Learning composable models of parameterized skills

Leslie Pack Kaelbling and Tomás Lozano-Pérez

Abstract— **There has been a great deal of work on learning new robot skills, but very little consideration of how these newly acquired skills can be integrated into an overall intelligent system. A key aspect of such a system is compositionality: newly learned abilities have to be characterized in a form that will allow them to be flexibly combined with existing abilities, affording a (good!) combinatorial explosion in the robot's abilities. In this paper, we focus on learning models of the preconditions and effects of new parameterized skills, in a form that allows those actions to be combined with existing abilities by a generative planning and execution system.**

I. Introduction

Our overall goal is to design intelligent robots that can operate flexibly and robustly in complicated, highly variable domains, from homes to factories to warehouses. Basic competences, such as the ability to reason about kinematics, free space, collisions, path planning, visibility, and even basic manipulation, are critical in almost every domain and can be carefully engineered into a robot before it is deployed. But for every new domain or even domain instance, a robot will need to be able to acquire and use new skills that are appropriate to that domain.

There has been a great deal of work on learning new robot skills, but very little consideration of how these newly acquired skills can be integrated into an overall intelligent system. A key aspect of such a system is *compositionality*: newly learned abilities have to be characterized in a form that will allow them to be flexibly combined with existing abilities, affording a (good!) combinatorial explosion in the robot's abilities.

In this paper, we focus on learning models of the preconditions and effects of new parameterized skills, in a form that allows those actions to be combined with existing abilities by a generative planning and execution system. We do this in the context of "task and motion" planning (TAMP) domains, which operate in hybrid (continuous and discrete state and action). We *do not* take a position on how the new skills are acquired: it could be via learning from demonstration, reinforcement learning, or even hand programming. We assume that they operate in a closed loop, mapping sensing to actuation until some termination condition is reached. They might include operations such as pouring, pushing, stirring the contents of a bowl, screwing in a bolt, etc. In our current implementation we assume that parameterized skills terminate with all the objects at rest.

In this setting, both the process of learning and the process of using the learned ability are highly leveraged by existing competences. When the skill itself or the model is not yet well learned, the robot can use its existing abilities to put itself in a position for active learning, for example, by planning a path and moving around a table to pick up a bottle to try to pour from. Most importantly, results of learning can immediately be combined with existing abilities, so once the robot knows how to pour, it can use that skill in service of tidying a kitchen or changing the oil in an engine.

Given a new parameterized skill, the objective of this work is to learn a model of its effects in the world in the form of an *operator description*. The operator description provides a high-level, approximate model of preconditions and effects of the action for the TAMP planner. Learning detailed models of these actions, e.g., to predict precise rate of flow during pouring, would require excessive amounts of training data. Furthermore, such models tend to be highly sensitive to physical parameters that would not be accurately known during planning.

We take a goal-oriented view of operator descriptions: they are focused on achieving some resulting condition in the domain, and so we must learn to characterize the world conditions in which executing the new action will lead to the desired outcome. These conditions are known as the *pre-image* of the result under the action. We use supervised learning to map features of the starting state, the action, and the result condition to a Boolean value, which is true if the starting state is in the pre-image and false otherwise. The learning process is started with an initial supervised training set; it is then continued actively, with the robot trying to solve problems that it encounters, generating what it thinks is an appropriate action instance, executing it, and then scoring the result. This actively gathered data is then fed back into the classifier, which improves the robot's understanding of the pre-image of the operation and ultimately its ability to choose actions effectively in order to achieve its goals.

This paper outlines the framework and learning methods in detail and presents a simple pilot experiment in which the robot learns, in simulation, how to use a new pushing skill to put an object into a relatively small region on the table, in the presence of unmodeled rotation and slippage.

II. Related work

There is a huge, and rapidly growing, body of literature that addresses robot "skill" learning, both from demonstrations [1], [2] and via reinforcement learning [3], [4]. Much

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA lpk@mit.edu, tlp@mit.edu. We gratefully acknowledge support from NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

of this work is focused on direct (model-free) learning of individual motor policies (skills), such as batting a T-ball or making a pool shot, with little consideration of how to combine the skills to achieve a larger task. More recently, there has been progress in learning more modular and composable policies [5], [6], [7], [8], in the form of parameterized skills, that can work together and apply in a broader range of contexts. However motor-policy-based methods tend to be limited in their generalization since they are essentially modeling closely related classes of robot trajectories. But, relatively small changes in a task (an object is out of reach or there is an object in the way) call for fundamentally different trajectories from the robot (moving around the table, moving an object out of the way).

Instead of learning policies directly, one can learn models of the actions and then derive policies via planning [4], [9]. When feasible, this generalizes much better than model-free methods but may require more experience to learn an adequate model.

One approach to model-based robot learning aims to derive symbolic characterizations of tasks from observing demonstrations, possibly with the aid of additional annotations from the human [10], [11], [12]. More recent systems of this type [13], [14] combine task-level planning, perception and motion-planning. These approaches. however, have been limited by the inability of existing symbolic planners to deal with the complexities of geometry and robot kinematics. In our work, we leverage the BHPN planner [15] which integrates symbolic planning, motion planning and decision-theoretic planning, so it can substantially generalize symbolic task specifications to apply to a wide range of situations.

