Visual Semantic Planning using Deep Successor Representations

Yuke Zhu*[3] Daniel Gordon*[4] Eric Kolve[1] Dieter Fox[4]
Li Fei-Fei[3] Abhinav Gupta[1,2] Roozbeh Mottaghi[1] Ali Farhadi[1,4]
[1]Allen Institute for Artificial Intelligence [2]Carnegie Mellon University
[3]Stanford University [4]University of Washington

Abstract

*A crucial capability of real-world intelligent agents is their ability to **plan** a sequence of actions to achieve their goals in the visual world. In this work, we address the problem of **visual semantic planning**: the task of predicting a sequence of actions from visual observations that transform a dynamic environment from an initial state to a goal state. Doing so entails knowledge about objects and their affordances, as well as actions and their preconditions and effects. We propose learning these through interacting with a visual and dynamic environment. Our proposed solution involves bootstrapping reinforcement learning with imitation learning. To ensure cross task generalization, we develop a deep predictive model based on successor representations. Our experimental results show near optimal results across a wide range of tasks in the challenging THOR environment. The supplementary video can be accessed at the following link: https://goo.gl/vXsbQP.*

1. Introduction

Humans demonstrate levels of visual understanding that go well beyond current formulations of mainstream vision tasks (e.g. object detection, scene recognition, image segmentation). A key element to visual intelligence is the ability to interact with the environment and *plan* a sequence of actions to achieve specific goals; This, in fact, is central to the survival of agents in dynamic environments [2, 37].

Visual semantic planning, the task of interacting with a visual world and predicting a sequence of actions that achieves a desired goal, involves addressing several challenging problems. For example, imagine the simple task of `putting the bowl in the microwave` in the visual dynamic environment depicted in Figure 1. A successful plan involves first finding the bowl, navigating to it, then grabbing it, followed by finding and navigating to the microwave, opening the microwave, and finally putting the bowl in the microwave.

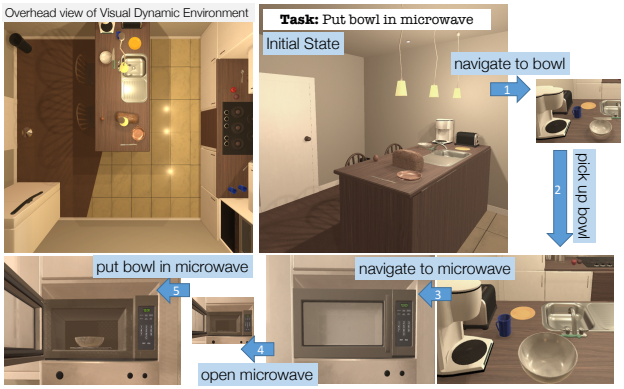

Figure 1. Given a task and an initial configuration of a scene, our agent learns to interact with the scene and predict a sequence of actions to achieve the goal based on visual inputs.

The **first** challenge in visual planning is that performing each of the above actions in a visual dynamic environment requires deep visual understanding of that environment, including the set of possible actions, their preconditions and effects, and object affordances. For example, to *open a microwave* an agent needs to know that it should be in front of the microwave, and it should be aware of the state of the microwave and not try to open an already opened microwave. Long explorations that are required for some tasks imposes the **second** challenge. The variability of visual observations and possible actions makes naïve exploration intractable. To *find a cup*, the agent might need to search several cabinets one by one. The **third** challenge is emitting a sequence of actions such that the agent ends in the goal state and the effects of the preceding actions meet the preconditions of the proceeding ones. Finally, a satisfactory solution to visual planning should enable cross task transfer; previous knowledge about one task should make it easier to learn the next one. This is the **fourth** challenge.

In this paper, we address *visual semantic planning* as a policy learning problem. We mainly focus on high-level actions and do not take into account the low-level details of motor control and motion planning. **Visual Semantic Plan-**

*indicates equal contribution.

ning (VSP) is the task of predicting a sequence of semantic actions that moves an agent from a random initial state in a visual dynamic environment to a given goal state.

To address the first challenge, one needs to find a way to represent the required knowledge of objects, actions, and the visual environment. One possible way is to learn these from still images or videos [12, 51, 52]. But we argue that learning high-level knowledge about actions and their preconditions and effects requires an active and prolonged interaction with the environment. In this paper, we take an interaction-centric approach where we learn this knowledge through interacting with the *visual dynamic environment*. Learning by interaction on real robots has limited scalability due to the complexity and cost of robotics systems [39, 40, 49]. A common treatment is to use simulation as *mental rehearsal* before real-world deployment [4, 21, 26, 53, 54]. For this purpose, we use the THOR framework [54], extending it to enable interactions with objects, where an action is specified as its pre- and post-conditions in a formal language.

To address the second and third challenges, we cast VSP as a policy learning problem, typically tackled by reinforcement learning [11, 16, 22, 30, 35, 46]. To deal with the large action space and delayed rewards, we use imitation learning to bootstrap reinforcement learning and to guide exploration. To address the fourth challenge of cross task generalization [25], we develop a deep predictive model based on successor representations [7, 24] that decouple environment dynamics and task rewards, such that knowledge from trained tasks can be transferred to new tasks with theoretical guarantees [3].

In summary, we address the problem of visual semantic planning and propose an interaction-centric solution. Our proposed model obtains near optimal results across a spectrum of tasks in the challenging THOR environment. Our results also show that our deep successor representation offers crucial transferability properties. Finally, our qualitative results show that our learned representation can encode visual knowledge of objects, actions, and environments.

2. Related Work

Task planning. Task-level planning [10, 13, 20, 47, 48] addresses the problem of finding a high-level plan for performing a task. These methods typically work with high-level formal languages and low-dimensional state spaces. In contrast, visual semantic planning is particularly challenging due to the high dimensionality and partial observability of visual input. In addition, our method facilitates generalization across tasks, while previous methods are typically designed for a specific environment and task.

