Journal of Artificial Intelligence Research 75 (2022) 709-745 Submitted 03/2022; published 11/2022

# DeepSym: Deep Symbol Generation and Rule Learning for Planning from Unsupervised Robot Interaction

**Alper Ahmetoglu** ALPER.AHMETOGLU@BOUN.EDU.TR
**M. Yunus Seker** YUNUS.SEKER1@BOUN.EDU.TR
*Department of Computer Engineering*
*Bogazici University, Istanbul, Turkey*

**Justus Piater** JUSTUS.PIATER@UIBK.AC.AT
*Department of Computer Science*
*Universität Innsbruck, Austria*

**Erhan Oztop** ERHAN.OZTOP@OZYEGIN.EDU.TR
*Osaka University, Osaka, Japan*
*Ozyegin University, Istanbul, Turkey*

**Emre Ugur** EMRE.UGUR@BOUN.EDU.TR
*Department of Computer Engineering*
*Bogazici University, Istanbul, Turkey*

## Abstract

Symbolic planning and reasoning are powerful tools for robots tackling complex tasks. However, the need to manually design the symbols restrict their applicability, especially for robots that are expected to act in open-ended environments. Therefore symbol formation and rule extraction should be considered part of robot learning, which, when done properly, will offer scalability, flexibility, and robustness. Towards this goal, we propose a novel general method that finds action-grounded, discrete object and effect categories and builds probabilistic rules over them for non-trivial action planning. Our robot interacts with objects using an initial action repertoire that is assumed to be acquired earlier and observes the effects it can create in the environment. To form action-grounded object, effect, and relational categories, we employ a binary bottleneck layer in a predictive, deep encoder-decoder network that takes the image of the scene and the action applied as input, and generates the resulting effects in the scene in pixel coordinates. After learning, the binary latent vector represents action-driven object categories based on the interaction experience of the robot. To distill the knowledge represented by the neural network into rules useful for symbolic reasoning, a decision tree is trained to reproduce its decoder function. Probabilistic rules are extracted from the decision paths of the tree and are represented in the Probabilistic Planning Domain Definition Language (PPDDL), allowing off-the-shelf planners to operate on the knowledge extracted from the sensorimotor experience of the robot. The deployment of the proposed approach for a simulated robotic manipulator enabled the discovery of discrete representations of object properties such as 'rollable' and 'insertable'. In turn, the use of these representations as symbols allowed the generation of effective plans for achieving goals, such as building towers of the desired height, demonstrating the effectiveness of the approach for multi-step object manipulation. Finally, we demonstrate that the system is not only restricted to the robotics domain by assessing its applicability to the MNIST 8-puzzle domain in which learned symbols allow for the generation of plans that move the empty tile into any given position.

## 1. Introduction

Intelligent robotic systems exploit a diverse range of data representations for control, learning, and reasoning. Interaction with the world requires processing low-level continuous sensorimotor representations whereas abstract reasoning requires the use of high-level symbolic representations. The representational gap between the low-level sensorimotor and high-level symbolic representations has been addressed in AI and robotics, often by using manually designed symbols that are grounded in the low-level sensorimotor experience of the robots interacting with their environment (Harnad, 1990; Taniguchi et al., 2018). However, this approach only works in either controlled environments or limited tasks. Yet, truly intelligent robots are expected to form abstractions (Konidaris, 2019) continually from their interaction with the world and use them on the fly for complex planning and reasoning in novel environments (Werner & Kaplan, 1963; Callaghan & Corbit, 2015).

In this paper, we address the challenging problem of discovering discrete symbols and unsupervised learning of rules from the low-level interaction experience of a self-exploring robot. For this purpose, we propose a novel deep neural architecture for symbol formation and rule extraction. At the core of our method, the symbols are discovered in the discrete latent space formed by the bottleneck layer of a predictive, deep encoder-decoder network that takes the image of an object and the action applied as the input, and produces the effect generated by the action as the output. Symbols, which are the output of the encoder network, hold information for the effect prediction for a given action. Furthermore, our architecture allows transforming the complete low-level sensorimotor experience into symbolic experience, facilitating direct rule extraction for AI planning. To this end, decision tree models are trained to learn probabilistic rules that are translated to Probabilistic Planning Domain Definition Language (PPDDL; Younes & Littman, 2004) operators that are standard in probabilistic planning. Note that the predicates that appear in the PPDDL operators correspond to the discovered symbols.

In order to realize this framework, we created a setup where a simulated robot manipulator interacts with objects, poking them in different directions and stacking them on top of each other to collect interaction experience for object categorization and rule learning. Our system successfully constructs a latent representation through which object and relational symbols are discovered, which can be interpreted by humans as ‘rollable’, ‘insertable’, ‘larger-than’. Contrary to symbols generated by systems that disregard actions and effects, our architecture is shown to generate action-effect-regulated symbols that are more effective in abstract reasoning over the actions of the robot and the consequences in the environment. Furthermore, the number of symbols is determined automatically by optimizing the trade-off between prediction capability and bottleneck size. Finally, the system acquired the capability to generate effective plans to achieve goals such as building towers of desired heights from given cubes, balls, and cups using off-the-shelf probabilistic planners. To show the generality of the proposed approach, we also conduct a second set of experiments in a non-robotic domain. To be concrete, we test our approach in the MNIST 8-tile puzzle domain adapted from Asai and Fukunaga (2018). Our experiments show that the system learns symbols that allow for creating plans to move the empty tile into arbitrary positions. Our implementation is publicly available[1].

1. https://github.com/alper111/DeepSym

Our primary contribution is a generic neural solution for mapping raw sensorimotor experience into the symbolic domain. The same architecture can be used to discover object symbols, effect symbols, and object-object relational symbols. The proposed network further allows progressive learning of increasingly complex abstractions, exploiting previously-learned abstractions as inputs. The learned symbols allow abstraction of the interaction of the robot with its environment as a Markov decision process which allows the use of symbolic planning systems for goal satisfaction. In the current study, to show this, we transformed the learned rules into Probabilistic PDDL operators, which allowed probabilistic plan generation and execution achieving goals beyond what was possible with the direct use of the training data.

## 2. Related Work

Bridging the representational gap between the continuous sensorimotor world of a robotic system with the discrete symbols and rules has been a key research goal from the early days of intelligent robotics (Kuipers et al., 2017; Murphy & Murphy, 2000). While grounding predefined symbols in the sensorimotor experience of the robot has been widely used for intelligent robot control (Klingspor et al., 1996; Petrick et al., 2008; Mourao et al., 2008; Wörgötter et al., 2009; Kulick et al., 2013), some argue that symbols "are not formed in isolation", and "they are formed in relation to the experience of agents" (Sun, 2000). We share this viewpoint that has been investigated in a number of studies. Pisokas and Nehmzow (2005) and Ugur et al. (2011) realized systems that clustered low-level sensory experience into categories and performed subsymbolic planning in the continuous perceptual space. While simple planning capability was achieved, the use of continuous prediction and state transition operators limited the use of powerful off-the-shelf symbolic AI planners. In another line of research, Ozturkcu et al. (2020) asked whether there are any symbols formed in a deep RL agent after training the agent for a given task, without imposing any prior on the architecture or the objective.

Mota and Sridharan (2019), Riley and Sridharan (2019) proposed a hybrid approach to exploit the prior domain knowledge by combining non-monotonic logical reasoning with deep networks. This architecture is a cascade of two models where the first model is the prior domain knowledge encoded as an Answer Set Prolog (ASP) program (Law et al., 2018), and the second model is a convolutional neural network (CNN). If the ASP program fails to classify an example, it redirects the necessary parts of the input to CNN for further processing. This pipeline results in better accuracy with less computation when compared with CNN classification. Furthermore, given labeled examples about the task, the ASP program can be further extended to include new rules about the environment by using the decision paths of a trained decision tree. These works primarily focus on integrating neural models with common-sense knowledge or domain knowledge to increase performance. Our work is similar to these works in the sense that they also learn previously unknown rules with decision trees from subsymbolic data that would help for the planning. On the other hand, we focus on learning symbols that depend on the action set of the agent.

The bottom-up generation of symbolic structures from the continuous interaction experience of a robot has started to draw attention in robotics (Taniguchi et al., 2018; Konidaris, 2019). Konidaris et al. (2014, 2015) studied the construction of symbols that are directly used as preconditions and effects of actions for the generation of deterministic and probabilistic plans in 2D agent settings, and Konidaris et al. (2018) extended the framework into a real-world robot setting. However, these studies use a global state representation, and therefore, symbols learned in an environment cannot be used in a novel environment directly. In follow-up work, James et al. (2020) constructs symbols with egocentric representations to allow the transfer of previously learned symbols. These studies train an SVM classifier for each effect cluster to find groundings of precondition symbols. Ugur and Piater (2015a, 2015b) formed symbols used in plan generation in manipulation using a combination of several ad-hoc machine learning techniques such as clustering with X-means and classification with SVMs. Furthermore, they used hand-crafted features to represent scenes and effects. On the other hand, our proposed architecture simultaneously learns object categories (in the encoder output) and their corresponding effect categories (in the decoder output) without resorting to any clustering techniques on the object or effect space. The object and effect categories automatically emerge as the network with binary bottleneck units minimizes the effect prediction error. Furthermore, deep neural networks allow us to efficiently process high-dimensional image data using convolutional layers. This design approach offers a generic symbol formation engine that runs at the pixel level using deep neural networks. In terms of symbol multiplicity, our approach is more parsimonious, as we do not form symbols for each action as in Ugur and Piater (2015b) and Konidaris et al. (2018); but instead, use a single decoder network that takes the action as part of the input. To be concrete, for $n$ effect categories and $k$ actions, our system generates $nk$ symbols, whereas the aforementioned approaches generate $n^k$ symbols. Learning a single model for all actions possibly allows internal representations learned for one action to be re-used directly for other actions. Another significant advantage of our model is that it is differentiable and thus can be integrated into gradient-based state-of-the-art machine learning architectures for further tackling more complex problems.

Asai and Fukunaga (2018) realized a similar neural framework where they first train a state autoencoder with discrete latent units, then learn the action precondition-effect mappings. In follow-up work, (Asai & Muise, 2021; Asai et al., 2022) combine these two steps and learn the action mapping together with the state auto-encoder. These works are in the visual domain (for example, 2D puzzles) and achieve visualized plan execution while we focus on robot action planning and execution in the 3D world. Moreover, a critical difference of our method from the aforementioned work is that we learn object symbols by taking into account action and the effects in addition to object features, which facilitates the formation of symbols that are likely to capture object affordances (Gibson, 2014; Zech et al., 2017).

Another line of research focuses on bilevel planning, in which a symbolic plan is complemented by a motion and task planner. Silver et al. (2021), Chitnis et al. (2021) learn operators for bilevel planning when given parameterized policies for continuous planning. In a follow-up work (Silver et al., 2022a), these parameterized policies are learned as well, completing the whole neurosymbolic planning pipeline. While these works fix the state abstractions, Silver et al. (2022b) also learns new state abstractions that are optimized for planning. In general, these works focus on learning high-level operators for bilevel planning, while we focus on learning symbols from continuous high-dimensional vectors. Another similar work (Yuan et al., 2022) trains a network that outputs relations between objects from RGB images given objects' canonical images.

## 3. Problem Formulation

In this work, we refer to symbols as discrete low-dimensional vectors extracted from deep neural networks for the current state and used to predict the observed effect of specific actions. More formally, a symbol $\mathbf{z} \in \mathcal{Z}$ is a discrete representation that represents a subset $\mathcal{P}$ of a continuous high-dimensional space $\mathbb{R}^n$ (e.g., the state-space, or the effect-space). The symbol-space $\mathcal{Z}$ can be defined as a set of $k$-dimensional boolean vectors $\mathcal{Z} = \mathbb{B}^m = \{0, 1\}^m$, or as a set of atoms $\mathcal{Z} = \{z_1, z_2, \ldots, z_m\}$. The important condition here is that the symbol-space should be finite, and its cardinality $|\mathcal{Z}|$ should preferably be small. In general, the symbol learning problem refers to finding the mapping $f : \mathbb{R}^n \to \mathcal{Z}$, which would allow us to do logical reasoning in the symbolic domain.

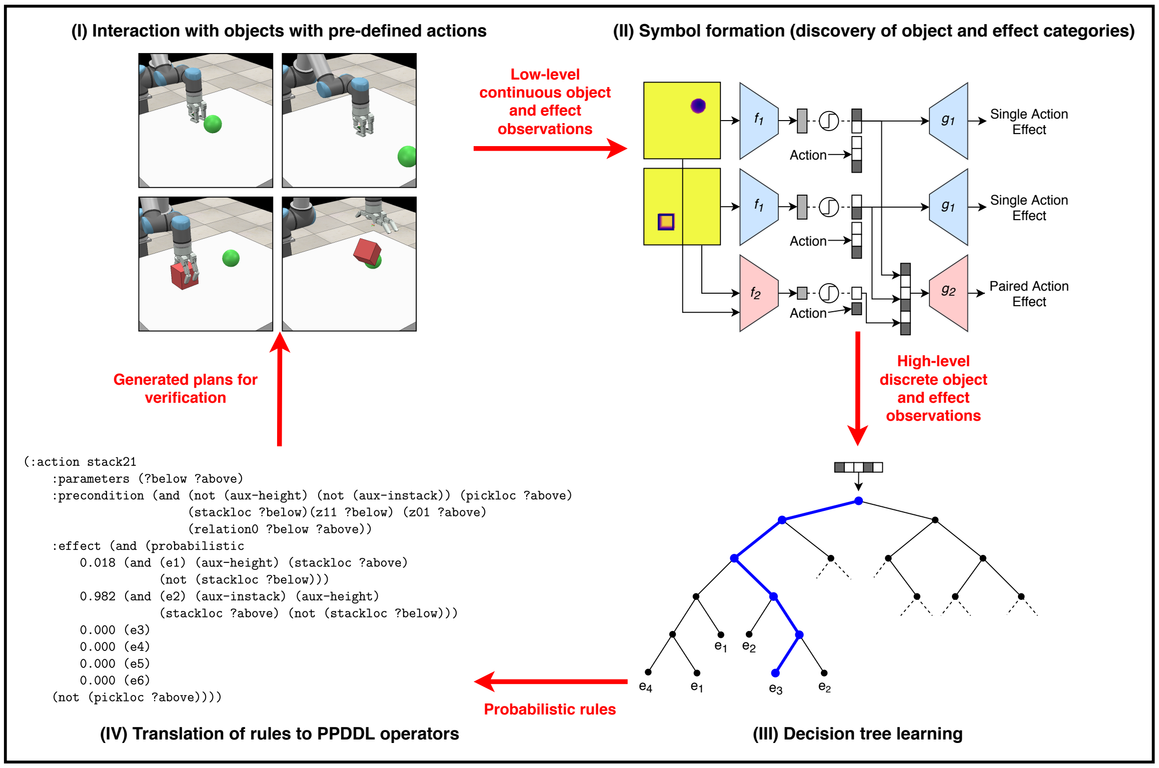
Given a set of discrete actions $\mathcal{A} = \{a_1, a_2, ..., a_k\}$, continuous object (or state) space $\mathbb{R}^n$, and continuous effect space $\mathbb{R}^m$, we are interested in learning an encoder function $f : \mathbb{R}^n \to \mathcal{Z}$ and a decoder function $g : \mathcal{Z} \times \mathcal{A} \to \mathbb{R}^m$ from samples $\{\mathbf{o}^{(i)}, a^{(i)}, \mathbf{e}^{(i)}\}_{i=1}^{N}$ collected by interacting with the environment. Essentially, the encoder outputs symbol $\mathbf{z}$ given the object state $\mathbf{o} \in \mathbb{R}^n$, and the decoder outputs effect $\mathbf{e} \in \mathbb{R}^m$ for symbol $\mathbf{z}$ and action $a$. After learning the encoder and the decoder function by iteratively optimizing an objective (which will be discussed in Section 4), $\mathbf{z}$ corresponds to an object symbol, and $c$ corresponds to an effect symbol that has the grounding $\mathbf{e} = g(\mathbf{z}, a)$ (note that $c$ is an atom while $\mathbf{e}$ is a continuous vector). Once we have such symbols, we can construct a high-level plan in the symbolic space by transforming the environment to a probabilistic PDDL domain defined over the symbols, and then use state-of-the-art off-the-shelf planners to find an action sequence that arrives at the desired goal state.