Instead of relying on demonstrations, in principle one could learn rules by interacting with the world. One approach to learning parameterized rules suitable for planning in observable stochastic domains, although it is limited to discrete state and action spaces, is that of Pasula et al. [16], which has served as the basis of subsequent work in learning planning rules for robotics [17].

A classic objection to approaches based on symbolic representations is that the symbols must be defined by human input. Very recently, there has been progress on learning symbolic representations for continuous domains [18], [19], [20]. However, this work deals with relatively simple actions that do not involve parameters.

There is work on learning to aid symbolic planning, both to speed up planning and to acquire the action models [21], [22], [23], but the methods generally apply only to discrete symbolic domains.

III. Framework and methods

We assume that a parameterized skill has already been learned or programmed, and it is our goal to learn a characterization of its pre-image and effects that can be used by a task-and-motion planner. A parameterized skill will, in general, be a closed-loop policy that runs for some time and then terminates, potentially yielding an observation characterizing its result.

For the purposes of planning, we describe states of the world in terms of conditions characterizing particular aspects of the world state, such as the pose of an object or a relationship between objects, for example, that one object is on top of another. Using this same vocabulary of conditions, we formulate *operator* descriptions to describe the pre-images and effects of parameterized skills.

We are committed to operation on a real robot, which has uncertainty as part of its essence: there is both future-state uncertainty due to stochastic outcomes of actions and and current-state uncertainty due to partial observability. Learning in partially observable domains is very difficult; we will assume in this work that the relevant aspects of training examples have been sufficiently well observed to enable a learning process that operates as if there were no observation noise. Section IV-D describes how the learned models are adapted for use in partially observable domains.

A. Pre-image backchaining

Particularly in very high-dimensional domains, it is critical for planning to be goal-driven, so that the algorithm can focus on selecting actions that change aspects of the domain that are relevant to the problem. The goal can affect the planning process either through a heuristic evaluation function or by serving as the root node of the search. We pursue the second strategy, searching "backward" from a node representing the set of world states that satisfy the goal condition, and repeatedly computing the pre-image of the goal under candidate actions. The form of the learned operators is such, however, that they could be used in a forward-search-based TAMP planner, such as FFRob [24] or the system of Lagriffoul et al. [25].

The *pre-image* of a set of states G under an action a, $\mathrm{Pre}(G, a)$, is a set of states such that, if action a were executed in one of those states, the resulting state would be in the set G. A pre-image backchaining search, then, starts with the goal as the root of the search and repeatedly computes the pre-image under sequences of different actions until a path is found to a set of states that contains the *initial state*. That path is a sequence of actions that, when executed from leaf to root, will move the system from the initial state to a state in the goal set.

To integrate a new parameterized skill into an overall high-level pre-image backchaining planning system, we need to be able to represent and compute action pre-images. In our domains of interest, operator descriptions are "lifted" in the sense that they are parameterized by the particular objects they are intended to operate on. An operator description comprises:

- a set of *variables* naming objects over which the operation is lifted;
- a *skill*, with parameters that govern the details of its operation (e.g., how far to move or how hard to grasp);
- a primary *result condition*, which typically names some objects and parameter values;
- a *pre-image*, in the form of a conjunction of conditions

defined on the variables and possibly introducing additional parameters;
- a possibly complex *constraint* on the values of all the parameters that occur in the result, the skill, and the pre-image conditions; and
- optionally, a set of other "side-effect" conditions that result from executing this action.

An operator description is *correct* if and only if, for all bindings of objects in the world to the variables of the operator and for all choices of parameters of the pre-image conditions, result conditions, and parameterized skill satisfying the parameter constraint, if the pre-image conditions hold in a world state and the parameterized skill is executed, then the result condition will hold in the resulting world state (and any other changes will be appropriately characterized in the side effects). In general, we cannot expect to learn completely correct operators from data; failures introduced by approximate operators are handled by the execution monitoring and replanning mechanisms of BHPN. Any prediction failures will generate training data that can improve the model and prevent future failures of that type.

In general, it may be difficult to characterize the entire pre-image in detail, but it is usually enough to characterize a subset of the pre-image of an action, as long as it offers a substantial set of ways to achieve the result condition. Among correct operator descriptions, we prefer those with larger ("weaker") pre-images. Note that it is also not necessarily critical to model all possible results of all possible actions: if there are some outcomes that are generally desirable, it is enough to learn models of actions that will drive the system into desirable states.

In a hybrid domain, in which operators are parameterized by continous variables, there are an infinite number of possible instantiations of each operator. We make use of procedures called *generators* to sample useful operator instances. A generator typically takes as input values of parameters that are bound in the result and produces samples of parameter values for free variables that occur in pre-image conditions and in the skill's parameters. The primary objective of the generator is to produce parameter values so that the complete set of parameters (in pre-image conditions, skill, and result condition) satisfy the pre-image constraint. For example, picking the hand pose and displacement values so as to push an object into a target region. A secondary objective is to produce multiple bindings that are diverse: they need to differ substantially from one another, if possible, in case a particular set generates preconditions that are unachievable (e.g., the robot is unable to make the necessary space clear, or the energy resources are not available.)