Perception and interaction. Our work integrates perception and interaction, where an agent actively interfaces with the environment to learn policies that map pixels to actions. The synergy between perception and interaction has drawn an increasing interest in the vision and robotics community. Recent work has enabled faster learning and produced more robust visual representations [1, 32, 39] through interaction. Some early successes have been shown in physical understanding [9, 26, 28, 36] by interacting with an environment.

Deep reinforcement learning. Recent work in reinforcement learning has started to exploit the power of deep neural networks. Deep RL methods have shown success in several domains such as video games [35], board games [46], and continuous control [30]. Model-free RL methods (e.g., [30, 34, 35]) aim at learning to behave solely from actions and their environment feedback; while model-based RL approaches (e.g., [8, 44, 50]) also estimate a environment model. Successor representation (SR), proposed by Dayan [7], can be considered as a hybrid approach of model-based and model-free RL. Barreto *et al*. [3] derived a bound on value functions of an optimal policy when transferring policies using successor representations. Kulkarni *et al*. [24] proposed a method to learn deep successor features. In this work, we propose a new SR architecture with significantly reduced parameters, especially in large action spaces, to facilitate model convergence. We propose to first train the model with imitation learning and fine-tune with RL. It enables us to perform more realistic tasks and offers significant benefits for transfer learning to new tasks.

Learning from demonstrations. Expert demonstrations offer a source of supervision in tasks which must usually be learned with copious random exploration. A line of work interleaves policy execution and learning from expert demonstration that has achieved good practical results [6, 43]. Recent works have employed a series of new techniques for imitation learning, such as generative adversarial networks [19, 29], Monte Carlo tree search [17] and guided policy search [27], which learn end-to-end policies from pixels to actions.

Synthetic data for visual tasks. Computer games and simulated platforms have been used for training perceptual tasks, such as semantic segmentation [18], pedestrian detection [33], pose estimation [38], and urban driving [5, 41, 42, 45]. In robotics, there is a long history of using simulated environments for learning and testing before real-world deployment [23]. Several interactive platforms have been proposed for learning control with visual inputs [4, 21, 26, 53, 54]. Among these, THOR [54] provides high-quality realistic indoor scenes. Our work extends THOR with a new set of actions and the integration of a planner.

Figure 2. Example images that demonstrate the state changes before and after an object interaction from each of the six action types in our framework. Each action changes the visual state and certain actions may enable further interactions such as opening the fridge before taking an object from it.

3. Interactive Framework

To enable interactions with objects and with the environment, we extend the THOR framework [54], which has been used for learning visual navigation tasks. Our extension includes new object states, and a discrete description of the scene in a planning language [13].

3.1. Scenes

In this work, we focus on *kitchen* scenes, as they allow for a variety of tasks with objects from many categories. Our extended THOR framework consists of 10 individual kitchen scenes. Each scene contains an average of 53 distinct objects with which the agent can interact. The scenes are developed using the Unity 3D game engine.

3.2. Objects and Actions

We categorize the objects by their affordances [15], i.e., the plausible set of actions that can be performed. For the tasks of interest, we focus on two types of objects: 1) *items* that are small objects (mug, apple, etc.) which can be picked up, held, and moved by the agent to various locations in the scene, and 2) *receptacles* that are large objects (table, sink, etc.) which are stationary and can hold a fixed capacity of *items*. A subset of *receptacles*, such as fridges and cabinets, are *containers*. These *containers* have doors that can be opened and closed. The agent can only put an *item* in a *container* when it is open. We assume that the agent can hold at most one *item* at any point. We further define the following actions to interact with the objects:

1. Navigate {*receptacle*}: moving from the current location of the agent to a location near the *receptacle*;
2. Open {*container*}: opening the door of a *container* in front of an agent;
3. Close {*container*}: closing the door of a *container* in front of an agent;
4. Pick Up {*item*}: picking up an *item* in field of view;
5. Put {*receptacle*}: putting a held item in the *receptacle*;
6. Look Up and Look Down: tilting the agent's gaze 30 degrees up or down.

In summary, we have six action types, each taking a corresponding type of action arguments. The combination of actions and arguments results in a large action set of 80 per scene on average. Fig. 2 illustrates example scenes and the six types of actions in our framework. Our definition of action space makes two important abstractions to make learning tractable: 1) it abstracts away from navigation, which can be tackled by a subroutine using existing methods such as [54]; and 2) it allows the model to learn with semantic actions, abstracting away from continuous motions, e.g., the physical movement of a robot arm to grasp an object. A common treatment for this abstraction is to "fill in the gaps" between semantic actions with callouts to a continuous motion planner [20, 47]. It is evident that not all actions can be performed in every situation. For example, the agent cannot pick up an *item* when it is out of sight, or put a tomato into fridge when the fridge door is closed. To address these requirements, we specify the pre-conditions and effects of actions. Next we introduce an approach to declaring them as logical rules in a planning language. These rules are only encoded in the environment but not exposed to the agent. Hence, the agent must learn them through interaction.

3.3. Planning Language

The problem of generating a sequence of actions that leads to the goal state has been formally studied in the field of automated planning [14]. Planning languages offer a standard way of expressing an automated planning problem instance, which can be solved by an off-the-shelf planner. We use STRIPS [13] as the planning language to describe our visual planning problem.

In STRIPS, a planning problem is composed of a description of an initial state, a specification of the goal state(s), and a set of actions. In visual planning, the initial state corresponds to the initial configuration of the scene. The specification of the goal state is a boolean function that returns true on states where the task is completed. Each action is defined by its precondition (conditions that must be satisfied before the action is performed) and postcondition (changes caused by the action). The STRIPS formulation enables us to define the rules of the scene, such as object affordances and causality of actions.

