Learning structured transition models for multi-object manipulation

Victoria Xia  Zi Wang  Leslie Pack Kaelbling

Abstract—We propose to learn the structure and parameters of rule-based probabilistic transition models for manipulation actions with the presence of multiple objects. For each rule, we use deictic references to myopically select relevant objects and learn a neural network predictor on the transition of their states. Our algorithm simultaneously assigns training data to each rule and learns the rule parameters through clustering and EM-like methods. We evaluate learned rules on variants of a simulated, 3D table-top pushing task involving stacks of objects. Comparing to a baseline that uses a vector representation of all the objects in the scene, our approach with a template representation of the model is more data-efficient and performs better under varying numbers of objects in the scene.

I. INTRODUCTION

Consider a household robot preparing dinner in an unfamiliar kitchen, or organizing papers in a cluttered office space. The robot must be able to perform a wide variety of object-manipulation tasks, such as opening cabinets, turning the knob on the stove, gathering papers, and so on. More importantly, the robot must have an understanding of how an action will affect the world state, i.e. the robot configuration and the poses of all of the objects in the room. Only then can the robot plan a sequence of actions to achieve possibly long-horizon, high-level goals like preparing a meal. This work focuses on the acquisition of this understanding, in the form of a transition model. Specifically, given an initial world state and an action, we wish to predict the resulting state of the world after the action is taken.

Even simple actions have complex effects. Consider a robot pushing an object, i.e. the target, on a table. Because of the absence of detailed information about the world, e.g., the imprecise exertion of force or uneven friction of surfaces, the pushing action cannot produce deterministic effects even if only the target is on the table [1]. Moreover, given the presence of additional objects close to the target, the push can be prevented from succeeding or other objects may move as well, such as any on top of the target object or in the path of the push. As a consequence, the resulting change in the world state is not only stochastic, but it also includes the poses of multiple objects; hence, it is very challenging to represent and learn a probabilistic transition model.

In this work, we propose to represent the continuous-space transition distribution using a rule language that is a generalization of PPDDL (probabilistic planning domain description language) operator descriptions to continuous domains. While actions may produce complex effects, the type of effects that may result are often structured and can be grouped into a small number of categories that capture plausible outcomes. By characterizing these structured categories, we obtain rules for how the state of the world changes as a result of an action.

Continuing our simple example of pushing, though the push action may have an effect on the target object, the objects on top the target and the nearby objects, other objects far away or at the opposite direction of the push are less likely to be affected. By capturing these structural relations between objects, henceforth referred to as deictic references [2], [3], we can represent the possible outcomes of an action as a collection of rules. This collection of rules is in turn a representation of a transition model.

To predict how a particular action will affect the objects in a scene, a rule starts with the target object of the action and pull in as context some small number of additional objects related to the target in predefined ways (e.g., objects on top of the target, in contact with the target, within some given radius of the target, etc.). Based on properties of these selected objects only, the rule will make a prediction for how the action will affect some subset of objects in the scene, again chosen by structural relations starting from the target object of the action. All other objects that are not selected by the rule are assumed to remain unchanged.

Because rules are agnostic to the total number of objects in the scene, the same, relatively "small" rules can be applied to a variety of complicated scenes with many objects, making rules a compact and general representation of a transition model. Because each rule is only responsible for the objects selected by the deictic references (i.e., structural relations between objects), it requires fewer training examples to make good predictions and scales easily to a large number of objects in the scene.

Main contributions: We specify the language of rules and how they can be used to predict outcomes of actions. We then introduce our algorithm for learning such rules. Finally, we evaluate our algorithm and the performance of rules on a simulated 3D pushing task.

II. RELATED WORK

The representation and learning of transition models for capturing the effects of actions in the environment has been studied extensively. A variety of representations have been presented, each with their own learning algorithms.
These representations include the representation of actions as STRIPS actions [4], for which proposed learning approaches include logical inference [5], [6], kernel perceptrons [7], etc., for deterministic actions. To consider stochastic action effects, action representations include “schemas” [8], [9] where states consist of some number of discrete sensor values, probabilistic STRIPS operators [10], and “affordances” [11], [12] that capture the relation between an object, an action, and an effect, so that given any two, predictions can be made for the third.