B. Learning operator descriptions

The learning problem we address is: given a parameterized skill $\alpha(\theta_a)$ and result condition $\phi(O, \theta_e)$, where θ_a and θ_e are parameter vectors and O is a vector of object names, learn one or more operator descriptions with parameterized action $\alpha(\theta_a)$ and result $\phi(O, \theta_e)$ that are as correct and as weak as possible. We will learn such operator descriptions from training examples of the form: (p, a, e). The p and e are detailed descriptions of the world state before and after the action is executed and a is a parameterized instance of the operator with known parameters.

Learning an operator description requires determining both structure and parameters, including:

Relevant objects: Our domains of interest will have many objects of different types; different training and testing examples will have different numbers and types of objects, and it is necessary to determine which ones are relevant for characterizing the operation. Any object mentioned in the result is relevant and some aspect of the robot is typically relevant. In this paper, we limit ourselves to considering just the robot and those objects named in the result condition, but more generally, there may be other objects such as tools that play a role in the pre-image.

Relevant properties and relations: Given the set of relevant objects, then any properties of or relations among those objects may be relevant to determining the pre-image. Determining a relevant subset of these properties and relations can be modeled as a feature-selection problem, and addressed through any of the wide variety of feature-selection mechanisms in machine learning. In the particular example domain we describe in this paper, the set of available features is very small and there is no need for feature selection, but we expect this to be an important aspect of future work, which might be addressed using techniques of relational learning [26] including those used by Pasula et al. [16].

Constraint on parameters: Learning the constraint in the pre-image can be framed as a classification problem, mapping from a vector of values of the parameters characterizing the pre-image, action, and result to a Boolean value. However, we will want to use the resulting classifier in a non-standard way. To implement the generator, we need to be able to, given bindings of some of the parameters, sample from the set of legal bindings of the other parameters. This is because we may need to consider many different instances of this operator when planning, in case some points in the pre-image of this operator are not achievable, due to obstacles or other constraints present in the planning problem. The problem of learning this constraint will be the focus of the rest of the paper.

C. Learning the constraint from data

Recall that a training example is of the form (p, a, e) and that the skill is parameterized by θ_a and the desired result by θ_e. Assume we have determined a set of properties and relations of the relevant objects, and let θ_p be a vector of the values of those properties and relations in state p. We define the constraint $\chi(\theta_p, \theta_a, \theta_e)$ to have value $+1$ if taking the action defined by θ_a in a state satisfying the pre-image conditions parameterized by θ_p will result in a state that satisfies the result condition parameterized by θ_e.

We might, for example, characterize a *place* parameterized skill with: $\theta_\alpha = \tau$ the trajectory followed by the place skill; an effect $Pose(O, \pi)$ so that $\theta_e = \pi$ is the desired pose of the object to be placed; and the pre-image as

Conf(c), *Grasp*(O, g), so that $\theta_p = (c, g)$ specifies the robot configuration c and grasp g in which it must be holding the object in order for the $place$ operation to work.

One way to formulate the learning problem would be to treat it as a *regression* problem: with the values of the parameters that are bound during backward chaining as input, predict, as output, values of the other parameters. In some circumstances, this strategy could be effective, but it doesn't satisfy our general requirements for several reasons. First, because of the flexible use of operators during planning, it may not always be that the same subset of the parameters are bound in each application of the operator. More importantly, for any given assignment of a subset of the parameters, there may be a large number of possible assignments of the other parameters; if this set of feasible assignments is not convex, then applying standard regression techniques may fail to generate any valid solution [27]. Finally, the planner generally requires multiple feasible solutions, in case some of them result in subgoals that are unachievable for reasons outside the scope of this operation.

Instead, we address a more general but more difficult learning problem, in which we train a classification algorithm to label vectors of input values according to whether or not they satisfy the constraint. Then the ith training example is (x^i, y^i), where x^i is the concatenation of $(\theta_p^i, \theta_a^i, \theta_e^i)$, derived from (p^i, a^i, e^i), and $y^i = \chi(\theta_p^i, \theta_a^i, \theta_e^i)$. Intuitively y^i is $+1$ if the result condition, as parameterized by θ_e^i, holds in state e^i. Thus, we have reduced a set of structured relational training examples to a standard vector-space classification data set. In the $place$ example from above, χ would classify tuples (c, g, τ, π) according to whether executing the place skill with trajectory τ, starting with the robot in configuration c holding an object in grasp g would result in the object being placed at pose π.