The experiments reported in this paper involve two environments from different domains, namely, a tabletop robotic manipulation environment and MNIST 8-puzzle environment adapted from (Asai & Fukunaga, 2018). The former is an embodied robotic environment in which symbols that emerge depend on the actions executed by a robotic arm and their corresponding effects. In the MNIST 8-puzzle environment, an agent without an embodiment executes actions and observes the corresponding effects as the visual change in the environment. Symbols are learned with respect to these actions and visual effects.

For simplification, we make the following assumptions in the tabletop manipulation environment:

- The agent is assumed to have a small number of actions, such as poking and stacking an object. Such an action repertoire can be autonomously acquired through a developmental progression as in (Ugur et al., 2012) or obtained through learning from demonstration and reinforcement learning (e.g., Seker et al., 2019; Akbulut et al., 2021).

- The agent is equipped with image processing capability to detect the objects in the camera image and also calculate their pixel coordinates. Furthermore, using the same object tracking method, the agent can take cropped images as input. In the tabletop setup, we realized this with a simple algorithm as the background is uncluttered. In a


(I) Interaction with objects with pre-defined actions
(II) Symbol formation (discovery of object and effect categories)
Low-level continuous object and effect observations
$f_1$
$g_1$
Action
Single Action Effect
$f_1$
$g_1$
Action
Single Action Effect
$f_2$
$g_2$
Action
Paired Action Effect
High-level discrete object and effect observations
Generated plans for verification
$e_1$
$e_2$
$e_4$
$e_1$
$e_3$
$e_2$
Probabilistic rules
(IV) Translation of rules to PPDDL operators
(III) Decision tree learning

```
(:action stack21
    :parameters (?below ?above)
    :precondition (and (not (aux-height) (not (aux-instack)) (pickloc ?above)
                       (stackloc ?below)(z11 ?below) (z01 ?above)
                       (relation0 ?below ?above))
    :effect (and (probabilistic
        0.018 (and (e1) (aux-height) (stackloc ?above)
                   (not (stackloc ?below)))
        0.982 (and (e2) (aux-instack) (aux-height)
                   (stackloc ?above) (not (stackloc ?below)))
        0.000 (e3)
        0.000 (e4)
        0.000 (e5)
        0.000 (e6)
    (not (pickloc ?above))))
```

Figure 1: General system overview of rule generation and refinement.

real-world scenario, state-of-the-art computer vision techniques can be used to detect and track objects in the 3D world.

In the MNIST 8-puzzle environment, the only assumption is that the agent has access to the action repertoire (e.g., ‘slide-left’, ‘slide-down’), which it can execute to see the effects of its actions.

## 4. Methods

Figure 1 provides the overall learning architecture of our proposed system in the robotic manipulation environment; the application of the architecture to the MNIST 8-puzzle domain is given in Section 6. In the environment interaction phase (I), the robot chooses an action from its action repertoire $a \in \mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, observes the object state $\mathbf{o}$, executes the action, and records the resulting effect $\mathbf{e}$.

Using the interaction experience $\{\mathbf{o}^{(i)}, a^{(i)}, \mathbf{e}^{(i)}\}_{i=1}^{N}$, the symbol formation is achieved in (II). To this end, a deep neural network model with two parts is trained to predict $\mathbf{e}$ given $\mathbf{o}$ and $a$. The first part is the encoder network, $f(\mathbf{o})$, which creates a *binary* latent vector $\mathbf{z}$ given the depth image of the object, $\mathbf{o}$. The second part, the decoder network $g(\mathbf{z}, a)$, predicts the effect $\mathbf{e}$ when action $a$ is executed on state $\mathbf{o}$ that has the latent representation $\mathbf{z}$. As the network tries to predict effects, symbolic representations are created by the encoder network that can be treated as object categories regulated by the corresponding action-effect experience.

The continuous interaction experience $\{\mathbf{o}^{(i)}, a^{(i)}, \mathbf{e}^{(i)}\}_{i=1}^{N}$ is transformed into the symbolic experience $\{\mathbf{z}^{(i)}, a^{(i)}, c^{(i)}\}_{i=1}^{N}$ using the discovered categories, and then the symbolic experience is used to distill a decision tree to predict effects given object categories and actions in (III). The reason to use a decision tree is that we can represent any statement in propositional logic with decision trees (Russell & Norvig, 2020, Ch. 19.3) and we can convert rules of the environment into logical statements that encode pre- and post-conditions of actions on the objects.

Finally, these statements are represented in PPDDL, which allows one to make plans in a probabilistic environment in (IV). Lastly, plans are executed to validate the learned symbols and rules. In the following sections, we describe these parts in detail.

## 4.1 Exploration with the Environment

A manipulator robot with a gripper and a depth camera is used to explore the environment and monitor the changes (Figure 2). The robot is initialized with a fixed set of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ through which it interacts with the objects in its workspace. Forward, side, and top poking actions are used to poke objects from different sides (Figure 2b, top). The stacking action is used to release one object on top of another object (Figure 2b, bottom). These actions are encoded with one-hot encoding. On the perception side, each detected object is represented with its top-down depth image. The generated change, on the other hand, is represented by the positional offset of the acted object in pixel coordinates together with the force change sensed at the wrist joint of the robot. In single-object interactions, the robot observes and stores the initial state as the object-centered, top-down depth image of the object, and the effect as the change in object position and force sensor readings:

$$\mathbf{e}_{\text{single}} = (\Delta x, \Delta y, \Delta d, \Delta F) \tag{1}$$

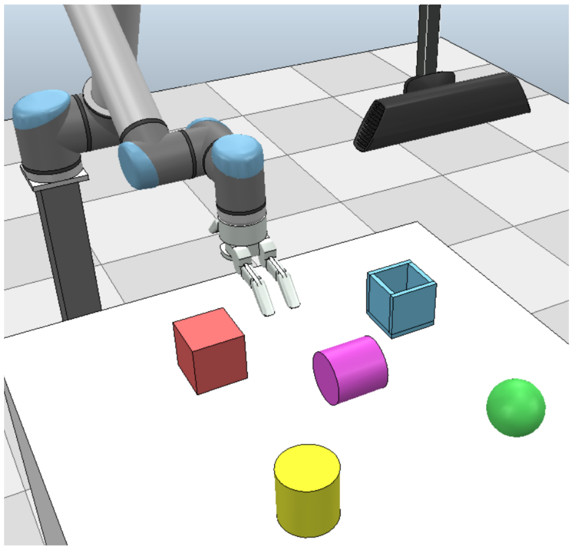
where $\Delta x$ and $\Delta y$ are the changes in $x$-axis and $y$-axis in pixel coordinates, respectively, $\Delta d$ is the change in depth, and $\Delta F$ is the change in force. In paired-object interactions, the robot observes and stores the initial state as the combination of two object-centered depth images $(o_1, o_2)$, and the effect as the change in position of both objects:

$$\mathbf{e}_{\text{paired}} = (\Delta x_1, \Delta y_1, \Delta d_1, \Delta x_2, \Delta y_2, \Delta d_2) \tag{2}$$

where $\Delta x_1$, $\Delta y_1$, $\Delta d_1$ refer to the displacement of the first object, and $\Delta x_2$, $\Delta y_2$, $\Delta d_2$ refer to the displacement of the second object.

## 4.2 Symbol Discovery with Deep Networks

The main objective of the network is to discover symbols, i.e., object and effect categories, that are effective in abstract reasoning about the consequences of robot actions. In other words, the object categories, together with robot actions, should give the ability to predict the effect categories. To achieve this, we propose a special neural network structure which is composed of two parts: an encoder $f(\mathbf{o})$ to predict $\mathbf{z}$ which is the object category, and a decoder $g(\mathbf{z}, a)$ to predict $\mathbf{e}$ (Figure 3, top). This is an encoder-decoder design that has been


(a) Experiment setup.

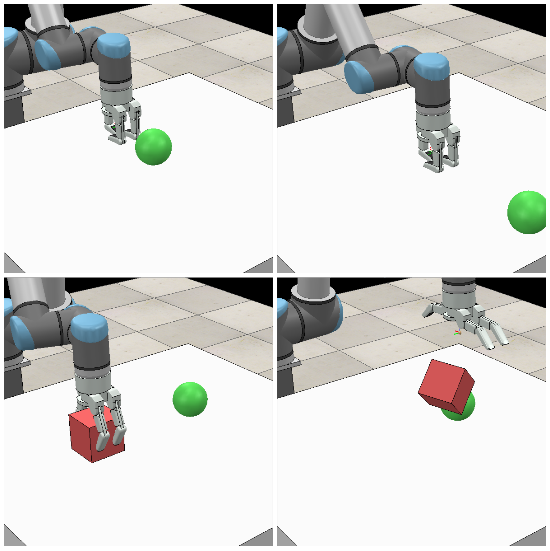

(b) Available actions.

Figure 2: A simulated UR10 robot arm and BarrettHand grasper are used for manipulation; a Kinect sensor is used for perception. Five types of objects are shown on the table.

shown to be quite successful in many different applications (Hinton & Salakhutdinov, 2006; Kingma & Welling, 2013; Sutskever et al., 2014; Devlin et al., 2018). The binary bottleneck layer forces the network to learn low-dimensional symbolic representations that are useful for predicting the generated effect of actions. As the input is a top-down depth image, the encoder is a convolutional neural network with the Gumbel-Sigmoid (GS) function (Maddison et al., 2017; Jang et al., 2017) as the last-layer activation function (where the error back-propagation is handled with the reparameterization trick; Kingma & Welling, 2013). We also experimented with the sign($x$) function using straight-through estimators (STE; Bengio et al., 2013) and found that GS has a lower variance. Results with STE are given in Appendix B. Using GS activation of the bottleneck neurons, the continuous representation is directly transformed into a discrete category. The decoder part is realized as a multi-layer perceptron (MLP). The category $\mathbf{z}$ of the object $\mathbf{o}$ concatenated with the one-hot vector of action $a$ is given to the decoder as input. The decoder predicts the effect $\mathbf{e}$ expected to be observed on object state $\mathbf{o}$ via action $a$. The network minimizes the following objective:

$$\mathcal{L} = \sum_{i=1}^{N} \frac{1}{2} \left( g(f(\mathbf{o}^{(i)}), a^{(i)}) - \mathbf{e}^{(i)} \right)^2 \tag{3}$$

This architecture effectively creates high-level symbolic categories of objects that encapsulate the effects of executed actions. One important advantage is that the model does not need hand-engineered object features and object clusters for finding object symbols, contrary to previous studies, since the system learns *discrete* categories directly to optimize the effect prediction performance. Moreover, as the bottleneck layer is discrete, the possible decoder outputs $\mathbf{e} = g(\mathbf{z}, a)$ form a finite set $\mathcal{E} = \{\mathbf{e}_1, \mathbf{e}_2 \dots\}$ which can be denoted by atoms

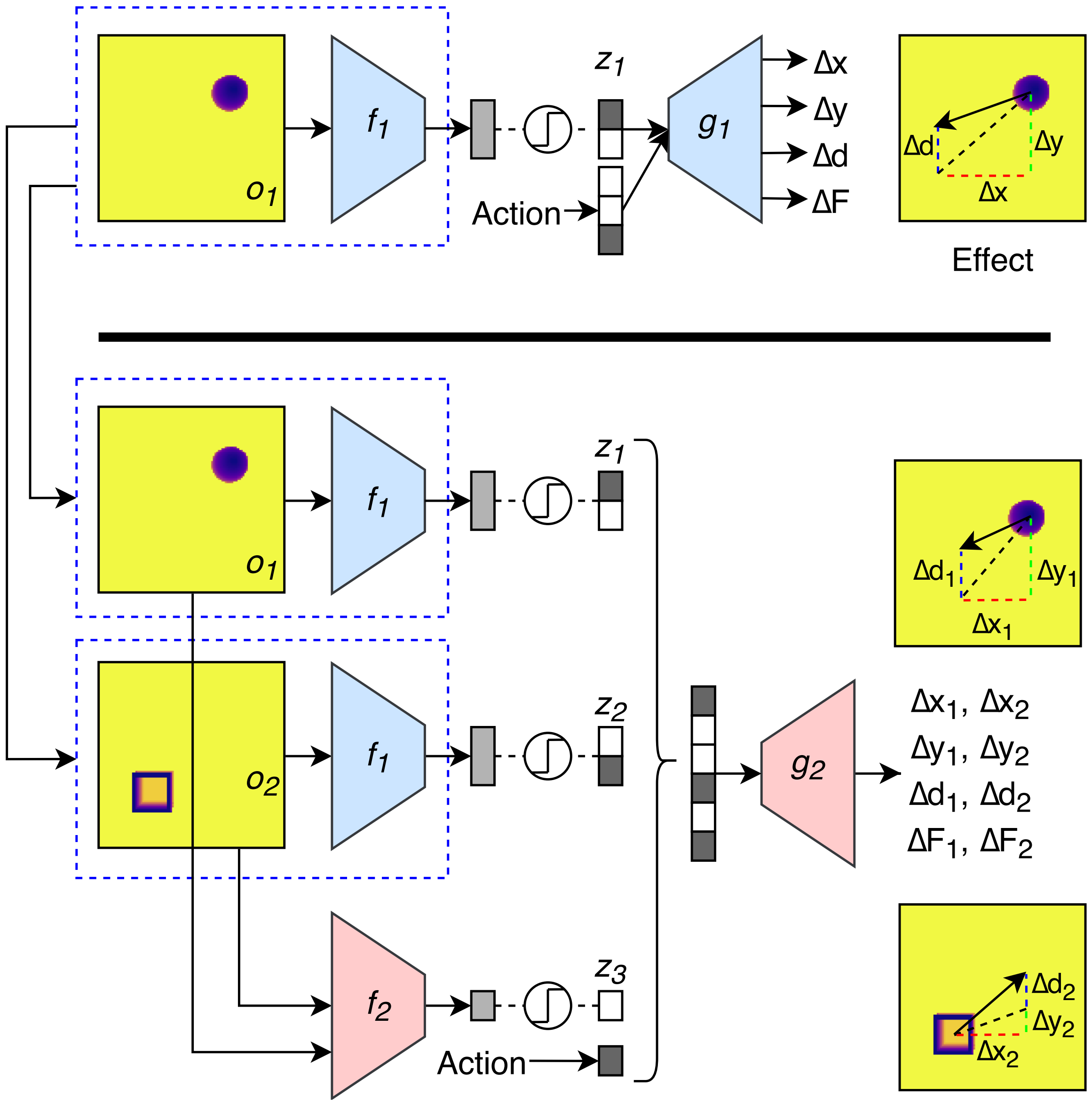


Figure 3: Network architectures for single-object interactions (top) and for paired-object interactions (bottom).