4. Our Approach

We first outline the basics of policy learning in Sec. 4.1. Next we formulate the *visual semantic planning* problem as a policy learning problem and describe our model based on successor representation. Later we propose two protocols of training this model using imitation learning (IL) and reinforcement learning (RL). To this end, we use IL to bootstrap

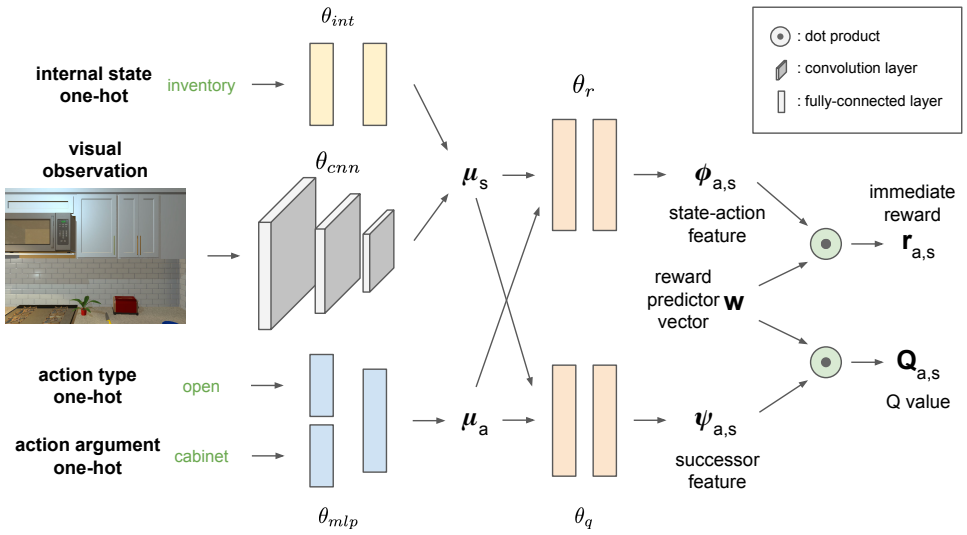

Figure 3. An overview of the network architecture of our successor representation (SR) model. Our network takes in the current state as well as a specific action and predicts an immediate reward $r_{a,s}$ as well as a discounted future reward $Q_{a,s}$, performing this evaluation for each action. The learned policy π takes the argmax over all Q values as its chosen action.

our model and use RL to further improve its performance.

4.1. Successor Representation

We formulate the agent's interactions with an environment as a *Markov Decision Process* (MDP), which can be specified by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions. For $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $p(s'|s,a)$ defines the probability of transiting from the state s to the next state $s' \in \mathcal{S}$ by taking action a. $r(s,a)$ is a real-value function that defines the expected immediate reward of taking action a in state s. For a state-action trajectory, we define the future discounted *return* $R = \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)$, where $\gamma \in [0,1]$ is called the discount factor, which trades off the importance of immediate rewards versus future rewards.

A *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ defines a mapping from states to actions. The goal of policy learning is to find the optimal policy π^* that maximizes the future discounted return R starting from state s_0 and following the policy π^*. Instead of directly optimizing a parameterized policy, we take a value-based approach. We define a state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ under a policy π as

$$Q^\pi(s,a) = \mathbb{E}^\pi[R|s_0 = s, a_0 = a], \tag{1}$$

i.e., the expected episode return starting from state s, taking action a, and following policy π. The Q value of the optimal policy π^* obeys the Bellman equation [49]:

$$Q^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[r(s,a) + \gamma \max_{a'} Q(s',a')] \tag{2}$$

In deep Q networks [35], Q functions are approximated by a neural network $Q(s,a|\theta)$, and can be trained by minimizing the ℓ_2-distance between both sides of the Bellman equation in Eq. (2). Once we learn Q^{π^*}, the optimal action at state s can be selected by $a^* = \arg\max_a Q^{\pi^*}(s,a)$.

Successor representation (SR), proposed by Dayan [7], uses a similar value-based formulation for policy learning. It differs from traditional Q learning by factoring the value function into a dot product of two components: a reward predictor vector $\mathbf{w}$ and a predictive successor feature $\psi(s,a)$. To derive the SR formulation, we start by factoring the immediate rewards such that

$$r(s,a) = \phi(s,a)^T \mathbf{w}, \tag{3}$$

where $\phi(s,a)$ is a *state-action feature*. We expand Eq. (1) using this reward factorization:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i r(s_i,a_i)|s_0 = s, a_0 = a] \\ &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi(s_i,a_i)^T \mathbf{w}|s_0 = s, a_0 = a] \\ &= \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi(s_i,a_i)|s_0 = s, a_0 = a]^T \mathbf{w} \\ &= \psi^\pi(s,a)^T \mathbf{w} \end{aligned} \tag{4}$$

We refer to $\psi(s,a)^\pi = \mathbb{E}^\pi[\sum_{i=0}^{\infty} \gamma^i \phi_{s,a}|s_0 = s, a_0 = a]$ as the *successor features* of the pair (s,a) under policy π.

Intuitively, the successor feature $\psi^\pi(s,a)$ summarizes the environment dynamics under a policy π in a state-action feature space, which can be interpreted as the expected future "feature occupancy". The reward predictor vector $\mathbf{w}$

induces the structure of the reward functions, which can be considered as an embedding of a task. Such decompositions have been shown to offer several advantages, such as being adaptive to changes in distal rewards and apt to option discovery [24]. A theoretical result derived by Barreto *et al.* implies a bound on performance guarantee when the agent transfers a policy from a task t to a similar task t', where task similarity is determined by the ℓ_2-distance of the corresponding $\mathbf{w}$ vectors between these two tasks t and t' [3]. Successor representation thus provides a generic framework for policy transfer in reinforcement learning.