Next we have to select a learning method. It must be able to represent set membership given hybrid inputs and be trainable using positive and negative examples. Critically, given bindings for a subset of variables (that is, values for some of the input dimensions) it must enable sampling over bindings of the other variables that would result in an overall vector that would be classified as positive. There are many plausible choices, including decision trees, Gaussian processes, and even equation learning. Based on some initial experimentation, we used a feed-forward neural network in the experiments presented in this paper. However, any other method fitting this specification would be satisfactory, and it is an area for further work to experiment more fully with alternative strategies.

We expect to receive an initial set of labeled training data, which should have a reasonable balance of positive and negative examples. It might be generated via demonstration or through a simulation and sampling process. In addition, we assume access to a simulation or real robot execution system that will allow actions to be attempted and their success or failure at achieving the result condition to be recorded. The learning process then proceeds as follows:

1) Use the initial data to train the classifier, generating an initial hypothesis, h_0; and set $t = 1$.
2) Loop, improving performance:
   Planning: Plan to solve a problem instance that needs the result of the learned operator. Bind parameters in the result condition, θ_e, and seek an instance of this operator that will make the result true. Call the learned generator to produce one or more sample bindings of the remaining parameters of the operator, θ_p, θ_a. Continue planning until the initial state is reached;
   Execution: Carry out the resulting plan in the world (terminate execution and restart planning if a plan step does not result in a state in the pre-image of the next step);
   Learning: For the instance of action a, let y^t be $+1$ if the resulting state satisfied the result condition and -1 otherwise, and let $x^t = (\theta_p, \theta_a, \theta_e)$. Add example (x^t, y^t) to the training data. Re-train the classifier, on the augmented data set, to obtain hypothesis h_t, and increment t.

It may be preferable to "batch" the training, and only retrain the classifier after some number of new samples has been obtained.

The process of gathering training data during the solution of actual problems is critically important, because it generates training examples from an input distribution that reflects actual problem instances encountered in practice and tends to explore parts of the space that important and where the constraint is imperfectly learned.

D. Using learned constraint to generate action instances

Generally, when it is time to apply an operator during backward search, the desired result condition is known and it serves to bind the parameters θ_e; however, there are many instances of the skill θ_a and particular preconditions θ_p that would guarantee the desired resulting condition.

So, given θ_e, we need to be able to sample multiple values of θ_p and θ_a. Using a neural network with $tanh$ activation function to represent the classifier means that we can, in some sense, run the network backwards, to try to find values for the unspecified inputs that maximize the output value. Let $h(\theta_p, \theta_a, \theta_e)$ be the functional form of the feed-forward neural-network hypothesis. One view of our goal is to find

$$\theta_p^*, \theta_a^* = \underset{\theta_p, \theta_a}{\arg\max}\, h(\theta_p, \theta_a, \theta_e) \ . \tag{1}$$

We approach this optimization problem by performing gradient descent, because h is a differentiable function.

The result of the local search depends on initialization. We initialize the search by finding the *positive* training example that is most similar along the dimensions of θ_e,

$$(\theta_p', \theta_a', \theta_e') = \underset{\{x^i | y^i = +1\}}{\arg\min}\ d(\theta_e^i, \theta_e) \ , \tag{2}$$

where d is a distance metric appropriate to θ_e. Then we use (θ_p', θ_a') as the initial values for the gradient descent.

To find additional samples of θ_p, θ_a values that satisfy the constraint, we can run gradient descent initialized from other positive training examples. It may be useful to perform a kind of "non-maximal suppression," in which the optimization criterion for subsequent searches includes a penalty that pushes the solution away from those already found.

In addition, during early phases of the training process, when the hypothesis may not be well trained, especially far away from the training data it has already seen, we may add a penalty to the optimization to keep the solution from moving too far from the overall range of the training data.

E. Integration into robot planning framework

The learning process takes place in a somewhat simplified and idealized version of the domain, which assumes complete observability and abstracts away details of the robot's interaction with the domain. Because we assume the ability to solve inverse kinematics and motion-planning problems, these aspects of the pre-image need not be learned. It suffices to learn pre-conditions on the hand or base placement, what objects are being held in what grasps, etc. The existing planning infrastructure can be used to find feasible values for robot configurations and paths, as needed.

Of course, it is possible that the generated hand or base configurations are kinematically infeasible for the robot, because of joint limits or permanent obstacles. In that case, it will be necessary to return to the generator associated with the learned operator, to sample additional points in the operator's pre-image that may be easier to achieve.

Generally, an additional precondition for any action that requires moving the robot will be that it not collide with any objects other than those it intends to (e.g., to push out of the way). Given many executions of a parameterized skill, it will be possible to characterize an "envelope" of space that must be free of obstacles, relative to some other salient object in the operation. Then, existing manipulation abilities can be brought to bear in order to make that space free (e.g., by picking up objects that are in the way and placing them elsewhere). In our current implementation, this envelope is determined by hand, but in future work we plan to develop strategies for learning it from observing the robot's behavior as it executes the parameterized skill.