$\mathcal{C} = \{c_1, c_2, \dots\}$. The learned object categories also serve as input for the discovery of new categories with new interaction experience such as actions applied on pairs of objects such as stacking an object $\mathbf{o}_1$ on top of another object $\mathbf{o}_2$ (Figure 2b, bottom). The same deep network structure is used to extract the corresponding symbols with a slight modification to incorporate previously learned knowledge (Figure 3, bottom). Here, an encoder $f_2$ takes the depth images of the objects and produces a binary latent vector $\mathbf{z}_3$. As the important point here, the single object symbols ($\mathbf{z}_1$ and $\mathbf{z}_2$) computed by the $f_1$ encoder are also added to the network as input together with the action information. The idea is that we can use previously-acquired symbols to encode new information more compactly, thus allowing a progressive increment of symbols. Note that the encoder $f_1$ is frozen at this second stage of the training. The encoder $f_1$ provides some interaction related information about objects and let the encoder $f_2$ focus and learn properties and relations *between* the objects.

**Number of symbols** is automatically set by selecting the number of bottleneck neurons using a hyperparameter search procedure. To limit the number of rules and predicates, this procedure aims to find the minimum number of symbols that provide competitive performance in prediction. Starting from one unit, we record the mean and the standard deviation of mean square errors (MSE) of multiple runs. We increase the number of units until there is no significant drop in the prediction error. MSE curves are reported in Appendix A.

### 4.3 Extracting Symbolic Rules

In the third part of the pipeline, a decision tree is trained to predict the effect $c$ of the stack action $a$ given high-level single ($\mathbf{z}_1$ and $\mathbf{z}_2$) and paired ($\mathbf{z}_3$) object categories (i.e., the dataset is $\{[\mathbf{z}_1^{(i)}; \mathbf{z}_2^{(i)}; \mathbf{z}_3^{(i)}; a^{(i)}], c^{(i)}\}_{i=1}^{N}$). Here, the aim is to extract the probabilistic rules of the environment by converting the decision rules on the paths of the tree into logical statements, which ultimately enables probabilistic planning. Each path from the root node to a leaf node in the decision tree stores the required set of predicates $\{p_1 = (z_3 < 0.5), p_2 = (z_2 > 0.5), \dots\}$ represented by discovered single and paired-object categories (in the internal nodes) in order to achieve the effect category $c$ (in the leaves). In other words, each path corresponds to a set of preconditions in order to reach a different effect. As the decision rules at each node in a path $\mathcal{P}$ are in conjunction ($p_1 \wedge p_2 \wedge \cdots \wedge p_k$), and these paths are in disjunction ($\mathcal{P}_1 \vee \mathcal{P}_2 \vee \cdots \vee \mathcal{P}_m$), the tree represents a statement in disjunctive normal form. Thus, any statement in propositional logic can be represented as a decision tree (Russell & Norvig, 2020, Ch. 19.3). The class probabilities at a leaf (the fraction of samples) correspond to probabilities of observing different effects for the same set of preconditions. Therefore, each path is directly converted to a different rule in probabilistic PDDL. While training the decision tree, the minimum number of samples required for a node to be a leaf node is empirically set to 100 samples. The extracted rules are only limited to predicting the effects of an action. In this way, the agent is not expected to learn representations (and consequently rules) unrelated to its embodiment and actions. For example, in our tabletop environment, the robot cannot differentiate cubes from vertical cylinders as different categories since they respond similarly to similar actions even though their visual appearances differ.

Our motivation to construct PPDDL descriptions is to use probabilistic AI planners to efficiently make plans and execute them. PPDDL is composed of a domain description and a problem definition. In the domain description, there are predicates and actions. Predicates represent boolean values that can be activated or deactivated. Each action has a precondition, which is a set of predicates that needs to be satisfied, and an effect, which activates/deactivates other predicates. The domain description is generated from the list of rules. In the problem definition, the initial state of the world is encoded along with the goal to be satisfied. To encode the initial state, the robot perceives the current environment and sets the truth values of the predicates for the existing categories. The planner finds the sequence of actions to satisfy the predicates given in the goal description starting from the initial state using the actions defined in the domain description.

## 5. Robot Experiments

In the following experiments, we aim to answer the following questions to evaluate the proposed method:

1. Do the learned symbols hold any high-level meaning?
2. Are the learned symbols effective for symbolic planning?

We compare our method with two alternative baselines:

1. An autoencoder with discrete activations where symbols are learned directly from passively observed states, independent from actions and effects.
2. An encoder-decoder network with continuous activations, followed by clustering in the latent space.

Regarding the first question, we evaluate the methods based on their performance in differentiating object categories. For the second question, we evaluate the planning performance of different methods.

### 5.1 Experiment Setup

**Interactions:** We adopted the robotic setup, including the action and object sets used, from Ugur and Piater (2015a) who showed effective skill transfer from the simulator to real-world, involving actions with 3-fingered prehension. The experiments are performed in CoppeliaSim VREP simulator (Rohmer et al., 2013) where a six-degrees-of-freedom UR10 (Universal Robots, 2012) robot arm and a Barrett Hand system (Townsend, 2000) interacts with the objects on the table, and a top-down facing Kinect sensor is used for environment perception (Figure 2). The objects used in the experiments include rectangular cups, horizontally and vertically placed cylinders, spheres, and cubes. For each object type, ten different objects with varying diameters/edge lengths in the range of 10 to 20 cm are included in the object dataset for interaction.

**Perception:** Before each action execution a top-down depth image ($128 \times 128$ pixels) of the scene is captured. Objects are placed at different reachable locations on the table during the interactions to ensure the network is invariant with the perspective. Pixels of

| DeepSym | | | | |
|---|---|---|---|---|
| Category | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Sphere | **99.9 ± 0.2** | 0.1 ± 0.2 | 0.0 ± 0.0 | 0.0 ± 0.0 |
| Cube | 0.0 ± 0.0 | **99.9 ± 0.2** | 0.0 ± 0.0 | 0.1 ± 0.2 |
| Vertical Cylinder | 0.0 ± 0.0 | **99.9 ± 0.2** | 0.1 ± 0.2 | 0.0 ± 0.0 |
| Horizontal Cylinder | 0.4 ± 0.9 | 3.1 ± 5.5 | **93.0 ± 4.7** | 3.4 ± 3.8 |
| Cup | 0.0 ± 0.0 | 0.0 ± 0.0 | 0.0 ± 0.0 | **100.0 ± 0.0** |
| Autoencoder (OBO) | | | | |
| Category | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Sphere | **60.6 ± 1.7** | 15.7 ± 5.0 | 15.0 ± 4.5 | 8.8 ± 3.2 |
| Cube | **37.4 ± 2.3** | 22.9 ± 3.1 | 19.3 ± 2.4 | 20.4 ± 4.3 |
| Vertical Cylinder | **44.0 ± 2.9** | 23.2 ± 5.7 | 19.4 ± 7.2 | 13.4 ± 3.1 |
| Horizontal Cylinder | **44.7 ± 2.9** | 20.0 ± 4.9 | 18.6 ± 3.7 | 16.7 ± 4.5 |
| Cup | **86.5 ± 2.0** | 4.3 ± 2.1 | 6.4 ± 2.3 | 2.8 ± 3.7 |
| Continuous bottleneck + clustering (OCEC) | | | | |
| Category | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Sphere | **94.2 ± 9.4** | 4.0 ± 8.4 | 1.8 ± 5.5 | 0.0 ± 0.0 |
| Cube | 0.0 ± 0.0 | **99.0 ± 3.2** | 0.0 ± 0.0 | 1.0 ± 3.2 |
| Vertical Cylinder | 0.0 ± 0.0 | **98.5 ± 4.7** | 0.0 ± 0.0 | 1.5 ± 4.7 |
| Horizontal Cylinder | 28.6 ± 29.1 | 0.0 ± 0.0 | **63.4 ± 26.1** | 8.0 ± 16.9 |
| Cup | 0.0 ± 0.0 | 13.8 ± 32.5 | 0.0 ± 0.0 | **86.2 ± 32.5** |

Table 1: The relative assignment frequencies of objects to different symbolic categories. Here, objects vary in their sizes and initial positions. The mean and the standard deviation of 10 runs are reported. For ease of understanding, we name columns so that the category where spheres are mostly placed is renamed to (0, 0), the category where cubes are mostly placed is renamed to (0, 1), and so on. The naming convention also allows us to take an average across different runs.

the images are normalized globally to increase the convergence speed of stochastic gradient descent (LeCun et al., 2012). Objects in the image are detected with a simple procedure by finding the point with minimum depth and cropping the area of $42 \times 42$ pixels centered around it. This procedure yields object-centered representations for the objects used in the current study but preserves the perspective distortion due to varying locations of the objects and fixed sensor position.

**Encoder-decoder network:** The encoder network (Figure 3) consists of four blocks each containing two convolutional layers that are followed by batch normalization (Ioffe & Szegedy, 2015) and ReLU activation. The number of filters in these blocks are 32, 64, 128, and 256. The last layer consists of two hidden units with a GS activation. The decoder network is a two-layer MLP with 32 hidden units. Further details of these networks can be found in Appendix A.

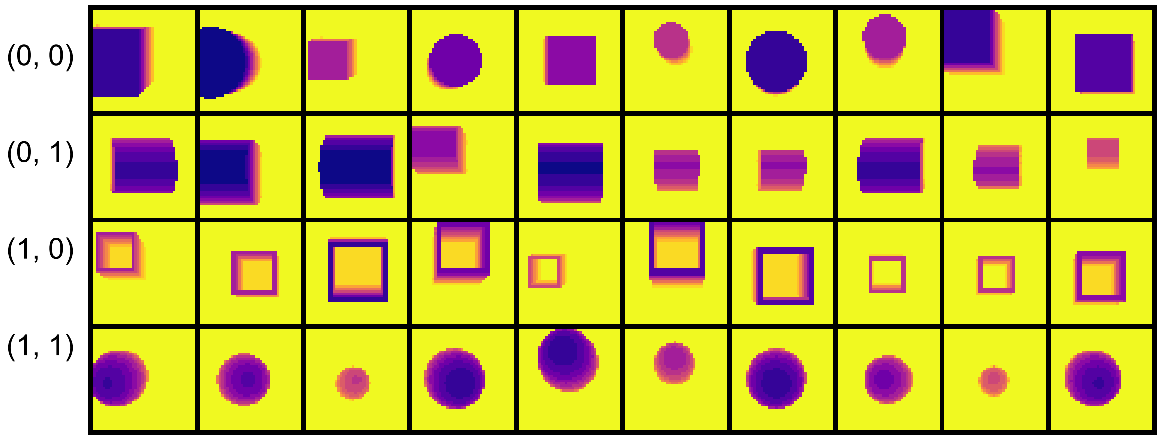


Figure 4: Example depth images as inputs to the encoder network $f_1$.

### 5.2 Discovered Object Categories

Based on the hyperparameter optimization procedure, the number of binary activation neurons in the bottleneck layer is automatically set to 2; therefore, the system found $2^2 = 4$ object categories. How different object types (unknown to the robot) are represented by the discovered object categories is analyzed and provided in Table 1. In general, different types of objects were coded into different categories except that cube and vertical cylinder share the same category even though their depth images differ. This is due to our action and effect regulated categorization: cubes and vertical cylinders behave the same under all available single-object actions of the robot. Although the depth images of the same type of objects with different sizes differ significantly, this information is not reflected in the categories because the size of the objects does not have a significant influence on the consequences of the current actions. The categories can be interpreted as 'pushable'; 'rollable in single direction'; 'pushable and insertable'; and 'rollable in all directions', respectively. Examples from each category are shown in Figure 4.

As a baseline for comparison, we trained

1. An autoencoder with a binary hidden layer using Gumbel-Sigmoid to reconstruct the depth images of objects (inputs to $f_1$) instead of effects. This approach is similar to Asai and Fukunaga (2018). Let us refer to this approach as Object-Binary-Object (OBO).

2. Our proposed encoder-decoder architecture with the binary bottleneck layer replaced with a usual continuous layer that is applied $k$-means clustering ($k = 4$) after learning. Let us call this approach Object-Continuous-Effect followed by Clustering (OCEC).

The results are shown in Table 1. For the autoencoder network (i.e., OBO), we see that objects are collapsed primarily into one category. The robot is expected to predict the consequences of its actions using these categories, and as shown, these categories are not distinctive to help such prediction. With this, we verified the advantage of extracting the symbols from the interaction experience of the robot that includes object-action-effect

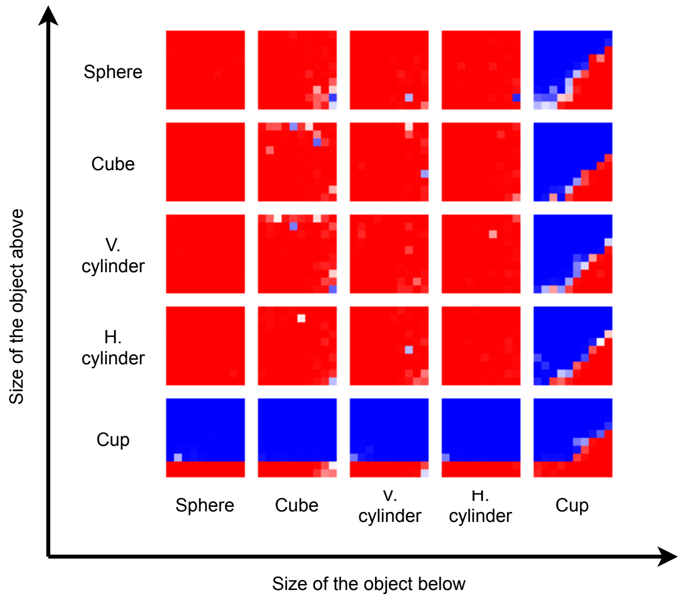


Figure 5: The encoder $f_2$ activations (blue for 0, red for 1) for paired objects. Here, $x$ and $y$ axes of each of the $5 \times 5$ plots represent the sizes of the objects below and above, respectively. Each square represents the relation for a given pair. Note that without any direct supervision, the system discovers approximately linear boundaries (e.g., last column) for some object pairs that would help in effect prediction.

information, i.e., from an object encoder - effect decoder network, rather than searching the symbols in *passively-observed* static features.

