRRs are compatible with games that do not fit into a regular grid, such as balls and paddles in Pong. Additionally, the idea of VRR can also model stochastic state transitions easily by extending the dictionary keys to a *distribution* of possible future states. We reserve these topics for future work.

A VRR Agent

Algorithm 4: VRR agent.

Input : Game environment env. Rules set VRRs.
while *not done* **do**
  BFS from current state, using VRR world model;
  `// Algo. 1`
  BFStree = empty tree, where nodes are states, edges are actions;
  `// BFS keeps track of reward, and status`
  if *BFS maximum reward* > 0 **then**
    actions =action sequence from root to winning state;
  else if status = *new rule* **then**
    `/* If agent cannot complete the game with current VRRs, explore new rules. */`
    actions =action sequence from root to novel state;
  else
    `/* No more new rules, games not winnable (such as a box in corner in Sokoban). */`
    agent gives up this round;
  for $a \in$ actions **do**
    $s, s', \text{reward} = \text{env}.step(a)$;
    `// Expand VRRs with Algo.3`
    VRRs = *learn_vrr*$(s, s', \text{reward}, a, \text{VRR}s, \text{agent.pos})$;
  end
end

A VRR agent is composed of a VRR world model described above and a compatible planning algorithm. We choose breadth-first search (BFS) for its simplicity, but VRR is trivially compatible with more sophisticated search algorithms such as Monte-Carlo Tree Search (Coulom 2006).

To learn the VRRs efficiently, a principled method for exploring the game is needed. As described in Algo 4, the agent first attempts to solve the game via BFS planning with the current set of learned VRRs. If BFS returns an action sequence that leads to the winning state, then the set of learned game rules is sufficient for the agent to solve the task. The

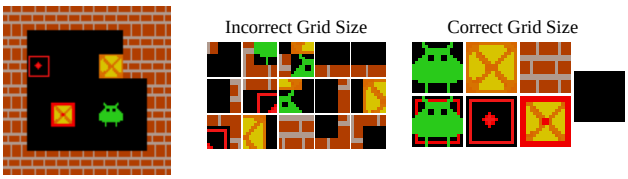

Figure 3: Correct grid size results in a minimal number of object types, while incorrect grid slicing results in a combinatorial number of object types.

agent then executes this action sequence to receive the reward and moves onto the next round without wasting any time. Conversely, if BFS terminates without finding the winning state, there must exist state transitions that the agent is yet to learn, that is, $\exists\,\text{game rule} \notin \text{VRR}s$.

The VRR world model is capable of indicating when it has encountered an unknown local component rewrite. That is, when a key is absent from the VRRs dictionary. In this circumstance, BFS planning will return an action sequence that guides the agent to explore this new transition. Fig. 4 illustrates such an example. If the agent has not yet interacted with the door, and thus cannot solve the game, the BFS will return action sequences that explore unknown state transitions. One of such action sequence will navigate the agent to the door, and then execute the action "toggle." The new rule describing door-opening will then be recorded in VRRs.

A notable exception here is Sokoban, in which irreversible behaviours exist. When agent pushes a box into a corner, it can never get that box out. VRR agent is able to identify such situations: neither is the game solvable, nor are there any unknown states to be explored. The agent will promptly give up the current round.

A VRR agent can learn game rules without any prior knowledge from scratch with extreme sample efficiency. At test time, VRR agent is capable of zero-shot generalization to new game levels, which we demonstrate in experiments. VRR agent learning is also amenable to lifelong learning, where it can learn a subset of game rules and update its knowledge in an online fashion as the game complexity increases. Moreover, devoid of any black box components, the VRR agent's learning and planning is completely interpretable, as shown in Fig. 4.

Experiments

In this section, we demonstrate the sample efficiency and generalization ability of our VRR agent in comparison with several deep RL agents: PPO (Schulman et al. 2017), IMPALA (Espeholt et al. 2018), and DreamerV2 (Hafner et al. 2021). With the exception of using the original implementation for DreamerV2, we adopt the Ray RLib (Liang et al. 2018) version for our baselines. The following environments are used for training and evaluation: MiniGrid (Chevalier-Boisvert, Willems, and Pal 2018) and gym-Sokoban environment (Schrader 2018). They are both procedurally-generated environments with varying game layouts and clearly-defined difficulty levels. See Fig. 5 for a brief description of the training environments and their variations. When reporting the average return of deep RL baselines, we show the mean and standard derivation from 3 independent runs. The average return is calculated as the rolling mean of the last 10 episodes. We will release our source code soon.