Our work on is mostly inspired by [3], which introduces and presents a learning method for Noisy Deictic Rules (NDRs), an extension of probabilistic STRIPS operators to include a possible noise outcome, as well as deictic references that refer to objects relative to the actions for which predictions are being made. NDRs are learned via a greedy search procedure involving three nested layers of search, where each layer progressively fills in more structure or details of the model. However, our rules differ substantially from NDRs in that, rather than using a predicate-based state description, our rules operate on continuous state spaces described by object features and relations, such as mass and relative pose. As a consequence, our rules represent predictions for the outcomes of stochastic actions as a continuous distribution over resultant object features, rather than as a collection of possible discrete outcomes with associated probabilities.

III. PROBLEM FORMULATION

In this section, we formalize this class of problems, define a new rule structure for specifying probabilistic transition models for these problems, and articulate an objective function for estimating these models from data.

A. Relational domain

A problem domain is given by tuple \(\mathcal{D} = (\mathcal{Y}, \mathcal{P}, \mathcal{F}, \mathcal{A})\) where \(\mathcal{Y}\) is a countably infinite universe of possible objects, \(\mathcal{P}\) is a finite set of properties \(P_i : \mathcal{Y} \mapsto \mathbb{R}, i \in [N_P]\) = \(\{1, \ldots, N_P\}\), and \(\mathcal{F}\) is a finite set of deictic reference functions \(F_i : \mathcal{Y}^{m_i} \mapsto \mathcal{S}^{(\mathcal{Y})}, i \in [N_F]\) where \(\mathcal{S}^{(\mathcal{Y})}\) denotes the powerset of \(\mathcal{Y}\). Each function \(F_i \in \mathcal{F}\) maps from an ordered list of objects to a set of objects, and we define it as

\[
F_i(o_1, \ldots, o_m_i) = \{o | f_i(o, o_1, \ldots, o_{m_i}) = \text{True}, o, o_j \in \mathcal{Y}, \forall j \in [m_i]\},
\]

where the relation \(f_i : \mathcal{Y}^{m_i+1} \mapsto \{\text{True, False}\}\) is defined in terms of the object properties in \(\mathcal{P}\). For example, if we have a location property \(P_{loc}\) and \(m_i = 1\), we can define \(f_1(o, o_1) = 1\) if \(||P_{loc}(o) - P_{loc}(o_1)|| < 0.5\) so that the function \(F_i\) associated with \(f_i\) maps from one object to the set of objects that are within 0.5 distance of its center; here I is an indicator function. Finally, \(\mathcal{A}\) is a set of action templates \(A_i : \mathbb{R}^{d_i} \times \mathcal{Y}^{n_i} \mapsto \Psi, i \in [N_A]\), where \(\Psi\) is the space of executable control programs. Each action template is a function parameterized by continuous parameters \(\alpha_i \in \mathbb{R}^{d_i}\) and a tuple of \(n_i\) objects that the action operates on. In this work, we assume that

\(\mathcal{P}, \mathcal{F}\) and \(\mathcal{A}\) are given

A problem instance is characterized by \(\mathcal{I} = (\mathcal{D}, \mathcal{U})\), where \(\mathcal{D}\) is a domain defined above and \(\mathcal{U} \subseteq \mathcal{Y}\) is a finite universe of objects with \(|\mathcal{U}| = N_U\). For simplicity, we assume that, for a particular instance, the universe of objects remains constant over time. In the problem instance \(\mathcal{I}\), we characterize a state \(s\) in terms of the concrete values of all properties in \(\mathcal{P}\) on all objects in \(\mathcal{U}\); that is, \(s = [P_i(o_j)]_{i=1, j=1}^{N_P, N_U} \in \mathbb{R}^{N_P \times N_U} = \mathcal{S}\). A problem instance induces the definition of its action space \(\mathcal{A}\), constructed by applying every action template \(A_i \in \mathcal{A}\) to all tuples of \(n_i\) elements in \(\mathcal{U}\) and all assignments \(\alpha_i\) to the continuous parameters; namely, \(\mathcal{A} = \{A_i(\alpha_i, [o_{ij}]_{j=1}^{n_i}) | \alpha_{ij} \in \mathcal{U}, \alpha_i \in \mathbb{R}^{d_i}\}\).