4.2. Our Model

We formulate the problem of *visual semantic planning* as a policy learning problem. Formally, we denote a task by a Boolean function $t : \mathcal{S} \to \{0, 1\}$, where a state s completes the task t iff $t(s) = 1$. The goal is to find an optimal policy π^*, such that given an initial state s_0, π^* generates a state-action trajectory $\mathcal{T} = \{(s_i, a_i) \,|\, i = 0 \dots T\}$ that maximizes the sum of immediate rewards $\sum_{i=0}^{T-1} r(s_i, a_i)$, where $t(s_{0 \dots T-1}) = 0$ and $t(s_T) = 1$.

We parameterize such a policy using the successor representation (SR) model from the previous section. We develop a new neural network architecture to learn ϕ, ψ and $\mathbf{w}$. The network architecture is illustrated in Fig. 3. In THOR, the agent's observations come from a first-person RGB camera. We also pass the agent's internal state as input, expressed by one-hot encodings of the held object in its inventory. The action space is described in Sec. 3.2. We start by computing embedding vectors for the states and the actions. The image is passed through a 3-layer convolutional encoder, and the internal state through a 2-layer MLP, producing a state embedding $\mu_s = f(s; \theta_{cnn}, \theta_{int})$. The action $a = [a_{type}, a_{arg}]$ is encoded as one-hot vectors and passed through a 2-layer MLP encoder that produces an action embedding $\mu_a = g(a_{type}, a_{arg}; \theta_{mlp})$. We fuse the state and action embeddings and generate the state-action feature $\phi_{s,a} = h(\mu_s, \mu_a; \theta_r)$ and the successor feature $\psi_{s,a} = m(\mu_s, \mu_a; \theta_q)$ in two branches. The network predicts the immediate reward $r_{s,a} = \phi_{s,a}^T \mathbf{w}$ and the Q value under the current policy $Q_{s,a} = \psi_{s,a}^T \mathbf{w}$ using the decomposition from Eq. (3) and (4).

4.3. Imitation Learning

Our SR-based policy can be learned in two fashions. First, it can be trained by imitation learning (IL) under the supervision of the trajectories of an optimal planner. Second, it can be learned by trial and error using reinforcement learning (RL). In practice, we find that the large action space in THOR makes RL from scratch intractable due to the challenge of exploration. The best model performance is produced by IL bootstrapping followed by RL fine-tuning.

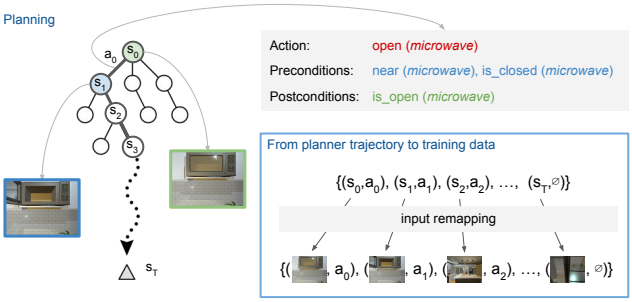

Figure 4. We use a planner to generate a trajectory from an initial state-action pair (s_0, a_0) to a goal state s_T. We describe each scene in a STRIPS-based planning language, where actions are specified by their pre- and post-conditions (see Sec. 3.3). We perform input remapping, illustrated in the blue box, to obtain the image-action pairs from the trajectory as training data. After performing an action, we update the plan and repeat.

Given a task, we generate a state-action trajectory:

$$\mathcal{T} = \{(s_0, a_0), \{(s_1, a_1), \dots, (s_{T-1}, a_{T-1}), (s_T, \emptyset)\} \quad (5)$$

using the planner from the initial state-action pair (s_0, a_0) to the goal state s_T (no action is performed at terminal states). This trajectory is generated on a low-dimensional state representation in the STRIPS planner (Sec. 3.3). Each low-dimensional state corresponds to an RGB image, i.e., the agent's visual observation. During training, we perform *input remapping* to supervise the model with image-action pairs rather than feeding the low-dimensional planner states to the network. To fully explore the state space, we take planner actions as well as random actions off the optimal plan. After each action, we recompute the trajectory. This process of generating training data from a planner is illustrated in Fig. 4. Each state-action pair is associated with a true immediate reward $\hat{r}_{s,a}$. We use the mean squared loss function to minimize the error of reward prediction:

$$\mathcal{L}_r = \frac{1}{T} \sum_{i=0}^{T-1} (\hat{r}_{s,a} - \phi_{s,a}^T \mathbf{w})^2. \quad (6)$$

Following the REINFORCE rule [49], we use the discounted return along the trajectory $\mathcal{T}$ as an unbiased estimate of the true Q value: $\hat{Q}_{s,a} \approx \sum_{i=0}^{T-1} \gamma^i \hat{r}_{s,a}$. We use the mean squared loss to minimize the error of Q prediction:

$$\mathcal{L}_Q = (\hat{Q}_{s,a} - \psi_{s,a}^T \mathbf{w})^2 \quad (7)$$

The final loss on the planner trajectory $\mathcal{T}$ is the sum of the reward loss and the Q loss: $\mathcal{L}_\mathcal{T} = \mathcal{L}_r + \mathcal{L}_Q$. Using this loss signal, we train the whole SR network on a large collection of planner trajectories starting from random initial states.