In addition to understanding general robot-motion capabilities, we assume that the planning infrastructure has a built-in capacity for managing uncertainty. In our implementation, we use the BHPN framework [15], which explicitly represents the robot's uncertainty about aspects of the domain, and performs motion-planning queries in a "shadow" world that incorporates the uncertainty of each object's pose relative to the robot in a grown shape of the object. Thus, although the basic operator description is learned and originally articulated under the assumption of complete observability, it is ultimately modified to operate in belief space, as described in detail in section IV-D.

IV. Implementation and experiments

We have implemented this framework for integrated learning and planning within BHPN, applied to manipulation problems using a PR2 robot. Built in to BHPN is the ability to perform pick-and-place operations in uncertain domains; it includes UKF-based state estimation for the poses of object and the robot base, task and motion planning in belief space, and a hierarchical execution and replanning architecture.

A. Learning to plan to push objects

To illustrate our approach, we add an operation to the basic pick-and-place capabilities of BHPN. We assume that we are given a very basic pushing skill, which holds the robot's hand perpendicular to the surface of a table and moves the hand linearly in a direction normal to the side face of the hand for a fixed distance. The object may rotate or bounce or slide a small amount, in a way that depends on the texture of the sliding surfaces, making it somewhat difficult to accurately predict the effects of the action.

We will incorporate this new skill into our planning system by learning the description of an operator that has the desired result of pushing an object into a region. We assume it is known what the relevant objects are: an object to be pushed, a robot hand, and a region on a horizontal surface. In the experiments reported below, we assume the object and region always have the same size and shape, but there is nothing in the overall framework that requires that restriction.

The parameters that characterize the operator constraint for the pushing operator include: $\theta_e = \pi_r$, parameters of the result condition, which are the pose of the region; $\theta_a = (H, \delta)$, parameters of the action, which are a hand $H \in \{\text{left}, \text{right}\}$ and a distance δ; and $\theta_p = (\pi_o, \pi_h)$, the parameters of the pre-image, which are the pose π_o of the object to be pushed and the pose π_h of the robot hand. With these parameters, we can construct the following preliminary operator description; it is not yet complete, because it does not address issues of kinematics and uncertainty, but it does illustrate the overall structure of an operator and the role played by the parameters and constraint χ:

$\textsc{Push}((O, H, R), (\pi_o, \pi_h, \pi_r, \delta))$:

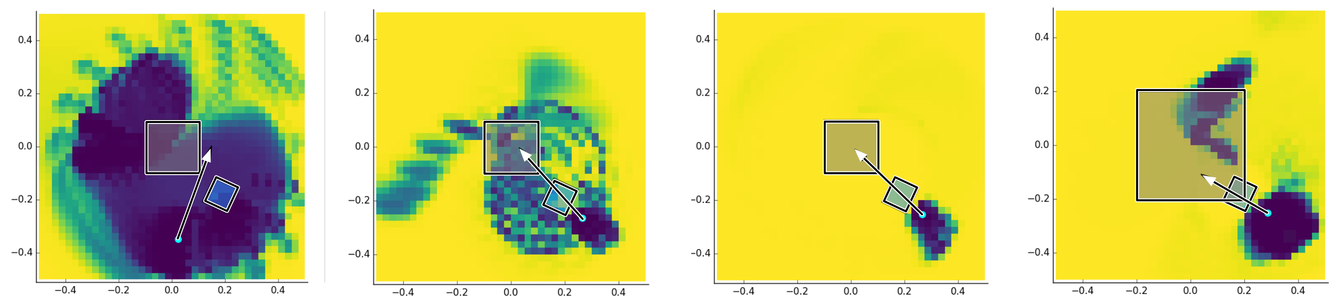
result:	$In(O, R)$
action:	$push(\delta)$
gen:	π_o, π_h, δ
precond:	$Pose(O) = \pi_o$
	$Pose(H) = \pi_h$
	$Pose(R) = \pi_r$
	$\chi(\pi_o, \pi_h, \pi_r, \delta)$

Because regions are geometric constructs, once the region has been determined (by binding variable R), its pose π_r is determined. It is appropriate to think of R as denoting the region object and π_r as denoting its pose; unlike the poses of other objects, the pose of a region cannot be changed.

B. Classifier learning

We must learn the condition χ on the parameters $(\pi_o, \pi_h, \pi_r, \delta)$ that guarantees the correctness of the operator description. In these experiments, all of the training data is gathered using a planar pushing simulation based on a numerical approximation of the minimum power principle [28].

The most straightforward representation of an input consists of three object poses (represented in a fixed global coordinate frame) and a distance $(x_r, y_r, \theta_r, x_o, y_o, \theta_o, x_h, y_h, \theta_h, \delta)$. Although it is probably



(a) After initial training. (b) After 200 active examples. (c) After 2000 active examples. (d) Bigger feasible region.