OCEC gave better results compared to OBO since the bottleneck layer in OCEC *does* include information from the effect space because of the predictive training similar to our proposed model. However, the latent codes in the bottleneck layer of OCEC might not be distributed locally, making clustering harder. When this is the case, we need more complicated clustering algorithms such as spectral clustering to cluster the latent space accurately. For example, in Table 1, we see that OCEC is more biased toward misclassifying cups as the stable category and the horizontal cylinders as spheres. When we take an average over all objects, our method predicts objects in the correct category with $98.5 \pm 0.94$ % accuracy compared to OCEC with $88.3 \pm 8.62$ % accuracy.

## 5.3 Discovered Relational Categories

The stack interaction experience of the robot is used to train the multi-object encoder-decoder network (Figure 3 bottom), transferring the object categories reported above. The number of binary activated bottleneck neurons is automatically set to 1 using the hyperparameter search described in the Methods section.

The response of the bottleneck neuron, i.e., how this neuron categorizes the input object pairs, is analyzed in Figure 5. Given different pairs of objects with different sizes, each image in this figure corresponds to a specific object pair, and each pixel provides the response of the bottleneck neuron (0 or 1) for specific object sizes. In our experiments, the effect of stack action depends on object categories and their relative size. For example, if an object is released on top of a larger cup, the released object drops into the cup. If the released object is larger than the cup, it is stacked on top of the cup's walls. The approximately linear boundaries for some object pairs in Figure 5 (for example, the last column) show that the bottleneck neuron captured these dynamics and found a symbol that roughly encodes the relative size; the output is 1 when the below cup is larger than the above object. In stacking interactions, the relative size relation only makes sense when the object below is a cup; and our system discovered this relational symbol. Another linear boundary found by the system is in the bottom row. The output of the encoder is 1 when the above object is a cup and below a specific size. We analyze the exploration data to understand why such a boundary emerges. We found out that if the above object is a small cup, the change in the position of the below object is very small.

The learned representations depend on the effect space and the action space of the agent. In our example, after the single-object training stage, the system differentiates different types of objects but does not differentiate different sizes of objects as they are not sufficiently important for the prediction of push actions. Only after it is trained with new data consisting of a new action, namely stacking, does the system start to differentiate between different sizes of cups. The agent only learns richer representations, and therefore better rules, when it has access to a richer action repertoire. This is a desired property of our system as it learns a minimal set of representations needed to predict the outcomes of its actions.

### 5.4 Discovered Effect Categories

After training, we pass the symbol space $\mathcal{Z}$ together with the action space $\mathcal{A}$ to the decoder to get the effect categories. More specifically:

$$\mathcal{C}_{\text{single}} = g_1(\mathcal{Z}_{\text{single}}, \mathcal{A}_{\text{single}}) \tag{4}$$

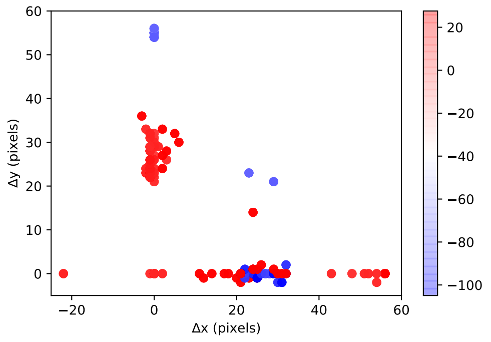
$$\mathcal{C}_{\text{paired}} = g_2(\mathcal{Z}_{\text{paired}}, \mathcal{A}_{\text{paired}}) \tag{5}$$

$$\tag{6}$$

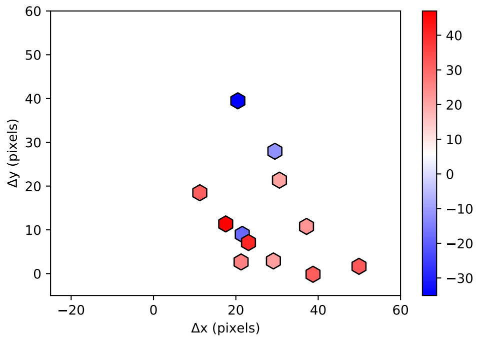
Here, $\mathcal{Z}_{\text{paired}}$ is the Cartesian product of the object category space $\{0,1\}^2$ with the action space $\mathcal{A}_{\text{single}} = \{(0,0,1),(0,1,0),(1,0,0)\}$ resulting in 12 different effect categories for the single object effects. For the paired object effects, the input consists of two single object categories and one relational object category. Therefore, this number is $\{0,1\}^2 \times \{0,1\}^2 \times \{0,1\} \times \mathcal{A}_{\text{paired}} = 32$. Here, $A_{\text{paired}}$ only contains the stack action, therefore $n(\mathcal{A}_{\text{paired}}) = 1$. These effect categories for the single and the paired interactions are shown in Figures 6 and 7, respectively. For visualization purposes, we use colors to represent the third dimension. In Figure 6, the low force values are in blue, and the high force values are in red. Likewise, in Figure 7, the low depth values are in blue, and the high depth values are in red.

### 5.5 Learned Rules and PPDDL Operators

The single- and paired-object categories (acquired from the output of the encoder) together with the action vector are used as inputs to the decision tree in order to predict

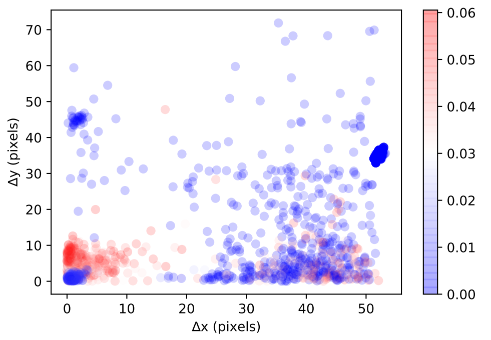


(a) Observed effects, $\mathcal{E}_{\text{single}}$.

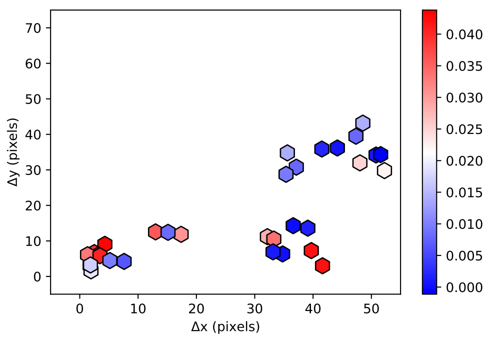


(b) Found effect categories, $\mathcal{C}_{\text{single}}$.

Figure 6: Effect space for the single object interactions. The low force values are in blue, and the high force values are in red. Note that the found effect categories faithfully represent the effect space without any clustering.



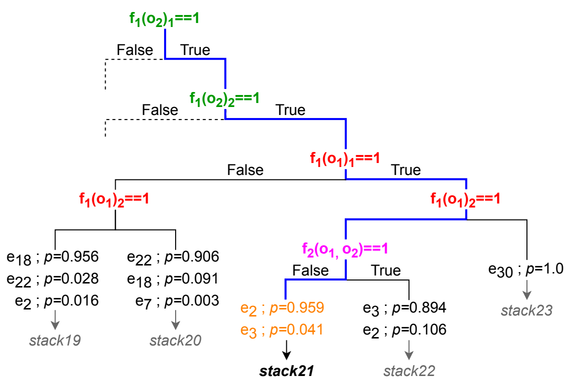
(a) Observed effects, $\mathcal{E}_{\text{paired}}$.



(b) Found effect categories, $\mathcal{C}_{\text{paired}}$.

Figure 7: Effect space for the paired object interactions. Effects of the below object $(\Delta x_2, \Delta y_2, \Delta d_2)$ are omitted as they are almost zero. The low depth values are in blue, and the high depth values are in red.

the effect categories (extracted from the output of the decoder). The learned tree is of depth 5, has 24 leaves, and its classification accuracy is 94.8%. The result of decision tree learning is shown in Figure 8a, where only a small number of decision paths out of 24 is explicitly shown because of the space constraints. Decision rules for the highlighted path is $(f_1(o_1)_1, f_1(o_1)_2, f_1(o_2)_1, f_1(o_2)_2, f_2(o_1, o_2)) = (1, 0, 1, 1, 0)$. Here, $(f_1(o_1)_1, f_1(o_1)_2)$ represents the category of the object above, $(f_1(o_2)_1, f_1(o_2)_2)$ represents the category of the object below, and $f_2(o_1, o_2)$ is the symbol for the paired-object relation. A natural-language translation of this path is as follows: 'If the above object is rollable in all directions (1, 1), and the below object is pushable and insertable (1, 1), and the below object is not larger than the above object, $e_2$ is observed (which is a stacking effect) with 0.959 probability'. PPDDL description corresponding to this decision path of the tree is shown in Figure 8b.



(a) One path of the decision tree is highlighted.

```
(:action stack21
  :parameters (?below ?above)
  :precondition (and (not (aux-height) (not (aux-instack))
                  (pickloc ?above) (stackloc ?below)
                  (f10 ?below) (f11 ?above)
                  (relation0 ?below ?above))
  :effect (and (probabilistic
                0.959 (and (e2) (aux-height) (stackloc ?above)
                        (not (stackloc ?below)))
                0.041 (and (e3) (aux-instack) (aux-height)
                        (stackloc ?above) (not (stackloc ?below)))
                0.000 (e1)
                  ..
                0.000 (e30)
               (not (pickloc ?above))))
(:action increaseheight1
  :precondition (and (aux-height) (H0))
  :effect (and (not (H0)) (H1) (not (aux-height))))

(:action increasestack1
  :precondition (and (aux-instack) (S0))
  :effect (and (not (S0)) (S1) (not (aux-instack))))

(:action makebase
  :parameters (?obj)
  :precondition (not (base))
  :effect (and (base) (aux-height) (aux-instack) (not (pickloc ?obj))
            (stackloc ?obj)))
```

(b) Highlighted path is converted into PPDDL.

Figure 8: An example expansion of the decision tree. $(f_1(o_1)_1, f_1(o_1)_2)$ represents the category of the object above where $(f_2(o_2)_1, f_2(o_2)_2 f)$ represents the category of the object below. The relational variable is denoted as $f_2(o_1, o_2)$. On the leaves, each effect $e_i$ is observed with the corresponding probability.

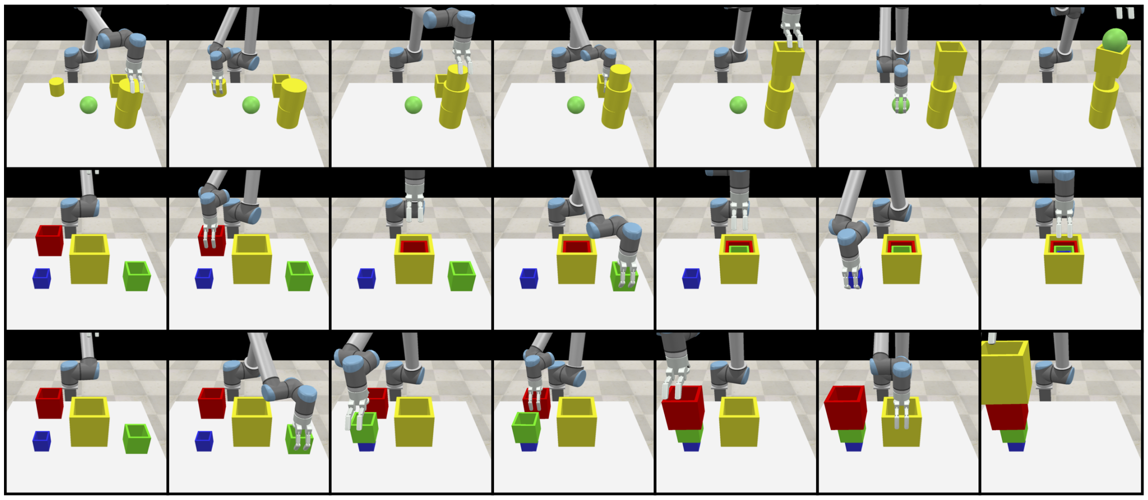


Figure 9: Top row: The objective is to construct a tower of height five using five objects, H5S5. The system assesses the success probability to be 0.07. Middle row: The objective is H1S4, and the system assesses the success probability to be p=0.76. Bottom row: If we change the objective to H4S4, the success probability increases to 0.88.

For our experiments in the tower building task, we manually introduced some auxiliary predicates, as well as special actions for the domain, to be able to chain multiple actions and count the number of objects in the tower. These are needed to set a goal of constructing a tower with multiple objects which are outside the experience of the robot.

| | Estimated prob. | Planning success | Execution success |
|---|---|---|---|
| DeepSym | 80.4 | 95.0 | 70.0 |
| OCEC | 12.1 | 25.0 | 15.0 |

Table 2: Planning results from random 20 configurations.

For effects with small $\Delta x_1$ and $\Delta y_2$ the `aux-instack` predicate is set to true if they satisfy $\Delta d_1 > \epsilon$ for some threshold $\epsilon$, and otherwise the `aux-height` predicate is set to true. For this specific application, these predicates allow us to differentiate stacking and inserting, our effects of interest, from other effects. Actions `increaseheight1` and `increasestack1` are treated as addition operators that increase the height of the tower (H) and the number of objects in the stack (S), respectively. There are multiple H and S predicates ranging from H1-H7 and S1-S7, and likewise multiple `increaseheight` actions. When the `aux-height` effect is observed, the planner must select the `increaseheight1` action to continue with the plan. Therefore, when a stack effect is observed, the height of the tower (which is represented by H) increases automatically.

These would not be needed if we were to use the numeric values associated with effect clusters (i.e., functions in PDDL) as then we would be able to set arbitrary goals such as $\Delta d_1 > 0.3$ ('make the height of the tower taller than 30cm'). We went on with the atomic effect representation (i.e., with no associated parameters) as they worked better with the mGPT implementation[2] that we used. In future work, we plan to extend the planner so that we can also utilize numeric values associated with effect clusters.