4.4. Reinforcement Learning

When training our SR model using RL, we can still use the mean squared loss in Eq. (6) to supervise the learning

of reward prediction branch for ϕ and $\mathbf{w}$. However, in absence of expert trajectories, we would need an iterative way to learn the successor features ψ. Rewriting the Bellman equation in Eq. (2) with the SR factorization, we can obtain an equality on ϕ and ψ:

$$\psi^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[\phi(s,a) + \gamma\psi(s',a')] \tag{8}$$

where $a' = \arg\max_a \psi(s',a)^T\mathbf{w}$. Similar to DQN [35], we minimize the ℓ_2-loss between both sides of Eq. (8):

$$L_{SR} = \mathbb{E}^{\pi}[(\phi_{s,a} + \gamma\psi_{s',a'} - \psi^{\pi}_{s,a})^2] \tag{9}$$

We use a similar procedure to Kulkarni *et al*. [24] to train our SR model. The model alternates training between the reward branch and the SR branch. At each iteration, a mini-batch is randomly drawn from a replay buffer of past experiences [35] to perform one SGD update.

4.5. Transfer with Successor Features

A major advantage of successor features is its ability to transfer across tasks by exploiting the structure shared by the tasks. Given a fixed state-action representation ϕ, let M_ϕ be the set of all possible MDPs induced by ϕ and all instantiations of the reward prediction vectors $\mathbf{w}$. Assume that π_i^* is the optimal policy of the i-th task in the set $\{M_i \in M_\phi | i = 1, \dots n\}$. Let M_{n+1} to be a new task. We denote $Q_{n+1}^{\pi_i^*}$ as the value function of executing the optimal policy of the task M_i on the new task M_{n+1}, and $\tilde{Q}_{n+1}^{\pi_i^*}$ as an approximation of $Q_{n+1}^{\pi_i^*}$ by our SR model. Given a bound on the approximations such that

$$|Q_{n+1}^{\pi_i^*}(s,a) - \tilde{Q}_{n+1}^{\pi_i^*}(s,a)| \leq \epsilon \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, i = 1, \dots, n,$$

we define a policy π' for the new task M_{n+1} using $\tilde{Q}_{1,\dots,n}$, where $\pi'(s) = \arg\max_a \max_i \tilde{Q}_{n+1}^{\pi_i^*}(s,a)$. Theorem 2 in Barreto *et al*. [3] implies a bound of the gap between value functions of the optimal policy π_{n+1}^* and the policy π':

$$Q_{n+1}^{\pi_{n+1}^*}(s,a) - Q_{n+1}^{\pi'}(s,a) \leq \frac{2\phi_m}{1-\gamma}(\min_i ||\mathbf{w}_i - \mathbf{w}_{n+1}|| + \epsilon),$$

where $\phi_m = \max_{s,a} ||\phi(s,a)||$. This result serves the theoretical foundation of policy transfer in our SR model. In practice, when transferring to a new task while the scene dynamics remain the same, we freeze all model parameters except the single vector $\mathbf{w}$. This way, the policy of the new task can be learned with substantially higher sample efficiency than training a new network from scratch.

4.6. Implementation Details

We feed a history of the past four observations, converted to grayscale, to account for the agent's motions. We use a time cost of -0.01 to encourage shorter plans and a task completion reward of 10.0. We train our model with imitation learning for 500k iterations with a batch size of 32, and a learning rate of 1e-4. We also include the successor loss in Eq. (9) during imitation learning, which helps learn better successor features. We subsequently fine-tune the network with reinforcement learning with 10,000 episodes.

5. Experiments

We evaluate our model using the extended THOR framework on a variety of household tasks. We compare our method against standard reinforcement learning techniques as well as with non-successor based deep models. The tasks compare the different methods' abilities to learn across varying time horizons. We also demonstrate the SR network's ability to efficiently adapt to new tasks. Finally, we show that our model can learn a notion of object affordance by interacting with the scene.

5.1. Quantitative Evaluation

We examine the effectiveness of our model and baseline methods on a set of tasks that require three levels of planning complexity in terms of optimal plan length.

Experiment Setup We explore the two training protocols introduced in Sec. 4 to train our SR model:

1. **RL**: we train the model solely based on trial and error, and learn the model parameters with RL update rules.
2. **IL**: we use the planner to generate optimal trajectories starting from a large collection of random initial state-action pairs. We use the imitation learning methods to train the networks using supervised losses.

Empirically, we find that training with reinforcement learning from scratch cannot handle the large action space. Thus, we report the performance of our SR model trained with imitation learning (SR IL) as well as with additional reinforcement learning fine-tuning (SR IL + RL).

We compare our SR model with the state-of-the-art deep RL model, A3C [34], which is an advantage-based actor-critic method that allows the agent to learn from multiple copies of simulation while updating a single model in an asynchronous fashion. A3C establishes a strong baseline for reinforcement learning. We further use the same architecture to obtain two imitation learning (behavior cloning) baselines. We use the same A3C network structure to train a softmax classifier that predicts the planner actions given an input. The network predicts both the action types (e.g., Put) and the action arguments (e.g., *apple*). We call this baseline CLS-MLP. We also investigate the role of memory in these models. To do this, we add an extra LSTM layer to the network before action outputs, called CLS-LSTM. We also include simple agents that take random actions and take random valid actions at each time step.