B. Sparse relational transition models

In many domains, there is substantial uncertainty, and the key to robust behavior is the ability to model this uncertainty and make plans that respect it. A sparse relational transition model (SPARE) for a domain \(\mathcal{D}\), when applied to a problem instance \(\mathcal{I}\) for that domain, defines a probability density function on the resulting state \(s'\) resulting from taking action \(a\) in state \(s\). Our objective is to specify this function in terms of domain elements \(\mathcal{P}, \mathcal{R}\), and \(\mathcal{F}\) in such a way that it will apply to any problem instance, independent of the number and properties of the objects in its universe. We achieve this by defining the transition model in terms of a set of transition rules, \(\mathcal{T} = \{T_k\}_{k=1}^{K}\) and a score function \(C : \mathcal{T} \times \mathcal{S} \mapsto \mathbb{N}\). The score function takes in as input a state \(s\) and a rule \(T \in \mathcal{T}\), and outputs a non-negative integer. If the output is 0, the rule does not apply; otherwise, the rule can predict the distribution of the next state to be \(p(s' | s, a; T)\). The final prediction of SPARE is

\[
p(s' | s, a; T) = \begin{cases} \frac{1}{|T|} \sum_{T \in \mathcal{T}} p(s' | s, a; T) & \text{if } |T| > 0 \\ N(s, \Sigma_{\text{default}}) & \text{otherwise} \end{cases},
\]

where \(\mathcal{T} = \arg \max_{T \in \mathcal{T}} C(T, s)\) and the matrix \(\Sigma_{\text{default}} = \mathcal{I}_{N_D} \otimes \text{diag}([\sigma_i]_{i=1}^{N_P})\) is the default predicted covariance for any state that is not predicted to change, so that our problem is well-formed in the presence of noise in the input. Here \(\mathcal{I}_{N_D}\) is an identity matrix of size \(N_D\), and \(\text{diag}([\sigma_i]_{i=1}^{N_P})\) represents a square diagonal matrix with \(\sigma_i\) on the main diagonal, denoting the default variance for property \(P_i\) if no rule applies. In the rest of this section, we formalize the definition of transition rules and the score function.

Transition rule \(T = (A, \Gamma, \Delta, \phi_0, \nu_{\text{default}})\) is characterized by an action template \(A\), two ordered lists of deictic references \(\Gamma\) and \(\Delta\) of size \(N_R\) and \(N_A\), a predictor \(\phi_0\) and the default variances \(\nu_{\text{default}} = [\nu_i]_{i=1}^{N_P}\) for each property \(P_i\) under this rule. The action template is defined as operating on a tuple of \(n\) object variables, which we will refer to

\[^{1}\text{There is a direct extension of this formulation in which we encode relations among the objects as well. Doing so complicates notation and adds no new conceptual ideas, and in our example domain it suffices to compute spatial relations from object properties so there is no need to store relational information explicitly, so we omit it from our treatment.}\]
as $O(0) = (O_1)^m \ni O_i \in U, \forall i$. A reference list uses functions to designate a list of additional objects or sets of objects, by making deictic references based on previously designated objects. In particular, $\Delta$ generates a list of objects whose properties affect the prediction made by the transition rule, while $\Delta$ generates a list of objects whose properties are affected after taking an action specified by the action template $A$.

We begin with the simple case in which every function returns a single object, then extend our definition to the case of sets. Concretely, for the $t$-th element $\gamma_t$ in $\Gamma (t \in \mathbb{N})$,

\[ \gamma_t = (F(O_k))^m \in \Gamma \]

where $F \in \mathcal{F}$ is a deictic reference function in the domain, $m$ is the arity of that function, and integer $k_{j} \in [n + t - 1]$ specifies that object $O_{n+t}$ in the object list can be determined by applying function $F$ to objects $(O_k)^m \in \Gamma$. Thus, we get a new list of objects, $O(1) = (O_1)^{m+1}$. So, reference $\gamma_1$ can only refer to the objects $(O_i)^{m+1}$ that are named in the action, and determines an object $O_{n+1}$. Then, reference $\gamma_2$ can refer to objects named in the action or those that were determined by reference $\gamma_1$, and so on.

When the function $F$ in $\gamma_t = (F(O_k))^m \in \Gamma$ returns a set of objects rather than a single object, this process of adding more objects remains almost the same, except that the $O_i$ may denote sets of objects, and the functions that are applied to them must be able to operate on sets. In the case that a function $F$ returns a set, it must also specify an aggregator, $g$, that can return a single value for each property $P \in \mathcal{P}$, aggregated over the set. Examples of aggregators include the mean or maximum values or possibly simply the cardinality of the set.

For example, consider the case of pushing the bottom (block $A$) of a stack of 4 blocks, depicted in Figure 1. Suppose the deictic reference is $F = \text{above}$, which takes one object and returns a set of objects immediately on top of the input object. Then, by applying $F = \text{above}$ starting from the initial set $O_0 = \{A\}$, we get an ordered list of sets of objects $(O_0, O_1, O_2)$ where $O_1 = F(O_0) = \{B\}, O_2 = F(O_1) = \{C\}$.