Lastly, the predicate `pickloc` is true for objects that are on the table and available for use for the tower construction; `stackloc` is true for the object that is at the top of the tower. These are shown in Figure 8b.

## 5.6 Performance of Planning

The PPDDL descriptions that are automatically constructed by the discovered symbols and rules were verified by generating plans given a set of goals, executing these plans in the simulator, and assessing the success of the executed action sequences in achieving these goals. To be concrete, we asked the system to generate plans to create towers of desired heights with a given fixed set of objects. The challenge of the task is to place objects on top of each other in the correct order. With five objects, there are 5! plans. For plan generation, the mGPT off-the-shelf probabilistic planner (Bonet & Geffner, 2005) with FF heuristic (Hoffmann & Nebel, 2001) was used.

Since in our experiments, the system is asked to generate plans given a number of objects on the table, we encode the task of, say, "construct a tower with a height of three (H3) using four objects (S4)" as H3S4 (Figure 9).

### 5.6.1 DeepSym vs. OCEC

We first compare our system with the alternative OCEC system. We train both systems ten times and select the best-performing models (based on the decision tree accuracy). We

2. `https://github.com/bonetblai/mini-gpt`

| Task | H4S4 | H3S4 | H2S4 | H1S4 |
|---|---|---|---|---|
| Estimated execution probability | 0.91 | 0.93 | 0.86 | 0.68 |
| Success | 0.88 | 0.56 | 0.68 | 0.32 |
| Planning fail | 0.12 | 0.20 | 0.20 | 0.32 |
| Execution fail | 0.00 | 0.24 | 0.12 | 0.36 |

Table 3: Planning results from 25 executions for each task. To satisfy the H1S4 objective, the robot needs to insert objects inside each other, which is more challenging compared to the other tower tasks. Thus, the success probability is lower.

initialized 20 random problems and asked the planner to solve the task using two different domain descriptions generated from different methods. We run the probabilistic planner 100 times for each problem and record the number of successes to estimate the success probability of the plans. The results are reported in Table 2. We report two different metrics: (1) planning success shows whether the system generated a feasible plan or not, and (2) execution success shows whether the execution is successful or not. The latter is concerned with the stochasticity of the environment, not with the feasibility of the plan. We see that the OCEC model performs considerably worse with 25% planning accuracy than our approach with 95% planning accuracy. This is mainly caused by the wrong classification of the cup object (see Table 1 in Section 5.2), which is an essential piece of information in this problem. When single- and paired-object categories are incorrectly classified, the system generates an invalid problem description, which results in infeasible plan outputs. For the same number of symbols, the learned symbols in the OCEC pipeline do not directly depend on the action and the generated effects, while symbols learned with our architecture directly depend on the action and its corresponding effect as they are directly used for effect prediction. This leads to the creation of symbols that are more appropriate for planning.

#### 5.6.2 DeepSym Performance

Now that we have shown the performance gap between the two methods, we want to analyze when our method fails and succeeds. We considered four different goals (towers of heights from 1 to 4) and performed 25 different runs with random initial object configurations for each objective. We configured object types and sizes to have at least one feasible solution. For example, for the H1 objective, we make sure that there are at least three cups that can be physically stacked into each other. The plan execution performance is reported in Table 3. There are three different outcomes: (1) the plan is executed successfully, (2) the planner outputs an erroneous plan due to a recognition error in the encoder, (3) the generated plan is correct, but the plan fails at execution time due to the stochasticity of the environment.

We see that the robot constructs towers with a height of four successfully. As the height of the tower decreases, the robot needs to insert some of the objects inside other cups. The insertion task is harder than the stacking task due to the stochasticity of the environment, which is also reflected in the estimated probabilities in Figure 8b; even if the below cup is larger than the above sphere, the insertion probability is 0.894. For example, for the challenging objective of creating an H1 tower including all objects, the system estimates the success probability to be 0.68, and therefore the failure probability to be 0.32. Accordingly, 36% of plans fail at execution time. This shows that the system can partially model the probabilistic nature of the environment. The planning errors are mostly due to the incorrect recognition of the paired-object categories. Example executions are shown in Figure 9.

#### 5.6.3 Deterministic vs. Probabilistic Planning

We also experimented with deterministic planning instead of a probabilistic one. To do so, while converting rules to PDDL, we take the maximum likely effect as the generated effect. For example, if a specific action schema produces effect $e_2$ with a probability of 0.91 and $e_{17}$ with a probability of 0.09, we take the effect with the maximum probability as the generated effect for the action schema. Thus, deterministic planning eliminates the possibility of other effects and, therefore, effectively eliminates possible solutions. When the learned rules faithfully represent the action-effect relations in the environment, we observe no significant difference between probabilistic and deterministic planning in terms of the success of plans. However, when there is significant inaccuracy in the learned representation (e.g., an incorrect comparison between a pair of objects), the probabilistic PDDL description can account for this inaccuracy in the probabilities of effects; the inaccuracies are reflected as the uncertainty of the environment.

## 6. Experiments on 8-puzzle

In this section, we evaluate DeepSym on the MNIST 8-puzzle adapted from Asai and Fukunaga (2018). In the original 8-puzzle, the aim is to have the tiles in a specific arrangement (considered the goal configuration) by moving tiles into the empty square. In the adapted MNIST 8-puzzle version, tiles do not have symbolic values such as digits but instead contain images of digits, and the 0-tile is treated as the empty tile. Given the domain definition, i.e., the knowledge of how the configuration changes in response to slide actions, the 8-puzzle game can be solved with AI planners. However, the problem becomes non-trivial when the states are represented with raw images of the board, and the state transitions are not known. In the adapted MNIST 8-puzzle, our system is given the raw image of the board with $(3 \times 28) \times (3 \times 28) = 7056$ pixels. Therefore, the state vector is 7056-dimensional. An instance of the MNIST 8-puzzle is shown in Figure 10. A system that can solve the puzzle should recognize the following:

1. Actions only modify some part of the image (i.e., there are tiles),
2. There are specific symbolic representations in these tiles (i.e., recognize the image content of the tiles),
3. The goal is only valid when these tiles are arranged in a specific order (sorted from left to right and top to bottom).

As in our robot experiments, the general pipeline (Figure 1) consists of four stages: (1) exploration, (2) symbol learning, (3) rule learning, (4) and the translation of rules to PDDL.

(1) In the exploration stage, the system initializes a random environment configuration, executes a random action (which is provided to the system), and records a 3-tuple $(\mathbf{x}_t, a_t, \mathbf{e}_t)$

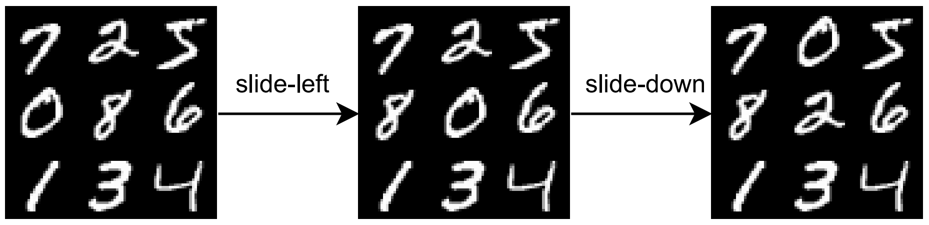


Figure 10: Two steps of the MNIST 8-puzzle. The 0-tile is treated as the empty tile. Each tile consists of a $28 \times 28$-pixel MNIST digit.

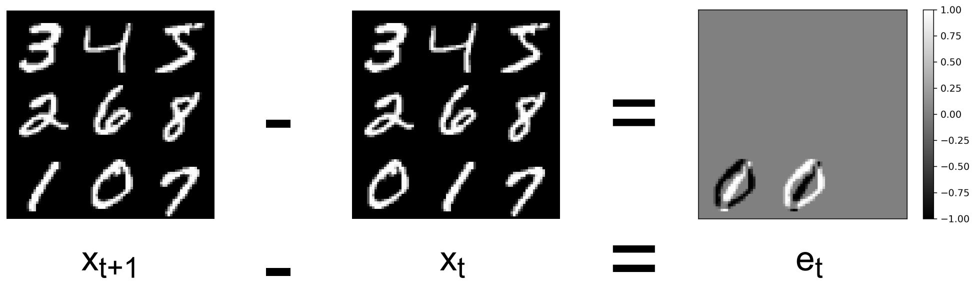


Figure 11: The effect is represented as the difference between two timesteps.

where $\mathbf{x}_t$ is the current state, $a_t$ is the executed action represented as a one-hot vector, and $\mathbf{e}_t$ is the generated effect represented as the pixel difference between the new state $\mathbf{x}_{t+1}$ and the current state $\mathbf{x}_t$ (Figure 11). We collect 100,000 such interactions from the environment. (2) In the symbol learning stage, we train an encoder-decoder network as in Section 4.2, where the encoder $f(\mathbf{x})$ takes the state vector $\mathbf{x}$ as an image of $84 \times 84$ pixels and outputs a binary vector $\mathbf{z}$, and the decoder $g(\mathbf{z}, a)$ takes the concatenation of $\mathbf{z}$ and the action vector $a$ to produce the effect $\mathbf{e}$ which is also an image of $84 \times 84$ pixels. Both the encoder and the decoder are convolutional networks. We did not employ any hyperparameter search on the architectures but followed the building principles in DCGAN (Radford et al., 2016). The details of the networks can be found in Appendix A. We train the model for 100 epochs with MSE loss in Equation 3. (3) After training, we distill the information in the decoder network into rules by training a decision tree using the predictions of the decoder. (4) Lastly, we translate the rules represented by the decision tree into PDDL rules as in Section 4.3.

### 6.1 Learned Symbols

In the MNIST 8-puzzle environment, there is a finite set of possible effects that can be generated in a single action from any environment configuration. If we use the same image for a digit as in Asai and Fukunaga (2018), then the encoder should represent 3248 different states (digits that are nearby the empty tile) in order for the decoder to produce the correct effect. Since we are using binary activations, $\log_2 3248 \approx 11.67$ units are necessary to avoid losing any information regarding effects. Therefore, we set the number of units to 13 (giving one more as a slack) in this experiment.

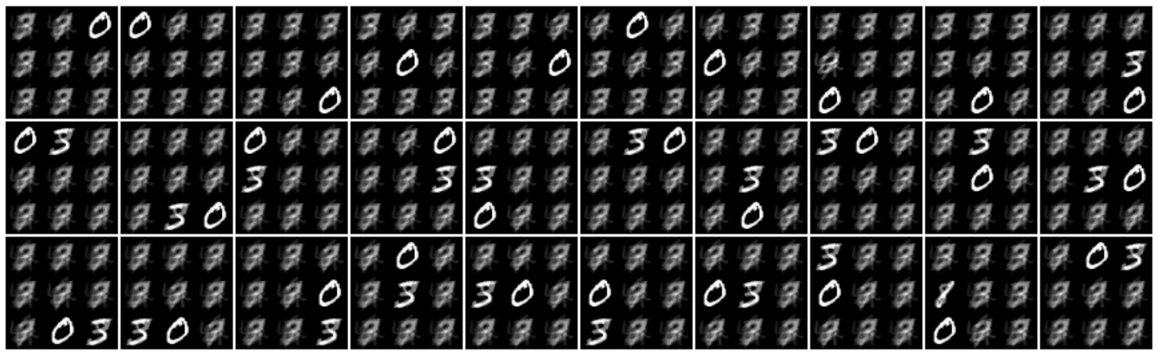


Figure 12: Average states that correspond to the top 30 symbols on MNIST 8-puzzle (sorted by their activation count from left to right and top to bottom)

To understand the symbols that correspond to the low-level subsymbolic representations (i.e., images), we sample 100,000 random states from the environment and get their symbolic representations from the encoder. Then, we take the average of images that correspond to the same symbol. We show the average image that corresponds to the top 30 symbols sorted by their activation counts in Figure 12. We notice that the first nine symbols correspond to different locations of the empty tile (recall that the digit ‘0’ is considered as the empty tile following (Asai & Fukunaga, 2018)), which accounts for 41.5% percent of all activations (i.e., in 41.5% of the time, the encoder only outputs the position of the empty tile). Other symbols correspond to cases where the digit ‘3’ or ‘5’ is near the empty tile.

In Figure 13, some predicted effects are visualized for a given state and actions together with the ground truth effects. We see that the decoder successfully models the slide of the digit ‘0’. When we combine the previous state with the predicted effect, we can have an estimate of the next state which is shown in the right column in Figure 13.

## 6.2 Learned Rules

To train a decision tree for the rule extraction, we collect the set of training examples as follows. Given the current state $\mathbf{x}_t$, the encoder generates the corresponding symbol $\mathbf{z}_t = f(\mathbf{x}_t)$ which is then used as input to the decoder together with the one-hot action vector $a_t$ to predict the effect: $\bar{\mathbf{e}}_t = g(\mathbf{z}_t, a_t)$. Then, we predict the next state $\mathbf{x}_{t+1}$ by summing the predicted effect $\bar{\mathbf{e}}_t$ with the current state ($\bar{\mathbf{x}}_{t+1} = \mathbf{x}_t + \bar{\mathbf{e}}_t$) as in Figure 13. Lastly, we use the encoder to generate the symbol $\bar{\mathbf{z}}_{t+1}$ that corresponds to $\bar{\mathbf{x}}_{t+1}$: $\bar{\mathbf{z}}_{t+1} = f(\bar{\mathbf{x}}_{t+1})$. The decision tree is trained with $\{[\mathbf{z}_t; a_t], \bar{\mathbf{z}}_{t+1}\}$ input-output pairs. This is even more generic than robot experiments where we trained the tree with $\{[\mathbf{z}_t; a_t], c_t\}$ pairs ($c_t$ is the effect category predicted by the decoder) since it allows us to express the goal using the image modality. In both cases, the fundamental idea is the same: training a decision tree with symbolic input-output pairs in order to learn probabilistic rules.