	Easy		Medium		Hard	
	Success Rate	**Mean (σ) Episode Length**	**Success Rate**	**Mean (σ) Episode Length**	**Success Rate**	**Mean (σ) Episode Length**
Random Action	1.00	696.33 (744.71)	0.00	-	0.04	2827.08 (927.84)
Random Valid Action	1.00	64.03 (68.04)	0.02	3897.50 (548.50)	0.36	2194.83 (1401.72)
A3C [34]	0.96	101.12 (151.04)	0.00	-	0.04	2674.29 (4370.40)
CLS-MLP	1.00	**2.42** (0.70)	0.65	256.32 (700.78)	0.65	475.86 (806.42)
CLS-LSTM	1.00	2.86 (0.37)	**0.80**	314.05 (606.25)	**0.66**	136.94 (523.60)
SR IL (ours)	1.00	2.70 (1.06)	**0.80**	32.32 (29.22)	0.65	**34.25** (63.81)
SR IL + RL (ours)	1.00	2.57 (1.04)	**0.80**	**26.56** (3.85)	-	-
Optimal planner	1.00	2.36 (1.04)	1.00	12.10 (6.16)	1.00	14.13 (9.09)

Table 1. Results of evaluating the model on the easy, medium, and hard tasks. For each task, we evaluate how many out of the 100 episodes were completed (success rate) and the mean and standard deviation for successful episode lengths. The numbers in parentheses show the standard deviations. We do not fine-tune our SR IL model for the hard task.

Levels of task difficulty We evaluate all of the models with three levels of task difficulty based on the length of the optimal plans and the source of randomization:

1. **Level 1 (Easy)**: Navigate to a *container* and toggle its state. A sample task would be `go to the microwave and open it if it is closed, close it otherwise.` The initial location of the agent and all *container* states are randomized. This task requires identifying object states and reasoning about action preconditions.
2. **Level 2 (Medium)**: Navigate to multiple *receptacles*, collect *items*, and deposit them in a *receptacle*. A sample task here is `pick up three mugs from three cabinets and put them in the sink.` Here we randomize the agent's initial location, while the item locations are fixed. This task requires a long trajectory of correct actions to complete the goal.
3. **Level 3 (Hard)**: Search for an *item* and put it in a *receptacle*. An example task is `find the apple and put it in the fridge.` We randomize the agent's location as well as the location of all items. This task is especially difficult as it requires longer-term memory to account for partial observability, such as which cabinets have previously been checked.

We evaluate all of the models on 10 easy tasks, 8 medium tasks, and 7 hard tasks, each across 100 episodes. Each episode terminates when a goal state is reached. We consider an episode fails if it does not reach any goal state within 5,000 actions. We report the episode success rate and mean episode length as the performance metrics. We exclude these failed episodes in the mean episode length metric. For the easy and medium tasks, we train the imitation learning models to mimic the optimal plans. However for the hard tasks, imitating the optimal plan is infeasible, as the location of the object is uncertain. In this case, the target object is likely to hide in a cabinet or a fridge which the agent cannot see. Therefore, we train the models to imitate a plan which searches for the object from all the *receptacles* in a fixed order. For the same reason, we do not perform RL fine-tuning for the hard tasks.

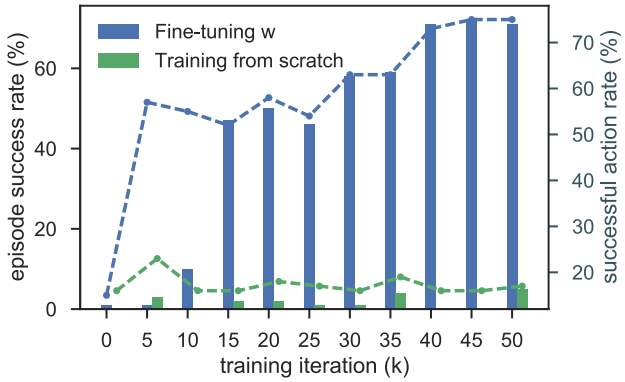

Figure 5. We compare updating $\mathbf{w}$ with retraining the whole network for new hard tasks in the same scene. By using successor features, we can quickly learn an accurate policy for the new item. Bar charts correspond to the episode success rates, and line plots correspond to successful action rate.

Table 1 summarizes the results of these experiments. Pure RL-based methods struggle with the medium and hard tasks because the action space is so large that naïve exploration rarely, if ever, succeeds. Comparing CLS-MLP and CLS-LSTM, adding memory to the agent helps improving success rate on medium tasks as well as completing tasks with shorter trajectories in hard tasks. Overall, the SR methods outperform the baselines across all three task difficulties. Fine-tuning the SR IL model with reinforcement learning further reduces the number of steps towards the goal. More qualitative results can be found in the video.[1]

5.2. Task Transfer

One major benefit of the successor representation decomposition is its ability to transfer to new tasks while only retraining the reward prediction vector $\mathbf{w}$, while freezing the successor features. We examine the sample efficiency of adapting a trained SR model on multiple novel tasks in the same scene. We examine policy transfer in the hard tasks, as the scene dynamics of the searching policy retains, even when the objects to be searched vary. We evaluate the speed at which the SR model converges on a new task by fine-

[1]Link to supplementary video: https://goo.gl/vXsbQP

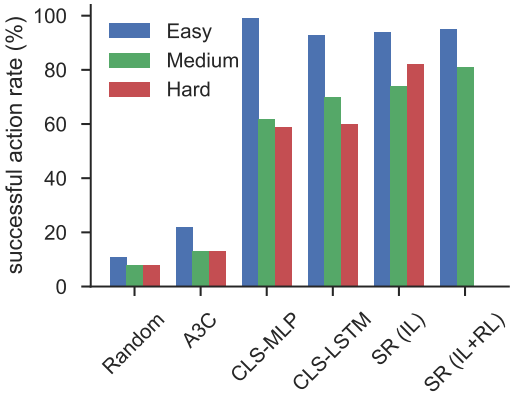

Figure 6. We compare the different models' likelihood of performing a successful action during execution. A3C suffers from the large action space due to naïve exploration. Imitation learning models are capable of differentiating between successful and unsuccessful actions because the supervised loss discourages the selection of unsuccessful actions.

tuning the $\mathbf{w}$ vector versus training the model from scratch. We take a policy for searching a bowl in the scene and substituting four new items (lettuce, egg, container, and apple) in each new task. Fig. 5 shows the episode success rates (bar chart) and the successful action rate (line plot). By fine-tuning $\mathbf{w}$, the model quickly adapts to new tasks, yielding both high episode success rate and successful action rate. In contrast, the model trained from scratch takes substantially longer to converge. We also experiment with fine-tuning the entire model, and it suffers from similar slow convergence.