Sample Efficiency in Training

As described in Algo 4, we train the VRR agent from scratch with an initially empty rule set. By interacting with the game environment, the VRR agent gradually expands its rule set (as shown in Fig. 7, top). In Fig. 6, we show that the VRR agent requires orders of magnitude fewer environment steps than the deep baselines before converging. In Sokoban, the VRR agent also achieves higher return at convergence.

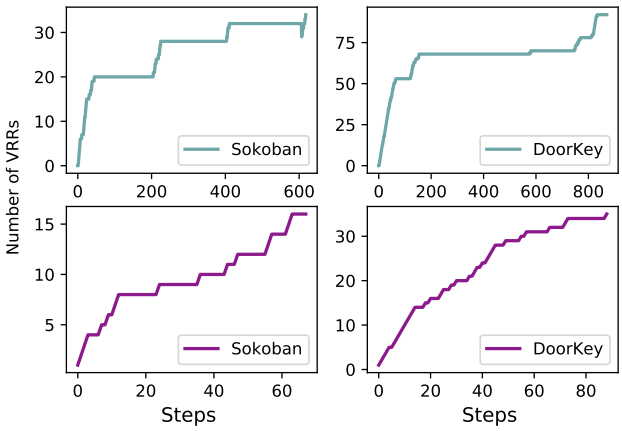

Figure 7: Number of VRRs learned during training as a function of training steps. Top: learning from scratch by interacting with the game environment. Bottom: learning from an extremely small set of human play data.

Game	Data Src.	Steps	Avg. Return	Avg. Steps
Sokoban	scratch	440	1.0	4.528
	human	81	0.96	4.239
DoorKey	scratch	1058	1.0	12.724
	human	89	1.0	10.249

Table 1: VRR agent trained on a small human play dataset achieves comparable performance at test time, while using an order of magnitude fewer training steps. Sokoban and DoorKey game board sizes are 7×7, and 6×6, respectively.

Minimal Guided Learning The sample efficiency of the VRR agent is explained by that the rule set captures only the novel and necessary local component transitions without redundancy. However, the sample efficiency is ultimately bounded by the exploration efficiency of the agent. The faster the agent discovers new game rules, the sooner it can solve the game.

To test the VRR agent's sample efficiency in the limit. We demonstrate that VRR agent can learn from an extremely small training set of under 100 steps (Fig. 7), while achieving identical performance as learning from scratch (Table 1). The dataset is collected from a human player, who solves a few levels of the game while demonstrating all the basic movements required to complete the task. Such a limited dataset is completely insufficient for training deep RL agents, for whom the required number of training steps before convergence is often 4-5 orders of magnitude larger. This resonates with our argument that by disentangling representation learning from policy learning, the sample efficiency problem will be greatly alleviated. So does the generalization problem, as we show below.

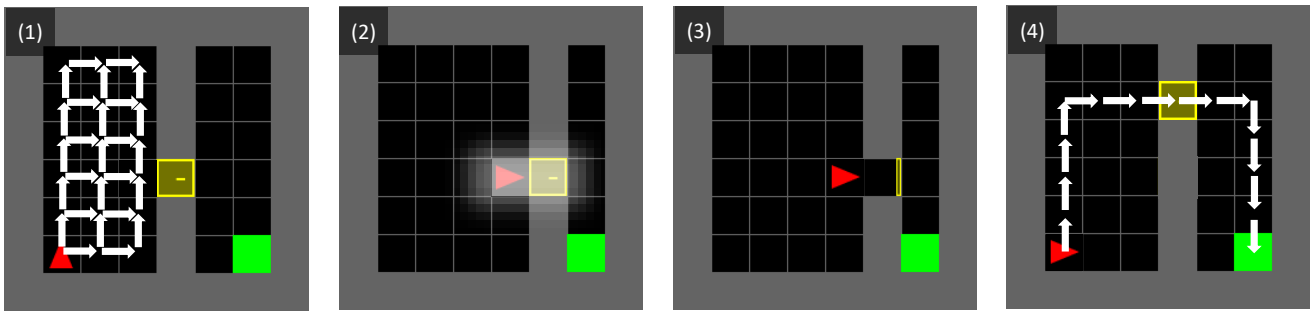

Figure 4: VRR agent explores new rules when its current knowledge is insufficient to solve the game. It balances exploration and exploitation leading to sample-efficient learning. For a live demo, see Sokoban experiments and DoorKey experiments.