As the last step, we convert the decision paths of the tree into probabilistic PDDL rules as in Section 5.5. As an example, a translated rule from a decision path is as follows,

```
(:action slide_left5
    :precondition (and (not (z9)) (not (z5)) (z3))
```

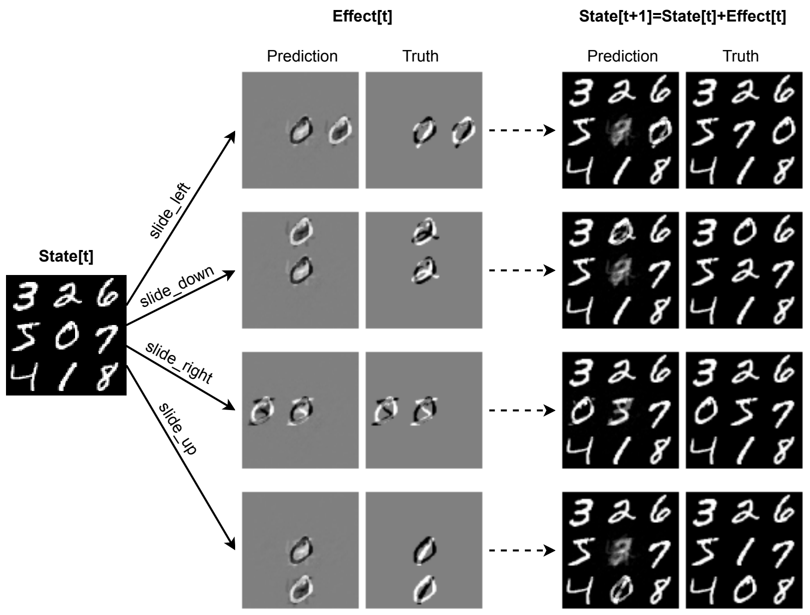


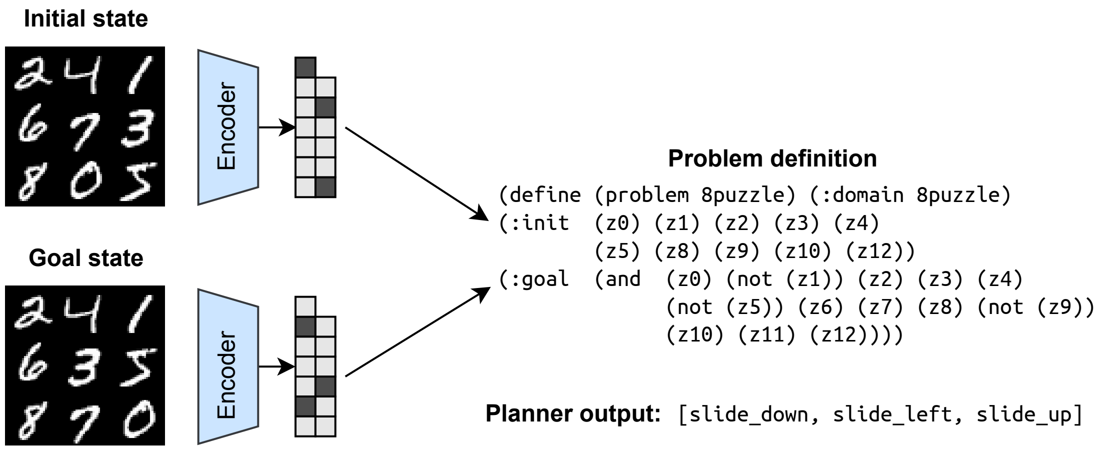
Figure 13: Four different effect predictions are shown together with their ground truths for different actions for a given state. For example, for the ‘slide-right’ action, in the center, ‘0’ is erased and ‘5’ is painted, and on the left, ‘5’ is erased and ‘0’ is painted.

```
:effect (probabilistic
         0.16667 (and (z0) (z1) (not (z2)) (not (z3)) (z4)
                      (z5) (not (z6)) (not (z7)) (z8) (z9)
                      (z10) (not (z11)) (z12))
         0.00758 (and (z0) (z1) (not (z2)) (not (z3)) (z4)
                      (z5) (not (z6)) (not (z7)) (z8) (z9)
                      (z10) (z11) (z12))
         0.68939 (and (z0) (z1) (not (z2)) (z3) (z4) (z5)
                      (not (z6)) (not (z7)) (z8) (z9) (z10)
                      (not (z11)) (z12))
         0.13636 (and (z0) (z1) (not (z2)) (z3) (z4) (z5)
                      (not (z6)) (not (z7)) (z8) (z9) (z10)
                      (z11) (z12))))
```

where predicates `(z0)` ...`z(12)` correspond to activations of each unit in $z$. There are no auxiliary predicates as there are in the tabletop environment; the PDDL file only consists of such translated rules given above.

## 6.3 Planning Examples

Using the generated PPDDL description, our system is requested to output a plan for the goal state from a random initial state. For this, the problem definition (where the current state and the goal state are indicated) is created in PPDDL using the activations of the encoder (see Figure 14).

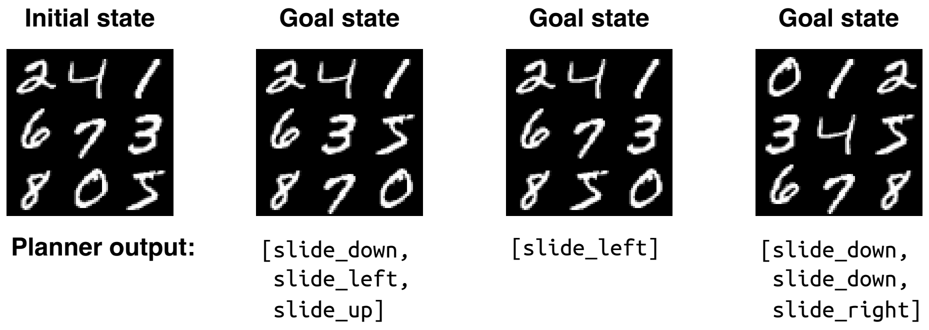


```
(define (problem 8puzzle) (:domain 8puzzle)
(:init  (z0) (z1) (z2) (z3) (z4)
        (z5) (z8) (z9) (z10) (z12))
(:goal  (and  (z0) (not (z1)) (z2) (z3) (z4)
              (not (z5)) (z6) (z7) (z8) (not (z9))
              (z10) (z11) (z12))))
```

```
[slide_down, slide_left, slide_up]
```

Figure 14: The generated plan for the goal state from a random initial state.

As shown in the figure, our system was able to find the correct action sequences in order to reach the given goal configuration. Note that we observed that the output plan only slides the tiles so as to move the empty tile into the correct position. This is a consequence of the system because the encoded activations do not represent the global state but a local state: the position of the empty tile and its neighbors. One can extend the locality by incorporating multiple timestep effects of actions in a similar approach with (Xu et al., 2021). This can be an advantage, or disadvantage, depending on the context and the problem, which will be discussed in Section 7.

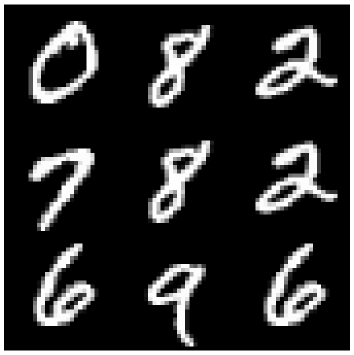


```
[slide_down,      [slide_left]      [slide_down,
 slide_left,                         slide_down,
 slide_up]                           slide_right]
```

Figure 15: Three different goal positions and their respective planner outputs.

As the current formulation cannot capture the global state, we experimented with local state representations. For example, in Figure 15, we set two different arbitrary goals that are one step and three steps away from the initial state (the first and the second goal in Figure 15). The planner outputs the correct plan since it can capture the nearby tile information. However, when asked for the third goal in Figure 15, the generated plan only moves the empty tile to the correct position while disregarding other tiles.

We generated 100 random goal states that are $n$-step away from the corresponding initial state and reported the planning results to quantitatively assess the performance of the method. We also add the results for executing random actions to assess the performance


(a) 8-puzzle w/replacement.

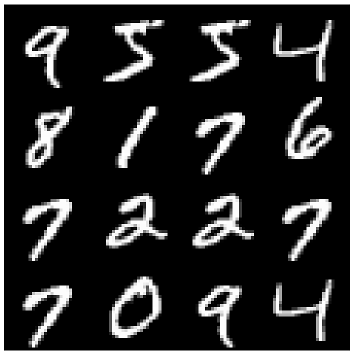

(b) 15-puzzle w/replacement.

Figure 16: In these environments, each digit except '0' may appear more than once.

increment. We report the percentage of plans that successfully move the empty tile to the correct position in Table 4. From the results, we see that DeepSym can successfully move the empty tile into the correct position for different plan lengths.

| | 1-step | 2-step | 3-step | 4-step |
|---|---|---|---|---|
| Random plans | $24.4 \pm 4.2$ | $9.8 \pm 2.3$ | $11.4 \pm 2.1$ | $10.6 \pm 4.0$ |
| DeepSym | $92.6 \pm 5.8$ | $88.0 \pm 8.6$ | $88.8 \pm 7.4$ | $89.0 \pm 7.9$ |

Table 4: The average percentage of successful plans that move the empty tile for different plan lengths over five runs.

### 6.4 Scaling-up to 15-puzzle

This section aims to analyze the performance of the system when we scale up the dimensionality of the environment. Examples in the previous section suggest that our system can correctly identify the empty tile, learn the transition based on the empty tile, and make plans to move the empty tile into different positions. We would like to test whether this is the case for larger environments. Therefore, we scale up the 8-puzzle in two different ways: (1) 8-puzzle with replacement (will be denoted as w/r) and (2) 15-puzzle with replacement. Each digit except '0' (the empty tile) may appear more than once in these versions. We train DeepSym with 14 units for 8-puzzle w/r, and with 15 units for 15-puzzle w/r (see Appendix C for details).

The low-level state-space and effect-space are $112 \times 112 = 12544$ dimensional for 15-puzzle w/r while it stays the same for 8-puzzle w/r. We used the same convolutional architecture with different paddings to ensure the same output size. The most frequently activated symbols are shown in Figures 19 and 20. We give the planning results for these environments in Figure 17. We see that the system moves the empty tile (by sliding other tiles) to the correct position but disregards other tiles.

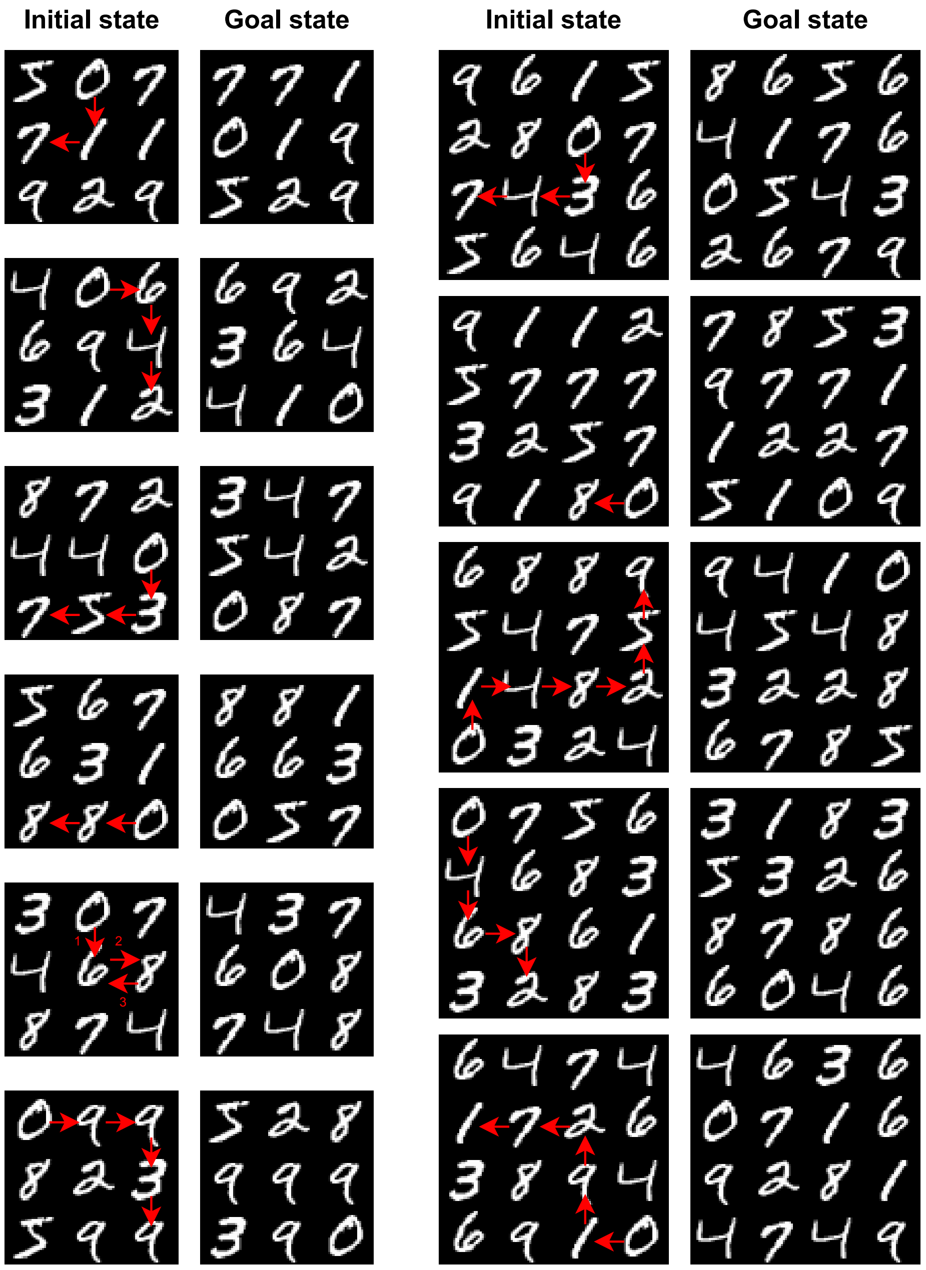


Figure 17: Planning results for 8-puzzle w/r and 15-puzzle w/r. The arrow denotes the movement of the empty tile at each step.

### 6.5 Comparison with Autoencoder

This section aims to compare DeepSym with an autoencoder baseline. We train an autoencoder (as in Asai & Fukunaga, 2018) in these three MNIST $n$-puzzle environments with the same architecture and the same number of bottleneck units as in DeepSym. Given the bottleneck size, it would be impossible for the autoencoder to encode all state-space. The most frequently activated symbols for 8-puzzle, 8-puzzle w/r, and 15-puzzle w/r are shown in Figures 21, 22, and 23, respectively. We train a decision tree for rule learning using the encoder activations to compare the planning performance. Namely, the decision tree is trained with $(f(\mathbf{x}_t), f(\mathbf{x}_{t+1}))$ input-output pairs where $f$ is the encoder network. After the training, we extracted rules from all paths of the tree and constructed a PPDDL description. The planner failed to produce any plan output for random initial and goal states. This is expected since all the state-space cannot be encoded, and therefore, some states are not represented correctly in the PPDDL description. One would need to increase the bottleneck size in order to convert all the state space into PPDDL descriptions. In Asai and Fukunaga (2018), the bottleneck size is set to 25 units (instead of 13 in our experiments) to cover the state space.

## 7. Discussion

A plan corresponds to a sequence of actions to move from an initial state to a goal state. One must chain the effects of actions to predict a future state. Thus, the effects of actions should be known to generate a plan. Therefore, the capability of knowing preconditions of actions and predicting the effects of actions is a requirement for generating a successful plan (Konidaris et al., 2014).