5.3. Learning Affordances

An agent in an interactive environment needs to be able to reason about the causal effects of actions. We expect our SR model to learn the pre- and post-conditions of actions through interaction, such that it develops a notion of affordance [15], i.e., which actions can be performed under a circumstance. In the real world, such knowledge could help prevent damages to the agent and the environment caused by unexpected or invalid actions.

We first evaluate each network's ability to *implicitly* learn affordances when trained on the tasks in Sec. 5.1. In these tasks, we penalize unnecessary actions with a small time penalty, but we do not explicitly tell the network which actions succeed and which fail. Fig. 6 illustrates that a standard reinforcement learning method cannot filter out unnecessary actions especially given delayed rewards. Imitation learning methods produce significantly fewer failed actions because they can directly evaluate whether each action gets them closer to the goal state.

We also analyze the successor network's capability of *explicitly* learning affordances. We train our SR model with reinforcement learning, by executing a completely random

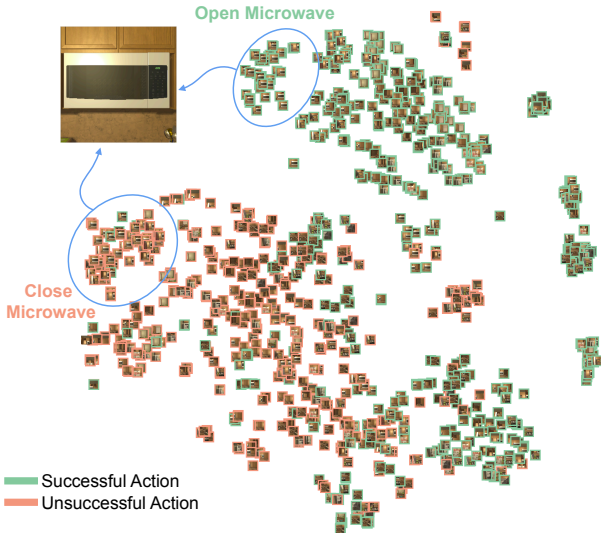

Figure 7. Visualization of a t-SNE [31] embedding of the state-action vector $\phi_{s,a}$ for a random set of state-action pairs. Successful state-action pairs are shown in green, and unsuccessful pairs in orange. The two blue circles highlight portions of the embedding with very similar images but different actions. The network can differentiate successful pairs from unsuccessful ones.

policy in the scene. We define the immediate reward of issuing a successful action as $+1.0$ and an unsuccessful one as -1.0. The agent learns in 10,000 episodes. Fig. 7 shows a t-SNE [31] visualization of the state-action features $\phi_{s,a}$. We see that the network learns to cluster successful state action pairs (shown in green) separate from unsuccessful pairs (orange). The network achieves an ROC-AUC of 0.91 on predicting immediate rewards over random state-action actions, indicating that the model can differentiate successful and unsuccessful actions by performing actions and learning from their outcomes.

6. Conclusions

In this paper, we argue that visual semantic planning is an important next task in computer vision. Our proposed solution shows promising results in predicting a sequence of actions that change the current state of the visual world to a desired goal state. We have examined several different tasks with varying degrees of difficulty and show that our proposed model based on deep successor representations achieves near optimal results in the challenging THOR environment. We also show promising cross-task knowledge transfer results, a crucial component of any generalizable solution. Our qualitative results show that our learned successor features encode knowledge of object affordances, and action pre-conditions and post-effects. Our next steps involve exploring knowledge transfer from THOR to real-world environments as well as examining the possibilities of more complicated tasks with a richer set of actions.

Acknowledgements: This work is in part supported by ONR N00014-13-1-0720, ONR MURI N00014-16-1-2007, NSF IIS-1338054, NSF-1652052, NRI-1637479, NSF IIS-1652052, a Siemens grant, the Intel Science and Technology Center for Pervasive Computing (ISTC-PC), Allen Distinguished Investigator Award, and the Allen Institute for Artificial Intelligence.

References

[1] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv*, 2016. 2

[2] M. L. Anderson. Embodied cognition: A field guide. *Artificial intelligence*, 2003. 1

[3] A. Barreto, R. Munos, T. Schaul, and D. Silver. Successor features for transfer in reinforcement learning. *arXiv*, 2016. 2, 5, 6

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 2013. 2

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, pages 2722–2730, 2015. 2

[6] H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009. 2

[7] P. Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993. 2, 4

[8] M. P. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011. 2

[9] M. Denil, P. Agrawal, T. D. Kulkarni, T. Erez, P. Battaglia, and N. de Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv*, 2016. 2

[10] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *SSRR*, 2009. 2

[11] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. In *ICLR*, 2017. 2

[12] A. Fathi and J. M. Rehg. Modeling actions through state changes. In *CVPR*, 2013. 2

[13] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 1971. 2, 3

[14] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. 2004. 3

[15] J. J. Gibson. *The ecological approach to visual perception: classic edition*. 2014. 3, 8

[16] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*, 2017. 2

[17] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *NIPS*, 2014. 2

[18] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. Understanding real world indoor scenes with synthetic data. In *CVPR*, 2016. 2

[19] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NIPS*, 2016. 2

[20] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011. 2, 3

[21] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jakowski. ViZDoom: A doom-based AI research platform for visual reinforcement learning. In *CIG*, 2016. 2