Returning to the definition of a transition rule, we can now see informally that if the parameters of action template $A$ are instantiated to actual objects in a problem instance, then $\Gamma$ and $\Delta$ can be used to determine lists of input and output objects (or sets of objects). We can use these lists, finally, to construct input and output vectors. The input vector $x$ consists of the continuous action parameters $\alpha$ of action $A$ and property $P_i(O_t)$ for all properties $P_i \in \mathcal{P}$ and objects $O_t \in \mathcal{O}^N$ that are selected by $\Gamma$ in arbitrary but fixed order. In the case that $O_t$ is a set of size greater than one, the aggregator associated with the function $F$ that computed the reference is used to compute $P_i(O_t)$. Similar for the desired output construction, we use the references in the list $\Delta$, initialize $O(0) = O(0)$, and gradually add more objects to construct the output set of objects $\hat{O} = \hat{O}(N\Delta)$. The output vector is $y = [P(\hat{o})]_{\hat{o} \in \hat{O}, P \in \mathcal{P}}$ where $\hat{o}$ is a set of objects, we apply a mean aggregator on the properties of all the objects in $\hat{o}$.

The predictor $\phi_\theta$ is some functional form $\phi$ (such as a feed-forward neural network) with parameters (weights) $\theta$ that will take values $x$ as input and predict a distribution for the output vector $y$. It is difficult to represent arbitrarily complex distributions over output values. In this work, we restrict ourselves to representing a Gaussian distributions on all property values in $y$, encoded with a mean and independent variance for each dimension.

Now, we describe how a transition rule can be used to map a state and action into a distribution over the new state. A transition rule $T = (A, \Gamma, \Delta, \phi_\theta, \psi_{\text{default}})$ applies to a particular state-action $(s, a)$ pair if $a$ is an instance of $A$ and if none of the elements of the input or output object lists is empty. To construct the input (and output) list, we begin by assigning the actual objects $o_1, \ldots, o_n$ to the object variables $O_1, \ldots, O_n$ in action instance $a$, and then successively computing references $\gamma_t \in \Gamma$ based on the previously selected objects, applying the definition of the deictic reference $F$ in each $\gamma_t$ to the actual values of the properties as specified in the state $s$. If, at any point, a $\gamma_t \in \Gamma$ or $\delta_t \in \Delta$ returns an empty set, then the transition rule does not apply. If the rule does apply, and successfully selects input and output object lists, then the values of the input vector $x$ can be extracted from $s$, and predictions are made on the mean and variance values $Pr(y \mid x) = \phi_\theta(x) = \mathcal{N}(\mu_\theta(x), \Sigma_\theta(x))$.

Let $(\mu_{(s)}(x), \Sigma_{(s)}(x))$ be the vector entry corresponding to the predicted Gaussian parameters of property $P_i$ of $j$-th output object set $\hat{o}_j$ and denote $s[o, P_i]$ as the property $P_i$ of object $o$ in state $s$, for all $o \in U$. The predicted distribution
of the resulting state \( p(s' | s, a; T) \) is computed as follows:

\[
p(s'[o, P_i]|s, a; T) = \begin{cases} \sum_{j \in J} N(\mu_{\Delta_j}(x), \Sigma_{\Delta_j}(x)) & \text{if } |J| > 0 \\
N(s[a, P_i], v_i) & \text{otherwise}
\end{cases}
\]

where \( J = \{ j : o \in \partial_j \} \) and \( v_i \in \nu_{\text{default}} \) is the default variance of property \( P_i \) in rule \( T \). There are two important points to note. First, it is possible for the same object to appear in the object-list more than once, and therefore for more than one predicted distribution to appear in its properties in the output vector. In this case, we use the mixture of all the predicted distributions with uniform weights. Second, when an element of the output object list is a set, then we treat this as predicting the same single property distribution for all elements of that set. This strategy has sufficed for our current set of examples, but an alternative choice would be to make the predicted values be changes to the current property value, rather than new absolute values. Then, for example, moving all of the objects on top of a tray could easily specify a change to each of their poses. We illustrate how we can use transition rules to build a SPARE in Fig. 2.