Fig. 1: Predictions: yellow indicates predicted failure; blue indicates predicted success.

possible to train a network on this “raw” representation, in order to speed up learning and exploit underlying structural knowledge we have of the domain, we applied a fixed feature transformation that makes the relationships among the objects more explicit:

$$(\alpha_{or}, d_{or}, \beta_{or}, \alpha_{hr}, d_{hr}, \beta_{hr}, \alpha_{oh}, d_{oh}, \beta_{oh}, \delta)$$

where α_{ab} is the angle from the center of object a to the center of object b, d_{ab} is the distance from the center of object a to the center of object b, and β_{ab} is the orientation of object b relative to object a; o is the object to be pushed, r is the target region, and h is the robot hand. We apply one final transformation to this vector, converting all of the angle features to their $\sin$ and $\cos$ so that the network does not have to discover the circular structure of angle space, yielding a final feature representation of 16 dimensions.

The initial data set is drawn as follows: without loss of generality, the region is defined to be at pose $(0, 0, 0)$ (recorded as π_r); the object is placed at a random pose on the supporting surface (recorded as π_o); a nominal pose for the hand is computed by putting the center of the hand on the line connecting the center of the object and the center of the region and positioning the hand near, but on the opposite side of the object from the region; a nominal pushing distance is set to be equal to the distance from the front surface of the robot hand to the center of the region, minus the radius of the object; these nominal values are perturbed by substantial noise (recorded as π_h and δ); the resulting action is executed in simulation and it is noted whether the object was fully contained in the region in the resulting state (recorded as the label y). This process results in substantially more negative than positive examples; sampling continues until a data set with balanced class membership of the desired size is obtained.

The next step is to use this data to train a neural network classifier. We use a feed-forward neural network with two hidden layers of 10 units each, with tanh activation functions, implemented using TensorFlow [29]. We train the neural network for 1000 epochs. If the network has not achieved at least 90% training accuracy, we re-start the training up to 5 times and pick the best weights.

C. Generating instances and retraining

Given a working hypothesis for the constraint χ, embodied in the neural network weights, we can use it to generate operator instances. For the pushing operator, the object and target region are specified, so π_r and the identity of O are known. To generate an operator instance, for planning, we need to sample from χ to generate values for π_o, π_h, and δ.

It is important to first see whether it is feasible to push the object starting at its pose in the current world state, in which case π_o is assigned to be the object's current pose. That leaves the hand pose and push distance to be determined; in terms of the optimization problem in equation 1, we have fixed θ_e and the part of θ_p that specifies the object pose. We perform a local search for optimizing values of π_h and δ, initializing the search in a grid of starting points centered around the estimate described in equation 2, and then performing local search using the Nelder-Mead method implemented in Scipy.This process yields values for the robot hand pose and push distance.

Figure 1c illustrates this process. There are four degrees of freedom in our optimization: x_h, y_h, θ_h, and δ. Letting $h(x_r, y_r, \theta_r, x_o, y_o, \theta_o, x_h, y_h, \theta_h, \delta)$ represent the function embodied in the neural-network structure and weights, and given a “query” region and object placement $(x^q_r, y^q_r, \theta^q_r, x^q_o, y^q_o, \theta^q_o)$, the figure shows, for each possible x_h, y_h pair, the classifier's output value for the maximizing values of θ_h and δ; that is,

$$f(x, y) = \max_{\theta,\delta} h(x^q_r, y^q_r, \theta^q_r, x^q_o, y^q_o, \theta^q_o, x, y, \theta, \delta) \ .$$

In this example, the object is at location $(x = 0.18, y = -0.18, \theta = -2.0)$ relative to the region, and we can see that locations of the hand in the dark areas, near $(x = 0.27, y = -0.025, \theta = -2.340]$ have high output values, meaning there are values of the hand orientation and push distance for which they are likely to succeed, but in most other locations, they will not. The cyan point in the figure indicates the initial position of the hand and the arrow indicates the direction and length of the push motion ($\delta = 0.33$). Figure 1d shows the learned pre-image constraint when the target region is larger; we can see that the region of good solutions is also larger, giving the planner more options in case some are infeasible for other reasons. In this figure, there is an area toward the upper right that also seems good, but which is not; the active learning process has not generated training data in that area, so it has not yet been ruled out, but performance is good because it has been reliable generating solutions in the good region in the lower right.

Figures 1a and 1b show earlier states of the classifier, in which the neural network representing h has not yet been trained very well; it classifies regions of the hand-position space as positive that will not result in a successful push. In general, whatever distribution is used to generate initial training data will not accurately or adequately represent the distribution of queries presented in practice; furthermore, the process of performing optimization on the function landscape, in order to fill in query parameters, will ”visit” and require good output values for many parts of the space that might not naturally be sampled.

For this reason, we adopt an active learning strategy. Whenever a query is presented and the optimizing completion is computed, that generated operator instance is tested. In this case, it means that given a region and object position, a hand position and push distance are generated. This action is then actually run (either in simulation, as in the results reported here, or on a real robot) and the true outcome is recorded. This example is saved as a new training data item. Early on, it frequently happens that the neural network predicts that a particular setting of hand pose and distance will succeed, when in fact they do not. Such examples are critical: when added to the training set, they prevent future versions of the classifier from making the same mistake.