The main difference between DeepSym and approaches that focus on compressing the state representation (e.g., with an autoencoder, Asai & Fukunaga, 2018; Asai et al., 2022, or with a world model, Hafner et al., 2020) is that the learned representations in DeepSym are only due to actions and effects of the agent (Taniguchi et al., 2018). Learning symbols based on the capabilities of the agent allows one to filter-out details of the environment not related to the agent. On the other hand, the approach of compressing the state representation brings its own advantages. One can use a large dataset of states to pre-train an unsupervised model to learn a compact model of the environment, and then use the learned model to train a supervised model for planning or policy learning.

Finding action-independent discrete representations is non-trivial in a large state-space even for the toy examples given in Section 6. In our robot experiments, the autoencoder with discrete units (Asai & Fukunaga, 2018) was shown not to generate a useful representation with a low bottleneck dimension. On the other hand, DeepSym can learn useful and compact representations for planning as it considers actions and effects. For environments that are more realistic for lifelong learning, such as Minecraft (Johnson et al., 2016), the raw state-space is virtually infinite, making it difficult to find a minimal set of meaningful discrete representations without taking actions and action effects into account. On the other hand, action- and effect-based learning allows for an efficient representation of the state space by filtering out the aspects of the environment not relevant to the actions of the agent. For example, in the 8-puzzle environment, the encoder disregards the tiles not near the empty tile since the generated effect does not depend on them. The learned representation allows for generating plans to move the empty tile to different positions. DeepSym learns the minimal set of representations that are needed for the effect prediction of action. Therefore, our system learns action-centric representations, i.e., representations that involve the empty tile. State- and action-based methods are two different (possibly complementary) approaches with advantages and disadvantages. For example, if the problem domain is small, or there exists a large-scale pre-trained model of the environment, encoding all the state space will allow one to solve any encountered problem. However, this approach might be infeasible for larger domains. On the other hand, action-based encoding learns the minimal set of symbols to predict effects at the cost of missing possibly global task requirements (e.g., a specific arrangement in 8-puzzle).

It is theoretically possible to learn a simpler feature-based representation that will be more computationally efficient when compared with deep networks when state, action, and observed effects are all known (Ugur & Piater, 2015a; Konidaris et al., 2018). However, this approach would need manual feature extraction for newly encountered domains, while a differentiable network that can be automatically tuned offers a more uniform and extendible approach.

One thing we observed is that with the narrow bottleneck size (i.e., 13 units for 3248 configurations), the encoder does not represent all the neighbor configurations that are needed for effect prediction. However, when we increase the bottleneck size, the encoder indeed learns all the necessary configurations. Even if the system successfully encodes all the local states, it would still need to symbolically encode the global state to solve the task globally. One approach to encoding the global state might be extending the locality by considering the effects of multiple timesteps (Xu et al., 2021) or partitioning the state into several chunks and representing the change in these chunks separately.

## 8. Conclusion

In this work, we introduced a method that discovers effect- and action-guided object categories, encodes them as discrete symbols, and learns rules that predict action effects. It sustains a general cognitive development progression where symbols are formed, rules are learned, planning is achieved, and verified in execution. Our system contributes to the state-of-the-art by showing the following desirable properties which have not been achieved/shown simultaneously elsewhere:

- We proposed a generic, single pipeline neural solution for mapping raw sensorimotor experience into the symbolic domain.
- The proposed network allows progressive learning of increasingly complex abstractions, exploiting previously-learned abstractions as inputs.
- It is gradient-friendly, so it can be incorporated into any gradient-based machine learning system for more complex processing.
- When compared with the continuous bottleneck layer version of our system, i.e., OCEC, our system performs better in effect category formation leading to more successful action planning. This suggests that instead of post-training clustering of the continuous unit outputs, employing discrete units from the beginning is beneficial.

In future work, we plan to scale up the system by augmenting the perceptual capabilities and the action repertoire of the robot. The ad-hoc perceptual system for determining action effects can be replaced by a state-of-the-art computer vision system. Beyond paired-object relations, graph neural networks can be employed to construct relations between varying numbers of objects. Applying the principles of learning and abstraction of this work to less-constrained scenarios will constitute a major step towards AI-enabled, general-purpose robots.

## Acknowledgments

This research was supported by TUBITAK (The Scientific and Technological Research Council of Turkey) ARDEB 1001 program (project number: 120E274) and by the BAGEP Award of the Science Academy. Additional support is provided by the International Joint Research Promotion Program, Osaka University under the project "Developmentally and biologically realistic modeling of perspective invariant action understanding" and the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research – project with number 22H03670. The numerical calculations reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA). We would like to thank Aysu Sayin and Serkan Bugur for their initial help with the experiments. We thank the reviewers of this paper for their detailed comments.

## Appendix A. Network Architecture and Hyperparameters

### A.1 Tabletop Environment

The network architectures of encoders $f_1$ and $f_2$ are shown in Tables 5a and 5b, respectively. Each convolution is followed by a batch normalization layer and ReLU activation after the normalization. The network architectures of decoders $g_1$ and $g_2$ are shown in Tables 6a and 6b, respectively. Decoders consist of fully connected (FC) layers with no batch normalization.

Adam optimizer (Kingma & Ba, 2015) with AMSGrad (Reddi et al., 2018) is used. The learning rate is set to 0.00005 with a 128 batch size. Each model is trained for 300 epochs, and we select the best model based on the mean square error.

While finding the number of hidden units, we take five runs and record the MSE. We increase the number of units if the one-sided Welch's t-test rejects the null hypothesis $\mathcal{H}_0$ : "Two numbers result in the same MSE" in favor of $\mathcal{H}_1$ : "Increased number results in lower MSE".

We realize that this requires multiple runs and, in fact, is quite inefficient. It would be better if we had a well-defined metric such as the Bayesian Information Criterion (BIC). We did not use BIC since it does not lead to plausible results with deep neural networks that have large parameter sizes.

| Layer | In ch. | Out ch. | Stride | Pad |
|---|---|---|---|---|
| Conv3x3 | 1 | 32 | 1 | 1 |
| Conv3x3 | 32 | 32 | 2 | 1 |
| Conv3x3 | 32 | 64 | 1 | 1 |
| Conv3x3 | 64 | 64 | 2 | 1 |
| Conv3x3 | 64 | 128 | 1 | 1 |
| Conv3x3 | 128 | 128 | 2 | 1 |
| Conv3x3 | 128 | 256 | 1 | 1 |
| Conv3x3 | 256 | 256 | 2 | 1 |
| Global average pooling over channels | | | | |
| FC | 256 | 2 | - | - |
| Gumbel-sigmoid | | | | |
| Number of parameters: 1,174,114 | | | | |

(a) Encoder $f_1$.

| Layer | In ch. | Out ch. | Stride | Pad |
|---|---|---|---|---|
| Conv3x3 | 2 | 32 | 1 | 1 |
| Conv3x3 | 32 | 32 | 2 | 1 |
| Conv3x3 | 32 | 64 | 1 | 1 |
| Conv3x3 | 64 | 64 | 2 | 1 |
| Conv3x3 | 64 | 128 | 1 | 1 |
| Conv3x3 | 128 | 128 | 2 | 1 |
| Conv3x3 | 128 | 256 | 1 | 1 |
| Conv3x3 | 256 | 256 | 2 | 1 |
| Global average pooling over channels | | | | |
| FC | 256 | 1 | - | - |
| Gumbel-sigmoid | | | | |
| Number of parameters: 1,174,145 | | | | |

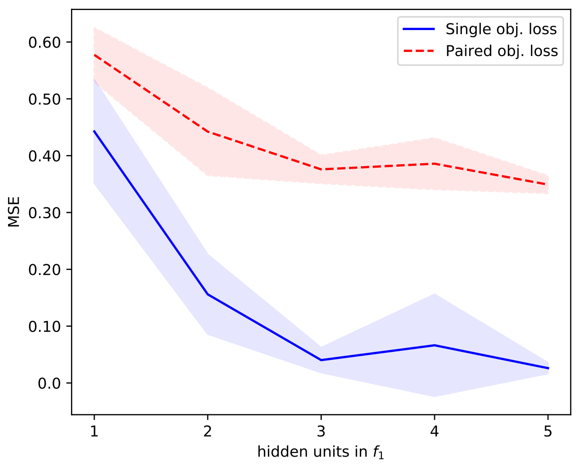
(b) Encoder $f_2$.

| Layer | Input units | Output units |
|---|---|---|
| FC+ReLU | 5 | 128 |
| FC+ReLU | 128 | 128 |
| FC | 128 | 3 |
| Number of parameters: 17,667 | | |

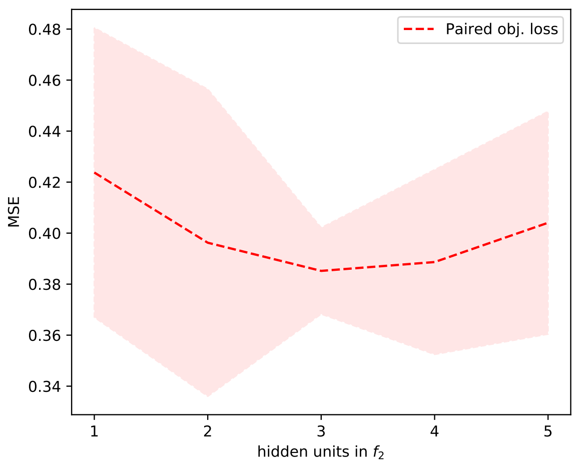
(a) Decoder $g_1$.

| Layer | Input units | Output units |
|---|---|---|
| FC+ReLU | 5 | 128 |
| FC+ReLU | 128 | 128 |
| FC | 128 | 6 |
| Number of parameters: 18,054 | | |

(b) Decoder $g_2$.



(a) MSE vs. the number of units in the bottleneck of $f_1$.



(b) MSE vs. the number of units in the bottleneck of $f_2$.

Figure 18: The mean square error losses for $f_1$-$g_1$ and $f_2$-$g_2$ network pairs. In (a), we also plot the paired object MSE with a single unit for a varying number of units in the bottleneck of $f_1$ to show how different numbers of units affect MSE in new observations).

### A.2 MNIST 8-puzzle Environment

Network architectures of the encoder and the decoder for the MNIST 8-puzzle environment are given in Tables 7 and 8. For 8-puzzle w/r and 15-puzzle w/r versions, the bottleneck size is changed from 13 to 14 and 15, respectively. To ensure the output size for the 15-puzzle, we change the padding of the third and the fourth convolutional layer in the decoder from 1 to 0.

| Layer | In ch. | Out ch. | Stride | Pad |
|---|---|---|---|---|
| Conv4x4 | 1 | 64 | 2 | 1 |
| Conv4x4 | 64 | 128 | 2 | 1 |
| Conv4x4 | 128 | 256 | 2 | 1 |
| Conv4x4 | 256 | 512 | 2 | 1 |
| Global average pooling over channels | | | | |
| FC | 512 | 13 | - | - |
| Gumbel-sigmoid | | | | |
| Number of parameters: 2,763,085 | | | | |

Table 7: The encoder for the 8-puzzle environment.

| Layer | In ch. | Out ch. | Stride | Pad |
|---|---|---|---|---|
| FC | 13+4 | 512 | - | - |
| Reshape (-1, 512) $\rightarrow$ (-1, 512, 1, 1) | | | | |
| ConvT5x5 | 512 | 256 | 1 | 0 |
| ConvT4x4 | 256 | 128 | 2 | 1 |
| ConvT4x4 | 128 | 64 | 2 | 1 |
| ConvT4x4 | 64 | 32 | 2 | 1 |
| ConvT4x4 (no batch norm.) | 32 | 1 | 2 | 1 |
| Number of parameters: 3,976,097 | | | | |

Table 8: The decoder for the 8-puzzle environment. ConvT stands for transposed convolutional layers. The last layer of the decoder does not include a batch normalization layer.

## Appendix B. Using the Straight-Through Estimator

The experiment results with STE on the tabletop environment are reported in Table 9.

## Appendix C. The Number of States in 8-puzzle w/r and 15-puzzle w/r

For the 8-puzzle w/r, the number of possible states increases from $9! = 362880$ to $9 \times 9^8 = 387420489$, which is an increase by about a factor of 1000. In general, the number of states is $n^2 k^{(n^2-1)}$ where $n$ stands for the size of the board (the size is 3 for 8-puzzle and 4 for 15-puzzle), and $k$ is the number of possible digits other than the empty tile. This translates to $\approx 3.29 \times 10^{15}$ states for 15-puzzle w/r. On the other hand, the number of states that

| DeepSym with STE | | | | |
|---|---|---|---|---|
| Category | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Sphere | **92.9 ± 8.6** | 2.5 ± 4.4 | 3.2 ± 6.8 | 1.4 ± 2.3 |
| Cube | 1.4 ± 3.1 | **92.9 ± 10.3** | 3.4 ± 5.6 | 2.3 ± 3.7 |
| Vertical Cylinder | 1.9 ± 4.6 | **93.6 ± 5.4** | 1.8 ± 2.7 | 2.7 ± 3.1 |
| Horizontal Cylinder | 15.8 ± 27.3 | 7.6 ± 10.7 | **74.7 ± 27.0** | 2.0 ± 3.8 |
| Cup | 0.1 ± 0.2 | 0.0 ± 0.0 | 0.0 ± 0.0 | **99.9 ± 0.2** |
| Autoencoder with STE (OBO) | | | | |
| Category | (0, 0) | (0, 1) | (1, 0) | (1, 1) |
| Sphere | **85.1 ± 12.1** | 12.3 ± 9.6 | 2.6 ± 4.3 | 0.0 ± 0.0 |
| Cube | **74.6 ± 17.7** | 20.4 ± 12.0 | 5.0 ± 7.0 | 0.0 ± 0.0 |
| Vertical Cylinder | **76.2 ± 15.6** | 18.3 ± 9.4 | 5.5 ± 8.5 | 0.0 ± 0.0 |
| Horizontal Cylinder | **78.1 ± 14.4** | 19.2 ± 11.1 | 2.8 ± 3.8 | 0.0 ± 0.0 |
| Cup | **92.4 ± 14.1** | 6.6 ± 13.5 | 0.9 ± 2.8 | 0.1 ± 0.2 |

Table 9: The relative assignment frequencies of objects to different symbolic categories. Here, objects vary in their sizes and initial positions. The mean and the standard deviation of 10 runs are reported.

the encoder should represent is $(n-2)^2k^4 + 4(n-2)k^3 + 4k^2$, which translates to 9801 and 32400 states for 8-puzzle w/r and 15-puzzle w/r, respectively. Therefore, we train DeepSym with 14 units for 8-puzzle w/r ($\log_2 9801 \approx 13.26$) and with 15 units for 15-puzzle w/r ($\log_2 32400 \approx 14.98$). In a sense, we use the most strict limit for the number of units.