For each transition rule \( T_k \in \mathcal{T} \) and state \( s \in \mathcal{S} \), we assign the score function value to be 0 if \( T_k \) does not apply to state \( s \). Otherwise, we assign the total number of deictic references plus one, \( N_\Gamma + N_\Delta + 1 \), as the score. The more references there are in a rule that is applicable to the state, the more detailed the match is between the rules conditions and the state, and the more specific the predictions we expect it to be able to make.

C. Learning SPAREs from data

We frame the problem of learning a transition model from data in terms of conditional likelihood. The learning problem is, given a problem domain description \( \mathcal{D} \) and a set of experience \( \mathcal{E} \) tuples, \( \mathcal{E} = \{(s^{(i)}, a^{(i)}, s^{(i)})\}_{i=1}^n \), find a SPARE \( \mathcal{T} \) that minimizes the loss function:

\[
\mathcal{L}(T; \mathcal{D}, \mathcal{E}) = -\frac{1}{n} \sum_{i=1}^n \log \Pr(s^{(i)} | s^{(i)}, a^{(i)}; T) .
\]

Note that we require all of the tuples in \( \mathcal{E} \) to belong to the same domain \( \mathcal{D} \), and require for any \( (s^{(i)}, a^{(i)}, s^{(i)}) \) in \( \mathcal{E} \) that \( s^{(i)} \) and \( s^{(i)} \) belong to the same problem instance, but individual tuples may be drawn from different problem instances (with, for example, different numbers and types of objects). In fact, to get good generalization performance, it will be important to vary these aspects across training instances.

IV. LEARNING ALGORITHM

We describe our learning algorithm in three parts. First, we introduce our strategy for learning \( \phi_\theta \), which predicts a Gaussian distribution on \( y \), given \( x \). Then, we describe our algorithm for learning reference instances \( \Gamma \) and \( \Delta \) for a single transition rule, which enable the extraction of \( x \) and \( y \) from \( \mathcal{E} \). Finally, we present an EM method for learning multiple rules.

A. Distributional prediction

For a particular transition rule \( T \) with associated action template \( A \), once \( \Gamma \) and \( \Delta \) have been specified, we can extract input and output features \( x \) and \( y \) from a given set of experience samples \( \mathcal{E} \). From \( x \) and \( y \), we would like to learn the transition rule’s predictor \( \phi_\theta \) to minimize Eq. (2). Our predictor takes the form \( \phi_\theta(x) = N(\mu_\theta(x), \Sigma_\theta(x)) \). We use the method of [13], where two neural-network models are learned as the function approximators, one for the mean prediction \( \mu_\theta(x) \) parameterized by \( \theta_1 \) and the other for the diagonal variance prediction \( \Sigma_\theta(x) \), parameterized by \( \theta_2 \).

We optimize the negative data-likelihood loss function

\[
\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \left[ \log \prod_{r \in \Gamma_i} \phi_\theta(x^{(i), r}) \prod_{a \in \Delta_i} \phi_\theta(y^{(i), a}) + \log \det \Sigma_\theta(x^{(i)}) \right]
\]

by alternatively optimizing \( \theta_1 \) and \( \theta_2 \). That is, we alternate between first optimizing \( \theta_1 \) with fixed covariance, and then optimizing \( \theta_2 \) with fixed mean.

Let \( \mathcal{E}_T \in \mathcal{E} \) be the set of experience tuples to which rule \( T \) applies. Then once we have \( \theta \), we optimize the default variance of the rule \( T = (A, \Gamma, \Delta, \phi_\theta, \nu_{\text{default}}) \) by optimizing \( \mathcal{L}(\{T\}; \mathcal{D}, \mathcal{E}_T) \). It can be shown that these loss-minimizing values for the default predicted variances \( \nu_{\text{default}} \) are the empirical averages of the squared deviations for all unpredicted objects (i.e., those for which \( \phi_\theta \) does not explicitly make predictions), where averages are computed separately for each object property.

We use \( \theta, \nu_{\text{default}} \leftarrow \text{LEARNDIST}(\mathcal{D}, \mathcal{E}, \Gamma, \Delta) \) to refer to this learning and optimization procedure for the predictor parameters and default variance.

Algorithm 1 Greedy procedure for constructing \( \Gamma \).