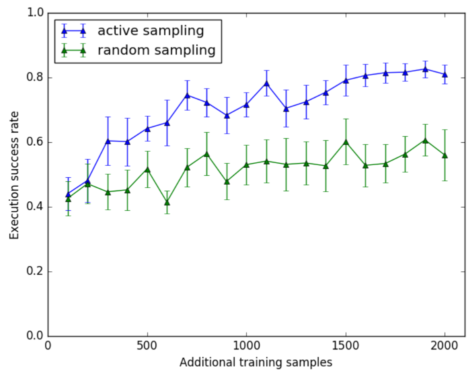


Fig. 2: Learning curves: prediction accuracy vs number of additional examples beyond initial training.

Figure 2 demonstrates the importance of this active learning strategy. It contains two learning curves. They both begin with a classifier trained on 5000 examples drawn from our training distribution. The Y axis shows how successful that classifier was, when used to generate values for hand-pose and distance on new test queries (where the success of a generated value was measured by actually trying it in simulation). The X axis measures the number of additional training examples used in construction of subsequent classifiers. In the random-sampling curve, new training examples are simply drawn from the original training distribution. In the active-sampling curve, new training examples are drawn from the results of the process, described above, of using the classifier to generate new hand-pose and distance parameters, given brand new object and region-pose queries and labeling those completed vectors using the simulator. We can see that the “active” learning process is critical; simply adding more data from the original distribution does not significantly improve performance, but adding the actively gathered data, particularly representing failures of the generated actions, causes the performance to improve significantly.

D. Integration with BHPN

Our goal, in learning an operator description, is to be able to integrate it with a robot task-and-motion planning system that already has a useful substrate of capabilities. In order to do so, we need to augment our understanding of the learned operator with constraints on the robot, with free-space constraints, and with reasoning about uncertainty.

After learning a classifier to represent the constraint χ and training actively in simulation, we extend the operator description to the following form, in which c_i is the initial robot configuration and c_f is the robot's configuration after finishing the push; we assume the robot returns to its initial configuration after the push action is completed.

$\textsc{Push}((O, H, R), (\pi_o, \pi_h, \pi_r r, \delta, c_i, c_f, P, P', \Sigma_o, \Delta_o, \Delta_c))$:

result: $\Pr(In(O, R)) > P$

action: $push(\delta)$

gen: $\pi_o, \pi_h, \delta, c_i, c_f, P', \Sigma_o, \Delta_o, \Delta_c$

precond: $\mu(Pose(O)) \in \pi_o \pm \Delta_o$
$\Sigma(Pose(O)) < \Sigma_o$
$InvKin(H, \pi_h, c_i)$
$InvKin(H, \pi_h + \delta, c_f)$
$Conf \in c_i \pm \Delta_c$
$\Pr(FreeDirectPath(c_i, c_f, [O])) > P'$
$\Pr(Holding(H) = \textbf{None}) > P'$
$Pose(R) = \pi_r$
$\chi(\pi_o, \pi_h, \pi_r, \delta)$

We have added several parameters and added or modified some of the preconditions and effects. The result is now articulated in terms of the robot's belief that the object is in the region, asserting that it is greater than some parameter P. The requirements on the pose of O are articulated in terms of the mean $\mu(Pose(O))$ and variance $\Sigma(Pose(O))$ of the robot's belief about the object's pose; the variance condition is on the distribution of the pose of O relative to the robot's base. Because we cannot command a hand pose directly, we need to reduce that condition to one on the robot configuration. We use existing inverse-kinematics solvers for two robot configurations; c_i is an initial configuration that places hand H in pose π_h and c_f is a configuration for the end of the pushing action in which hand H has moved forward (orthogonal to its original orientation) by distance δ. Then, we require that the initial robot configuration be near c_i (within some fixed tolerance Δ_c). In addition, we require that there be, with high probability, a collision-free direct path from c_i to c_f, with the possible exception of object O, and that hand H be empty, both with probability P' derived from and greater than P. The constraint χ remains the one learned by the neural network.

This operator description is added to the other standard operators in BHPN, for moving the robot, picking and placing objects, and looking at objects to reduce uncertainty.

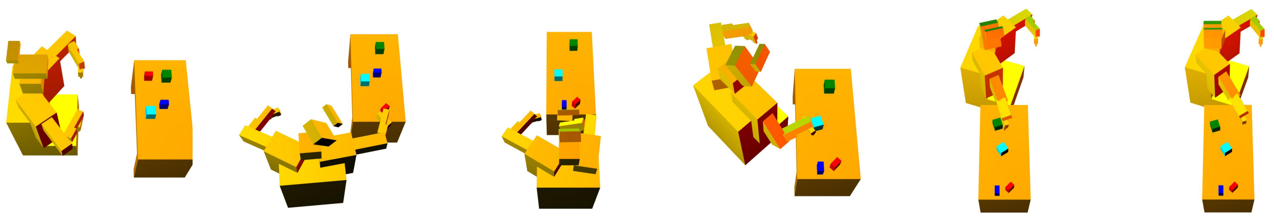


Fig. 3: Simulated execution of two pushing motions (on the cyan and green blocks), each requiring clearing the target region.