## Appendix D. Symbols Learned in 8-puzzle w/r and 15-puzzle w/r

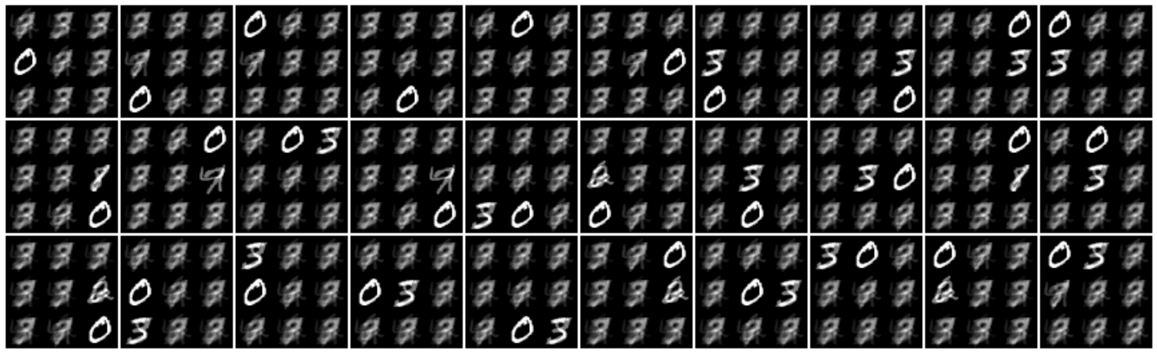


Figure 19: Average states that correspond to symbols learned in 8-puzzle w/r environment.

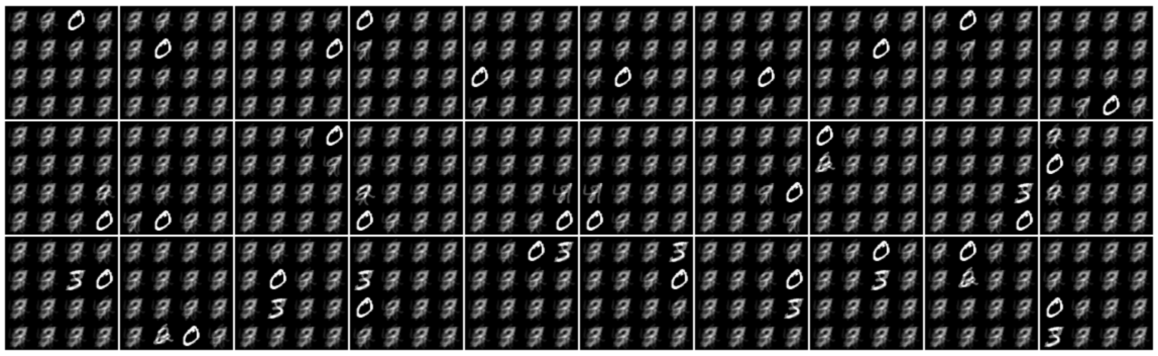


Figure 20: Average states that correspond to symbols learned in 15-puzzle w/r environment.

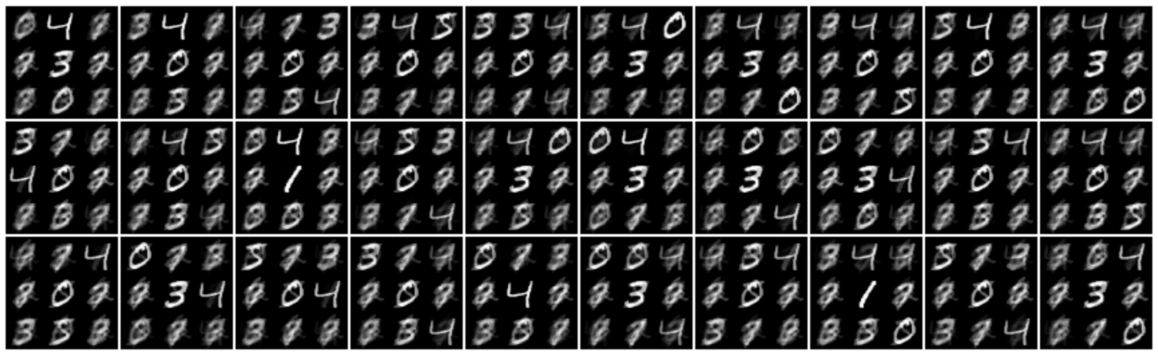


Figure 21: Average states that correspond to autoencoder symbols learned in the 8-puzzle environment.

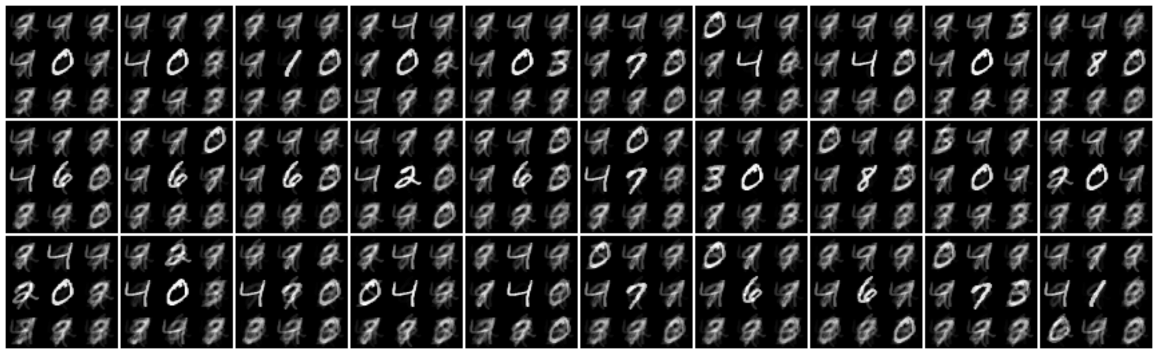


Figure 22: Average states that correspond to autoencoder symbols learned in 8-puzzle w/r environment.

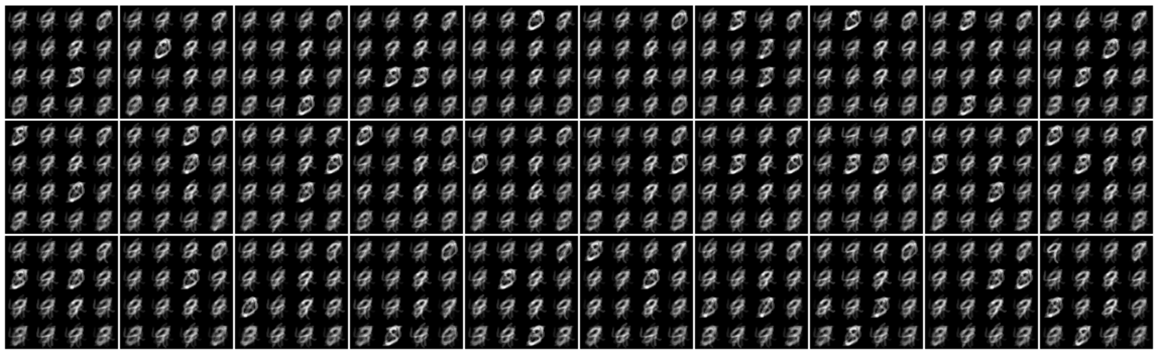


Figure 23: Average states that correspond to autoencoder symbols learned in 15-puzzle w/r environment.

## References

Akbulut, M., Oztop, E., Seker, M. Y., Hh, X., Tekden, A., & Ugur, E. (2021). ACNMP: Skill transfer and task extrapolation through learning from demonstration and reinforcement learning via representation sharing. In *Conference on Robot Learning*, pp. 1896–1907. PMLR.

Asai, M., & Fukunaga, A. (2018). Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

Asai, M., Kajino, H., Fukunaga, A., & Muise, C. (2022). Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, *74*, 1599–1686.

Asai, M., & Muise, C. (2021). Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In *International Joint Conference on Artificial Intelligence*, pp. 2676–2682.

Bengio, Y., Léonard, N., & Courville, A. C. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, *abs/1308.3432*.

Bonet, B., & Geffner, H. (2005). mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, *24*, 933–944.

Callaghan, T., & Corbit, J. (2015). *The Development of Symbolic Representation*, chap. 7, pp. 1–46. John Wiley & Sons, Ltd.

Chitnis, R., Silver, T., Tenenbaum, J. B., Perez, T., & Kaelbling, L. P. (2021). Learning neuro-symbolic relational transition models for bilevel planning. In *Combining Learning and Reasoning: Programming Languages, Formalisms, and Representations.*

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805.*

Gibson, J. J. (2014). *The ecological approach to visual perception: Classic edition.* Psychology Press.

Hafner, D., Lillicrap, T. P., Norouzi, M., & Ba, J. (2020). Mastering atari with discrete world models. In *International Conference on Learning Representations.*

Harnad, S. (1990). The symbol grounding problem. *Physica D*, *42*(1-2), 335–346.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456. PMLR.

James, S., Rosman, B., & Konidaris, G. (2020). Learning portable representations for high-level planning. In *International Conference on Machine Learning*, pp. 4682–4691. PMLR.

Jang, E., Gu, S., & Poole, B. (2017). Categorical reparametrization with Gumbel-softmax. In *International Conference on Learning Representations*.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation.. In *International Joint Conference on Artificial Intelligence*, pp. 4246–4247. Citeseer.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, *abs/1312.6114*.

Klingspor, V., Morik, K., & Rieger, A. D. (1996). Learning concepts from sensor data of a mobile robot. *Machine Learning*, *23*(2-3), 305–332.

Konidaris, G., Kaelbling, L., & Lozano-Perez, T. (2015). Symbol acquisition for probabilistic high-level planning. In *International Joint Conference on Artificial Intelligence*.

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

Konidaris, G., Kaelbling, L., & Lozano-Perez, T. (2014). Constructing symbolic representations for high-level planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, *61*, 215–289.

Kuipers, B., Feigenbaum, E. A., Hart, P. E., & Nilsson, N. J. (2017). Shakey: From conception to history.. *AI Magazine*, *38*(1), 88–103.

Kulick, J., Toussaint, M., Lang, T., & Lopes, M. (2013). Active learning for teaching a robot grounded relational symbols. In *International Joint Conference on Artificial Intelligence*, pp. 1451–1457.

Law, M., Russo, A., & Broda, K. (2018). The complexity and generality of learning answer set programs. *Artificial Intelligence*, *259*, 110–146.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the Trade*, pp. 9–48. Springer.

Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Mota, T., & Sridharan, M. (2019). Commonsense reasoning and knowledge acquisition to guide deep learning on robots.. In *Robotics: Science and Systems*.

Mourao, K., Petrick, R. P., & Steedman, M. (2008). Using kernel perceptrons to learn action effects for planning. In *International Conference on Cognitive Systems*, pp. 45–50. Citeseer.

Murphy, R., & Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT press.

Ozturkcu, O. B., Ugur, E., & Oztop, E. (2020). High-level representations through unconstrained sensorimotor learning. In *International Conference on Development and Learning*.

Petrick, R., Kraft, D., Mourao, K., Pugeault, N., Krüger, N., & Steedman, M. (2008). Representation and integration: Combining robot control, high-level planning, and action learning. In *Proceedings of the 6th International Cognitive Robotics Workshop*, pp. 32–41.

Pisokas, J., & Nehmzow, U. (2005). Experiments in subsymbolic action planning with mobile robots. In *Adaptive Agents and Multi-Agent Systems II, Lecture Notes in AI*, pp. 80–87. Springer.

Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*.

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. In *International Conference on Learning Representations*.

Riley, H., & Sridharan, M. (2019). Integrating non-monotonic logical reasoning and inductive learning with deep learning for explainable visual question answering. *Frontiers in Robotics and AI*, *6*, 125.

Rohmer, E., Singh, S. P. N., & Freese, M. (2013). CoppeliaSim (formerly V-REP): A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. www.coppeliarobotics.com.

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th edition). Pearson.

Seker, M. Y., Imre, M., Piater, J., & Ugur, E. (2019). Conditional neural movement primitives. In *Robotics: Science and Systems*.

Silver, T., Athalye, A., Tenenbaum, J. B., Lozano-Perez, T., & Kaelbling, L. P. (2022a). Learning neuro-symbolic skills for bilevel planning. *CoRR*, *abs/2206.10680*.

Silver, T., Chitnis, R., Kumar, N., McClinton, W., Lozano-Perez, T., Kaelbling, L. P., & Tenenbaum, J. (2022b). Inventing relational state and action abstractions for effective and efficient bilevel planning. *CoRR*, *abs/2203.09634*.

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

Sun, R. (2000). Symbol grounding: A new look at an old idea. *Philosophical Psychology*, *13*(149–172).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *27*.

Taniguchi, T., Ugur, E., Hoffmann, M., Jamone, L., Nagai, T., Rosman, B., Matsuka, T., Iwahashi, N., Oztop, E., Piater, J., et al. (2018). Symbol emergence in cognitive developmental systems: a survey. *IEEE Transactions on Cognitive and Developmental Systems*, *11*(4), 494–516.

Townsend, W. (2000). The BarrettHand grasper-programmably flexible part handling and assembly. *Industrial Robot: An International Journal*, *27*(3), 181–188.

Ugur, E., Oztop, E., & Sahin, E. (2011). Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, *59*(7–8), 580–595.

Ugur, E., & Piater, J. (2015a). Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation*, pp. 2627–2633. IEEE.

Ugur, E., & Piater, J. (2015b). Refining discovered symbols with multi-step interaction experience. In *2015 IEEE-RAS International Conference on Humanoid Robots*, pp. 1007–1012. IEEE.

Ugur, E., Şahin, E., & Oztop, E. (2012). Self-discovery of motor primitives and learning grasp affordances. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3260–3267. IEEE.

Universal Robots (2012). The UR10 collaborative industrial robot. https://www.universal-robots.com/products/ur10-robot. Online; accessed 10 September 2020.

Werner, H., & Kaplan, B. (1963). *Symbol formation*. Wiley.

Wörgötter, F., Agostini, A., Krüger, N., Shylo, N., & Porr, B. (2009). Cognitive agents: A procedural perspective relying on the predictability of Object-Action-Complexes OACs. *Robotics and Autonomous Systems*, *57*(4), 420–432.

Xu, D., Mandlekar, A., Martín-Martín, R., Zhu, Y., Savarese, S., & Fei-Fei, L. (2021). Deep affordance foresight: Planning through what can be done in the future. In *2021 IEEE International Conference on Robotics and Automation*, pp. 6206–6213. IEEE.

Younes, H. L., & Littman, M. L. (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*, *2*, 99.

Yuan, W., Paxton, C., Desingh, K., & Fox, D. (2022). Sornet: Spatial object-centric representations for sequential manipulation. In *Conference on Robot Learning*, pp. 148–157. PMLR.

Zech, P., Haller, S., Lakani, S. R., Ridge, B., Ugur, E., & Piater, J. (2017). Computational models of affordance in robotics: a taxonomy and systematic classification. *Adaptive Behavior*, *25*(5), 235–271.