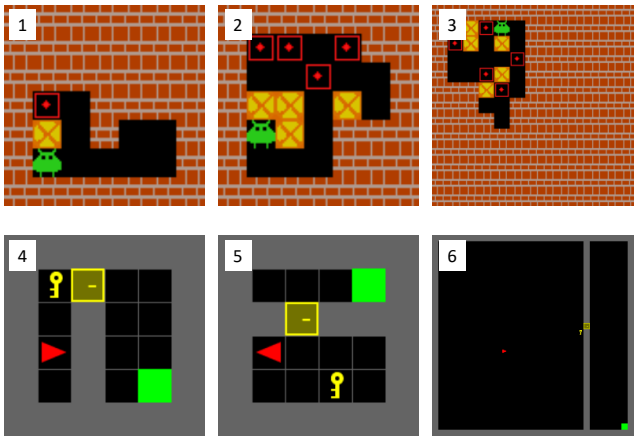

Figure 5: Game environments. 1) 7×7 Sokoban with 1 box (train), 2) 7×7 Sokoban with 4 boxes (test), 3) 13×13 Sokoban with 5 boxes (test), 4) 6×6 DoorKey (train), 5) 6×6 DoorKey rotated (test), 6) 32×32 DoorKey (test).

Generalization Experiments

Next, we test the zero-shot generalization performance of VRR agent. Since both DoorKey and Sokoban are procedurally generated, their underlying game rules remain unchanged regardless of game parameters. The game parameters we investigate include (Fig. 5):

- number of boxes in Sokoban,
- random rotation to initial states in DoorKey,
- game board size (such as 7×7 vs. 13×13).

We show that the VRR agent can solve more complex game levels with the essential rule set learned from the most basic level. For all zero-shot generalization experiments below, we train VRR and baseline agents on 7×7 Sokoban environment with 1 box, and 6×6 DoorKey environment.

Sokoban There are primarily two game parameters that affect the complexity of the game: number of boxes and board size. We test the performance of VRR agent in larger maps with the same number of boxes used during training. Table 2 shows that VRR agent performance is invariant to Sokoban board size, while baselines struggle to generalize to larger board sizes. Empirically, a larger map enables more complex layouts and larger search spaces, which makes it easier to trap the agent in irreversible situations. Furthermore, CNN-based networks are trained to a fixed resolution. When the spatial resolution of the observation changes, the convolutional feature scales accordingly, which severely degrades the performance. Conversely, VRR world model is only concerned with the *local* subset of state vector, and its performance is invariant to the overall resolution. The only limiting factor for VRR agent is the search algorithm's runtime, which is not problematic for 13×13 Sokoban.

We further test the agents' ability to solve in 2-, 3-, and 4-box Sokoban after being trained only on the 1-box Sokoban. Game board size is fixed to 7×7. Fig. 8 shows that VRR agent's performance degrades gracefully, and far outperforms baseline agents in the multiple-box Sokoban games. Note that the agent has never seen more than 1 box, and is disallowed to learn new rules. Notably, VRR agent not only accrues higher returns, but also solves each game episode with fewer steps. We accredit this advantage to the explicit planning via search tree in VRR agent.

	Average Return			
Game	VRR	DreamerV2	IMPALA	PPO
Original	1.0	0.91	1.0	1.0
Rotated	1.0	0.07	0.43	0.37

Table 3: Zero-shot agent performance on the randomly rotated DoorKey environment.

	Average Return			
Game	VRR	Dream	IMPALA	PPO
Sokoban (7×7)	1.0	0.65	0.76	0.64
Sokoban (13×13)	1.0	0.04	0.11	0.0
DoorKey (6×6)	1.0	1.0	1.0	1.0
DoorKey (32×32)	1.0	0.0	0.0	0.0

Table 2: Zero-shot performance with varying grid size. Note: Sokoban is the 1-box map. Dream: DreamerV2.