When it is used during planning, the variables $O, R,$ and P are bound by matching the result condition to some desired condition in the goal. The region pose π_r is a fixed function of the region R. The generator first (heuristically) tries setting π_o to be the pose of the object in the current belief state and then uses the neural network representation of χ to find feasible values for π_h and δ given π_r and π_o. Then we use standard inverse kinematics routines to generate c_i and c_f. Various parameters governing probabilistic conditions (P', Σ_o, Δ_o, Δ_c) are computed using reasoning about the dynamics of a Bayesian belief-update process as described in our earlier work [15].

Figure 3 shows several frames from a simulated execution of the resulting system achieving the goal of having each of the two large objects inside its own specified region on the table. (A movie of this execution, as well as a number of others, is available in the companion video.) Note that, in order to solve this problem, the ability to plan and execute pushing motions is combined naturally with and significantly leveraged by the robot's ability to:

- Perform inverse kinematics and motion planning in order to move to configurations from which pushing actions can be successfully executed;
- Plan and execute observation actions, and maintain a belief distribution over the poses of the objects relative to one another and to the robot; and
- Reason about the free space necessary for the pushing actions and move obstacles out of the way.

Additionally, but not demonstrated in this particular example, the robot can use either hand to execute the push actions, and can use its ability to push an object into a region to aid in other high-level goals (e.g., pushing the object into an area in which it can be washed or painted.)

The integration of learning and planning is essential for building truly capable robots. The framework presented here is a step toward using existing competences in planning to leverage learning to use new sensorimotor skills.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics*. Springer, 2008.

[2] B. Argall and A. Billard, "A survey of tactile human-robot interactions," *Robotics and Autonomous Systems*, vol. 58, no. 10, 2010.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *I. J. Robotics Res.*, vol. 32, no. 11, 2013.

[4] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, 2013.

[5] O. Kroemer and G. S. Sukhatme, "Learning spatial preconditions of manipulation skills using random forests," in *Humanoids*, 2016.

[6] G. Konidaris, S. Kuindersma, R. A. Grupen, and A. G. Barto, "Robot learning from demonstration by constructing skill trees," *I. J. Robotics Res.*, vol. 31, no. 3, 2012.

[7] G. Neumann, C. Daniel, A. Paraschos, A. G. Kupcsik, and J. Peters, "Learning modular policies for robotics," *Front. Comput. Neurosci.*, vol. 2014, 2014.

[8] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *I. J. Robotics Res.*, vol. 34, no. 2, 2015.

[9] A. Yamaguchi and C. G. Atkeson, "Neural networks and differential dynamic programming for reinforcement learning problems," in *ICRA*, 2016.

[10] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann, "Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration," in *ICRA*, 2005.

[11] H. Veeraraghavan, "Teaching sequential tasks with repetition through demonstration," in *AAMAS*, 2008.

[12] M. Nicolescu and M. Mataric, "Methods for robot task learning: Demonstrations, generalization and practice," in *AAMAS*, 2003.

[13] S. Ekvall and D. Kragic, "Robot learning from demonstration: A task-level planning approach," *Intl. J. on Advanced Robotics Systems*, vol. 5, no. 3, 2008.

[14] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss, "Learning manipulation actions from a few demonstrations," in *ICRA*, 2013.

[15] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *IJRR*, vol. 32, no. 9-10, 2013.

[16] H. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *JAIR*, vol. 29, 2007.

[17] T. Lang and M. Toussaint, "Planning with noisy probabilistic relational rules," *JAIR*, vol. 39, 2010.

[18] N. Jetchev, T. Lang, and M. Toussaint, "Learning grounded relational symbols from continuous data for abstract reasoning," ICRA Workshop on Autonomous Learning, 2013.

[19] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez, "Constructing symbolic representations for high-level planning," in *AAAI*, 2014.

[20] ——, "Symbol acquisition for probabilistic high-level planning," in *IJCAI*, 2015.

[21] T. Zimmerman and S. Kambhampati, "Learning-assisted automated planning," *AI Magazine*, 2003.

[22] K. Mourão, L. S. Zettlemoyer, R. P. A. Petrick, and M. Steedman, "Learning STRIPS operators from noisy and incomplete observations," in *UAI*, 2012.

[23] H. H. Zhuo, H. Muñoz-Avila, and Q. Yang, "Learning hierarchical task network domains from partially observed plan traces," *Artif. Intell.*, vol. 212, 2014.

[24] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: An efficient heuristic for task and motion planning," in *WAFR*, 2014.

[25] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *IJRR*, 2014.

[26] L. Getoor and B. Taskar, Eds., *Introduction to Statistical Relational Learning*. The MIT Press, 2008.

[27] S. Finney, L. P. Kaelbling, and T. Lozano-Pérez, "Predicting partial paths from planning problem parameters," in *RSS*, 2007.

[28] M. A. Peshkin and A. C. Sanderson, "Minimization of energy in quasi-static manipulation," *Trans. Robotics and Automation*, vol. 5, 1989.

[29] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.