1: procedure \textsc{GreedySelect}(\mathcal{D}, \mathcal{E}, A, \Delta, N_{\Gamma})
2: \hspace*{1em} train model using \( \Gamma_0 = \emptyset \), save loss \( L_0 \)
3: \hspace*{1em} \( i \leftarrow 1 \)
4: \hspace*{1em} while \( i \leq N_{\Gamma} \) do
5: \hspace*{1em} \hspace*{1em} \( \gamma_i \leftarrow \text{None} \); \( L_i \leftarrow \infty \)
6: \hspace*{1em} \hspace*{1em} for all \( \gamma \in R_i \) do
7: \hspace*{1em} \hspace*{1em} \hspace*{1em} \( \Gamma_i \leftarrow \Gamma_{i-1} \cup \{ \gamma \} \)
8: \hspace*{1em} \hspace*{1em} \( \theta, \nu_{\text{default}} \leftarrow \text{LEARNDIST}(\mathcal{D}, \mathcal{E}_{\text{train}}, \Gamma_i, \Delta) \)
9: \hspace*{1em} \hspace*{1em} \( l \leftarrow \mathcal{L}(T_{\Gamma_i}; \mathcal{D}, \mathcal{E}_{\text{val}}) \)
10: \hspace*{1em} \hspace*{1em} if \( l < L_i \) then \( L_i \leftarrow l \); \( \gamma_i \leftarrow \gamma \)
11: \hspace*{1em} \hspace*{1em} if \( L_i < L_{i-1} \) then \( \Gamma_i \leftarrow \Gamma_{i-1} \cup \{ \gamma_i \} \); \( i \leftarrow i + 1 \)
12: \hspace*{1em} else break

B. Single rule

In the simple setting where only one transition rule \( T \) exists in our domain \( \mathcal{D} \), we show how to construct the input and output reference lists \( \Gamma \) and \( \Delta \) that will determine the vectors \( x \) and \( y \). Suppose for now that \( \Delta \) and \( \nu_{\text{default}} \) are fixed, and we wish to learn \( \Gamma \). Our approach is to incrementally build up \( \Gamma \) by adding \( \gamma_i = (F(O_k), \nu_k)_{k=1}^m \) tuples one at a time via a greedy selection procedure. Specifically, let \( R_i \) be the universe of possible \( \gamma_i \), split the experience samples \( \mathcal{E} \) into a training set \( \mathcal{E}_{\text{train}} \) and a validation set \( \mathcal{E}_{\text{val}} \), and initialize the list \( \Gamma \) to be \( \Gamma_0 = \emptyset \). For each \( i \), compute \( \gamma_i = \arg \min_{\gamma \in R_i} \mathcal{L}(T_{\Gamma_i}; \mathcal{D}, \mathcal{E}_{\text{val}}) \), where \( \mathcal{L} \)
in Eq. (2) evaluates a sparse \( T \), with a single transition rule \( T = (A, \Gamma_i \cup \{ \gamma \}, \Delta, \phi_0, \psi_{\text{default}}) \), where \( \theta \) and \( \psi_{\text{default}} \) are computed using the \textsc{LearnDist} described in Section IV-A. If the value of the loss function \( L(T\gamma_i; D, E_{\text{val}}) \) is less than the value of \( L(T\gamma_{i-1}; D, E_{\text{val}}) \), then we let \( \Gamma_i = \Gamma_{i-1} \cup \{ \gamma \} \) and continue. Else, we terminate the greedy selection process with \( \Gamma = \Gamma_{i-1} \), since further growing the list of deictic references hurts the loss. We also terminate the process when \( i \) exceeds some predetermined maximum allowed number of input deictic references, \( N_T \). Pseudocode for this algorithm is provided in Algorithm I.

In our experiments we set \( \Delta = \Gamma \) and construct the lists of deictic references using a single pass of the greedy algorithm described above. This simplification is reasonable, as the set of objects that are relevant to predicting the transition outcome often overlap substantially with the objects that are affected by the action. Alternatively, we could learn \( \Delta \) via an analogous greedy procedure nested around or, as a more efficient approach, interleaved with, the one for learning \( \Gamma \).

C. Multiple rules

Our training data in robotic manipulation tasks are likely to be best described by many rules instead of a single one, since different combinations of relations among objects could be present in different states. For example, we may have one rule for pushing a single object and another rule for pushing a stack of objects. We now address the case where we wish to learn \( K \) rules from a single experience set \( \mathcal{E} \), for \( K > 1 \). We do so via initial clustering to separate experience samples into \( K \) clusters, one for each rule to be learned, followed by an EM-like approach to further separate samples and simultaneously learn rule parameters.