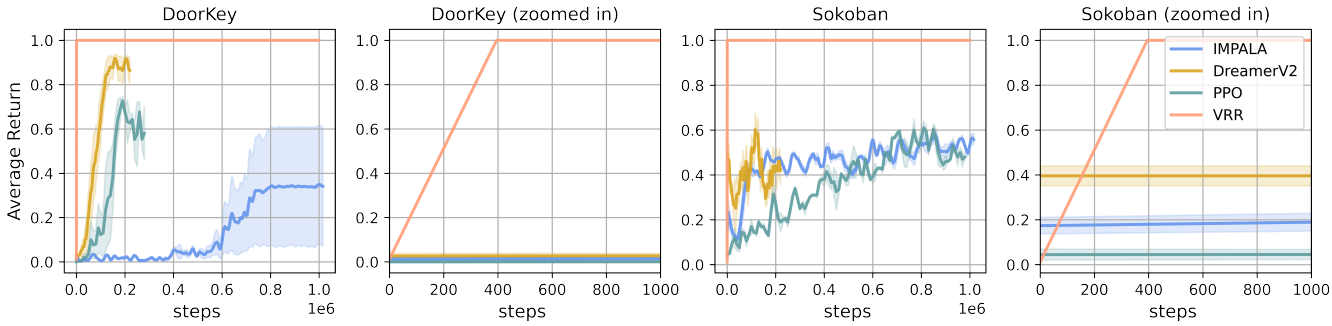

Figure 6: Average return during training. From left to right: DoorKey (6×6), DoorKey (6×6) zoomed, Sokoban (7×7), Sokoban (7×7) zoomed. The VRR agent achieves the same reward while using 3 orders of magnitude fewer training steps.

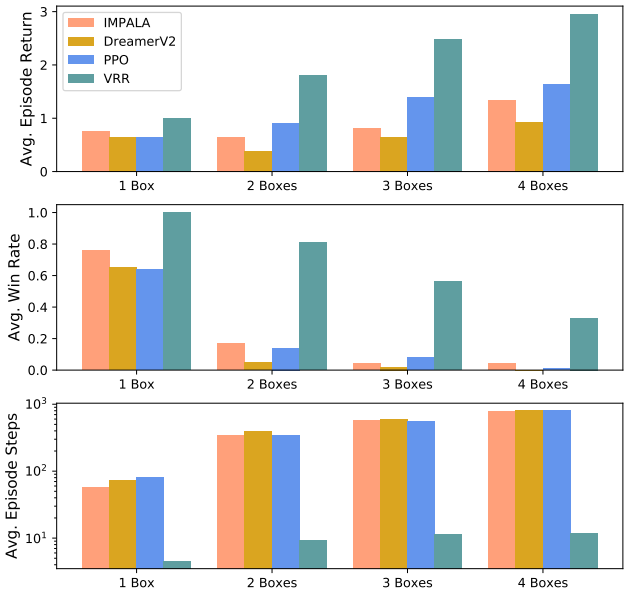

Figure 8: VRR and baseline agents on 7x7 Sokoban with varying number of boxes.

DoorKey First, the VRR and baseline agents are tested in a larger 32×32 DoorKey environment (Fig. 5, pic.(6)), where the layout is exactly identical: the agent starting position and the key are located to the left of the vertical wall, and the goal is located at the bottom right. Similar to Sokoban, Table 2 shows that VRR agent performance is invariant to board size, while baselines completely fail.

Additionally, we test VRR and baseline agents in 6×6 DoorKey environment with randomly rotated initial states. For example, the goal state and agent initial positions may be swapped, or the agent may need to move upwards to open the door, instead of to the right. Table 3 shows that VRR agent generalizes to rotated game board layouts, while baselines overfit to a particular orientation of the game board. The generalization ability of VRR agent is due to the VRR world model is invariant to rotations by design.

Conclusion and Future Work

In this paper, we attacked the generalization problem in RL using the concept of visual rewrite rules (VRR). VRR agents maintain a set of action-dependent graphical rules that describe action effects as local visual changes around the agent. Inspired by human game-playing priors, VRR models environment dynamics as minimal factored local changes. Given its simplicity and locality, a VRR agent can be trained with orders of magnitudes less data but still generalize better than the deep RL agents. Though we demonstrate the effectiveness of our method in grid-based environments, many open questions emerge from the assumptions of VRR. For example, can visual rewrite rules be easily generalized to continuous space environments? We suppose one could easily plug in an object detector to get the factored state observation and follow the same VRR algorithm, but we do not know if the training cost of the object detector will cancel out the efficiency and generalization ability of VRR. We also do not deny that deep representation learning is essential to high-dimensional problems. So it is natural to ask if the idea of visual rewrite is compatible with deep representation learning. Will this approach impose new inductive biases in deep agent design? Relaxations of any of the assumptions identified in the paper would be interesting future topics.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. L. 2018. State Abstractions for Lifelong Reinforcement Learning. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 10–19. PMLR.

Ackley, D. H. 2018. Digital protocells with dynamic size, position, and topology. In *ALIFE 2018: The 2018 Conference on Artificial Life*, 83–90.

Asadi, K.; Misra, D.; Kim, S.; and Littman, M. L. 2019. Combating the Compounding-Error Problem with a Multi-step Model. ArXiv preprint arXiv:1905.13320.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z. D.; and Blundell, C. 2020. Agent57:

Outperforming the Atari Human Benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 507–517. PMLR.