To facilitate the learning of our model, we will additionally learn membership probabilities \( Z = (z_{i,j})_{i=1}^{\mathcal{E}}, j=1 \) \(^{K}\), where \( z_{i,j} \) represents the probability that the \( i \)-th experience sample is assigned to transition rule \( T_j \), and \( \sum_{j=1}^{K} z_{i,j} = 1 \) for all \( i \in |\mathcal{E}| \). We initialize membership probabilities via clustering, then refine them through EM.

Because the experience samples \( \mathcal{E} \) may come from different problem instances and involve different numbers of objects, we cannot directly run a clustering algorithm such as \( k \)-means on the \((s, a, s')\) samples themselves. Instead, we first learn a single transition rule \( T = (A, \Gamma, \Delta, \phi_0, \psi_{\text{default}}) \) from \( \mathcal{E} \) using the algorithm in Section IV-B, use the resulting \( \Gamma \) and \( \Delta \) to transform \( \mathcal{E} \) into \( \mathcal{X} \) and \( \mathcal{Y} \), and then run \( k \)-means clustering on the concatenation of \( \mathcal{X}, \mathcal{Y} \) and values of the loss function when \( T \) is used to predict each of the samples. For each experience sample, the squared distance from the sample to each of the \( K \) cluster centers is computed, and membership probabilities for the sample to each of the \( K \) transition rules to be learned are initialized to be proportional to the (multiplicative) inverses of these squared distances.

Before introducing the EM-like algorithm that simultaneously improves the assignment of experience samples to transition rules and learns details of the rules themselves, we make a minor modification to transition rules to obtain mixture rules. Whereas a probabilistic transition rule has been defined as \( T = (A, \Gamma, \Delta, \phi_0, \psi_{\text{default}}) \), a mixture rule is \( T = (A, \pi_T, \pi_\Delta, \Phi) \), where \( \pi_T \) represents a distribution over all possible lists of input reference \( \Gamma \) (and similarly for \( \pi_\Delta \) and \( \Delta \)), of which there are a finite number, since the set of available reference functions \( F \) is finite, and there is an upper bound \( N_T \) on the maximum number of references \( \Gamma \) may contain.

As before, we focus on the case where for each mixture rule, \( \Gamma^{(k)} = \Delta^{(k)}, k \in [K] \), and \( \pi_T = \pi_\Delta \) as well. Our EM-like algorithm is then as follows:

1) For each \( j \in [K] \), initialize distributions \( \pi_T = \pi_\Delta \) for mixture rule \( T_j \) as follows. First, use the algorithm in Section IV-B to learn a transition rule on the weighted experience samples \( \mathcal{E}_{Z_j} \) with weights equal to the membership probabilities \( Z_j = (z_{i,j})_{i=1}^{\mathcal{E}}, j=1 \) \(^{K}\). In the process of greedily assembling reference lists \( \Gamma = \Delta \), data likelihood loss function values are computed for multiple explored shells, in addition to the shell \( \Gamma = \Delta \) that was ultimately selected. Initialize \( \pi_T = \pi_\Delta \) to distribute weight proportionally, according to data likelihood, for these explored shells: \( \pi_T (\Gamma) = \exp(-L(T\Gamma; D, \mathcal{E}_{Z_j}))/\chi, \) where \( T\Gamma \) is the sparse model with a single transition rule \( T = (A, \Gamma, \Delta = \Gamma, \phi_0) \), and \( \chi = (1 - \epsilon) \sum_{\Gamma} \exp(-L(T\Gamma; D, \mathcal{E}_{Z_j}))/\chi \). This normalization factor \( \chi \) is factored so that the total weight assigned by \( \pi_T \) to explored shells is \( 1 - \epsilon \). The remaining \( \epsilon \) probability weight is distributed uniformly across unexplored shells.