Bellman, R. 1966. Dynamic programming. *Science*, 153(3731): 34–37.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 2048–2056. PMLR.

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2019. Quantifying Generalization in Reinforcement Learning. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, 1282–1289. PMLR.

Coulom, R. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In van den Herik, H. J.; Ciancarini, P.; and Donkers, H. H. L. M., eds., *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, 72–83. Springer.

Dean, T. L.; and Givan, R. 1997. Model Minimization in Markov Decision Processes. In Kuipers, B.; and Webber, B. L., eds., *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, 106–111. AAAI Press / The MIT Press.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In Cohen, W. W.; McCallum, A.; and Roweis, S. T., eds., *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, 240–247. ACM.

Dubey, R.; Agrawal, P.; Pathak, D.; Griffiths, T.; and Efros, A. A. 2018. Investigating Human Priors for Playing Video Games. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 1348–1356. PMLR.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 1406–1415. PMLR.

Furnas, G. W. 1991. New graphical reasoning models for understanding graphical interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 71–78.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W.; and Titterington, D. M., eds., *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, 249–256. JMLR.org.

Goldwaser, A.; and Thielscher, M. 2020. Deep Reinforcement Learning for General Game Playing. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 1701–1708. AAAI Press.

Ha, D.; and Schmidhuber, J. 2018. Recurrent World Models Facilitate Policy Evolution. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2455–2467.

Hafner, D.; Lillicrap, T. P.; Norouzi, M.; and Ba, J. 2021. Mastering Atari with Discrete World Models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Hamrick, J. B.; Ballard, A. J.; Pascanu, R.; Vinyals, O.; Heess, N.; and Battaglia, P. W. 2017. Metacontrol for Adaptive Imagination-Based Optimization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Hamrick, J. B.; Friesen, A. L.; Behbahani, F.; Guez, A.; Viola, F.; Witherspoon, S.; Anthony, T.; Buesing, L. H.; Velickovic, P.; and Weber, T. 2021. On the role of planning in model-based deep reinforcement learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 12498–12509.

Jiang, N.; Kulesza, A.; and Singh, S. P. 2015. Abstraction Selection in Model-based Reinforcement Learning. In Bach, F. R.; and Blei, D. M., eds., *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille,*

France, 6-11 July 2015, volume 37 of *JMLR Workshop and Conference Proceedings*, 179–188. JMLR.org.

Jong, N. K.; and Stone, P. 2005. State Abstraction Discovery from Irrelevant State Variables. In Kaelbling, L. P.; and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 752–757. Professional Book Center.

Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, 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 *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Konidaris, G. 2019. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29: 1–7. Artificial Intelligence.

Konidaris, G. D.; and Barto, A. G. 2009. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In Bengio, Y.; Schuurmans, D.; Lafferty, J. D.; Williams, C. K. I.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009.*, 1015–1023. Curran Associates, Inc.

Laskin, M.; Lee, K.; Stooke, A.; Pinto, L.; Abbeel, P.; and Srinivas, A. 2020. Reinforcement Learning with Augmented Data. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lee, K.; Lee, K.; Shin, J.; and Lee, H. 2020. Network Randomization: A Simple Technique for Generalization in Deep Reinforcement Learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. In *Ninth International Symposium on Artificial Intelligence and Mathematics*.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J.; Jordan, M. I.; and Stoica, I. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 3059–3068. PMLR.

Racanière, S.; Weber, T.; Reichert, D. P.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; Pascanu, R.; Battaglia, P. W.; Hassabis, D.; Silver, D.; and Wierstra, D. 2017. Imagination-Augmented Agents for Deep Reinforcement Learning. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 5690–5701.

Schrader, M.-P. B. 2018. gym-sokoban. https://github.com/mpSchrader/gym-sokoban.

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.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Sharma, A.; Gu, S.; Levine, S.; Kumar, V.; and Hausman, K. 2020. Dynamics-Aware Unsupervised Discovery of Skills. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Sutton, R. S. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.*, 2(4): 160–163.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2): 181–211.

Tsividis, P. A.; Pouncy, T.; Xu, J. L.; Tenenbaum, J. B.; and Gershman, S. J. 2017. Human learning in Atari. In *2017 AAAI Spring Symposium Series*.

Witty, S.; Lee, J. K.; Tosch, E.; Atrey, A.; Littman, M. L.; and Jensen, D. D. 2018. Measuring and Characterizing Generalization in Deep Reinforcement Learning. *CoRR*, abs/1812.02868.