2) For each \( j \in [K] \), let \( T_j = (A, \pi_T, \pi_\Delta, \Phi) \), where we have dropped subscripting according to \( j \) for notational simplicity:
   a) For \( k \in [K] \), train predictor \( \Phi_k = (\Gamma^{(k)}, \Delta^{(k)}, \phi_{g^{(k)}}, \psi_{d^{(k)}}) \) using the procedure in Section IV-B on the weighted experience samples \( \mathcal{E}_{Z_j} \), where we choose \( \Gamma^{(k)} = \Delta^{(k)} \) to be the list of references with \( k \)-th highest weight according to \( \pi_T = \pi_\Delta \).
   b) Update \( \pi_T = \pi_\Delta \) by redistributing weight among the top \( K \) shells according to a voting procedure where each
training sample “votes” for the shell whose predictor minimizes the validation loss for that sample. In other words, the $i$-th experience sample $\mathcal{E}^{(i)}$ votes for mixture rule $v(i) = k$ for $k = \arg \min_{k \in [k]} \mathcal{L}(\phi_k; D, \mathcal{E}^{(i)})$. Then, shell weights are assigned to be proportional to the sum of the sample weights (i.e., membership probability of belonging to this rule) of samples that voted for each particular shell: the number of votes received by the $k$-th shell is $V(k) = \sum_{i=1}^{\sum_k} 1_{v(i) = k} \cdot z_{i,j}$, for indicator function $1_k$ and $k \in [k]$. Then, $\pi_t(k)$, the current $k$-th highest value of $\pi_t$, is updated to become $V(k)/\xi$, where $\xi$ is a normalization factor to ensure that $\pi_t$ remains a valid probability distribution. (Specifically, $\xi = (\sum_{k=1}^K \pi_t(k)) / (\sum_{k=1}^K V(k))$.)

3) Update membership probabilities by scaling by data likelihoods from using each of the $K$ rules to make predictions: $z_{i,j} = z_{i,j} \cdot \exp(-\mathcal{L}(T_j; D, \mathcal{E}^{(i)}))/\zeta$, where $\exp(-\mathcal{L}(T_j; D, \mathcal{E}^{(i)}))$ is the data likelihood from using mixture rule $T_j$ to make predictions for the $i$-th experience sample $\mathcal{E}^{(i)}$, and $\zeta = \sum_{j=1}^J z_{i,j} \cdot \exp(-\mathcal{L}(T_j; D, \mathcal{E}^{(i)}))$ is a normalization factor to maintain $\sum_{j=1}^K z_{i,j} = 1$.

4) Repeat Steps 2 and 3 some fixed number of times, or until convergence.

V. EXPERIMENTS

We apply our approach, SPARE, to a challenging problem of predicting pushing stacks of blocks on a cluttered table top. The object universe $\mathcal{U}$ is composed of blocks of different sizes and weight, the property set $\mathcal{P}$ includes shapes of the blocks (width, length, height) and the 3-D position of the block. We have one action template, $\text{push}(\alpha, \alpha)$, which pushes toward a target object $o$ with parameters $\alpha \in \mathbb{R}^4$. We simulate this 3D domain using the physically realistic PyBullet [14] simulator with Gaussian noise on the action parameters.

We consider the following deictic references in the reference collection $F$: (1) above($O$), which returns the objects immediately above $O$; (2) above*($O$), which returns all of the objects that are above $O$; (3) below($O$), which returns the objects immediately below $O$; (4) nearest($O$), which returns the object that is closest to $O$.

We first compare our single-rule learning approach to the baseline, a neural network function approximator that takes in as input the current state and action parameter, and outputs the predicted Gaussian distribution of the next state. In each problem instance, a robot pushes a block with the presence of other blocks (Fig. 3). Figure 4(a) shows the performance, as measured by the log data likelihood, as a function of the number of extra blocks when a stack of 3 blocks is pushed. As more objects are added to the table, baseline performance drops as the presence of these additional objects appear to complicate the scene and the baseline is forced to consider more objects when making its predictions. However, performance of the SPARE approach remains unchanged, as deictic references are used to select just the three blocks in the stack as input in all cases, regardless of the number of extra blocks on the table. Fig. 4(b) plots the data likelihood as a function of the number of training samples. Both our approach and the baseline benefit from having more training samples, but our approach is much more sample efficient and achieves good performance within only a thousand training samples.

Now we put our approach in a more general setting where multiple transition rules need to be learned for prediction of the next state. Our approach adopts an EM-like procedure to assign each training sample its distribution on the transition rules and learn each transition rule with re-weighted training samples. First, we construct a training dataset and 70% of it is on pushing 4-block stack. Our EM approach is able to concentrate to the 4-block case as shown in Fig. 5(a).

Fig. 5(b) tracks the assignment of samples to rules over the same five runs of our EM procedure. The three curves correspond to the three stack heights in the original dataset, and each shows the average weight assigned to the “target” rule among samples of that stack height, where the target rule is the one that starts with a high concentration of samples of that particular height. At iteration 0, we see that the rules were initialized such that samples were assigned 70% probability of belonging to specific rules, based on stack height. As the algorithm progresses, the samples separate further, suggesting that the algorithm is able to separate samples into the correct groups.
REFERENCES