[22] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via reinforcement learning. In *NIPS*, 2004. 2

[23] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *IJRR*, 2013. 2

[24] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. *arXiv*, 2016. 2, 5, 6, 11

[25] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *arXiv*, 2016. 2

[26] A. Lerer, S. Gross, and R. Fergus. Learning physical intuition of block towers by example. In *ICML*, 2016. 2

[27] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016. 2

[28] W. Li, S. Azimi, A. Leonardis, and M. Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv*, 2016. 2

[29] Y. Li, J. Song, and S. Ermon. Inferring the latent structure of human decision-making from raw visual inputs. *arXiv preprint arXiv:1703.08840*, 2017. 2

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016. 2

[31] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 8

[32] M. Malmir, K. Sikka, D. Forster, J. R. Movellan, and G. Cottrell. Deep q-learning for active recognition of germs: Baseline performance on a standardized dataset for active learning. In *BMVC*, 2015. 2

[33] J. Marin, D. Vázquez, D. Gerónimo, and A. M. López. Learning appearance in virtual scenarios for pedestrian detection. In *CVPR*, 2010. 2

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 2, 6, 7

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 2, 4, 6, 11

[36] R. Mottaghi, M. Rastegari, A. Gupta, and A. Farhadi. "what happens if..." learning to predict the effect of forces in images. In *ECCV*, 2016. 2

[37] A. Noë and J. K. ORegan. On the brain-basis of visual consciousness: A sensorimotor account. *Vision and mind: Selected readings in the philosophy of perception*, 2002. 1

[38] J. Papon and M. Schoeler. Semantic pose using deep networks trained on synthetic rgb-d. In *ICCV*, 2015. 2

[39] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *ECCV*, 2016. 2

[40] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*, 2016. 2

[41] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016. 2

[42] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016. 2

[43] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011. 2

[44] J. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *IJCNN*, 1990. 2

[45] A. Shafaei, J. J. Little, and M. Schmidt. Play and learn: using video games to train computer vision models. *arXiv*, 2016. 2

[46] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016. 2

[47] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, 2014. 2, 3

[48] S. Srivastava, L. Riano, S. Russell, and P. Abbeel. Using classical planners for tasks with continuous operators in robotics. In *ICAPS*, 2013. 2

[49] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. 1998. 2, 4, 5

[50] A. Tamar, S. Levine, and P. Abbeel. Value iteration networks. In *NIPS*, 2016. 2

[51] S. D. Tran and L. S. Davis. Event modeling and recognition using markov logic networks. In *ECCV*, 2008. 2

[52] X. Wang, A. Farhadi, and A. Gupta. Actions ˜ transformations. In *CVPR*, 2016. 2

[53] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, and C. Guionneau. Torcs: The open racing car simulator. 2015. 2

[54] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *ICRA*, 2017. 2, 3, 11

A. Experiment Details

A.1. Experiment Setup

We used the Adam optimizer from (Kingma and Ba) for learning our Successor Representation (SR) model with a learning rate of 1e-4 and a mini-batch size of 32. For the reinforcement learning experiments, we use the discounted factor $\gamma = 0.99$ and a replay buffer size of 100,000. The exploration term ϵ is annealed from 1.0 to 0.1 during the training process. We run an ϵ-greedy policy ($\epsilon = 0.1$) during evaluation. We use soft target updates ($\tau = 0.1$) after every episode. For the easy and medium tasks, we assign $+10.0$ immediate reward for task completion, -5.0 for invalid actions, and -1.0 for other actions (to encourage a shorter plan). For the hard task, we train our SR model to imitate a plan that searches all the *receptacles* for an object in a fixed order of visitation based on the spatial locations of the *receptacles*. We assign $+1.0$ immediate reward for task completion, and an episode terminates as failure if the agent does not follow the order of visitation in the plan.

A.2. Network Inputs

The input to the SR model consists of three components: action (action type and argument), agent's observation (image), and agent's internal state. The action type is encoded by a 7-dimensional one-hot vector, indicating one of the seven action types (Navigate, Open, Close, Pick Up, Put, Look Up, and Look Down). The action argument is encoded by a one-hot vector that has the same dimension as the number of interactable objects plus one. The first dimension denotes null argument used for Look Up and Look Down actions, and the other dimensions correspond to the index of each object. RGB images from the agent's first-person camera are preprocessed to 84×84 grayscale images. We stack four history frames to make an $84 \times 84 \times 4$ tensor as the image input to the convolutional networks. The agent's internal state is expressed by the agent's inventory, rotation, and viewpoint. The inventory is a one-hot vector that represents the index of the held *item*, with an extra dimension for null. The rotation is a 4-dimensional one-hot vector that represents the rotation of the agent (90 degree turns). The viewpoint is a 3-dimensional one-hot vector that represents the tiling angle of the agent's camera ($-30°$, $0°$, and $30°$).

A.3. Network Architecture

Here we describe the network architecture of our proposed SR model. The convolutional image encoder θ_{cnn} takes an $84\times84\times4$ image as input. The three convolutional layers are 32 filters of size 8×8 with stride 4, 64 filters of size 4×4 with stride 2, 64 filters of size 3×3 with stride 1. Finally a fully-connected layer maps the outputs from the convolutional encoder into a 512-d feature. The actions encoder θ_{mlp} and internal state encoder θ_{int} are both 2-layer MLPs with 512 hidden units. A concatenated vector of action, internal state, and image encodings is fed into two 2-layer MLPs θ_r and θ_q with 512 hidden units to produce the 512-d state-action feature $\phi_{s,a}$ and the successor feature $\psi_{s,a}$. We take the dot product of the 512-d reward predictor vector $\mathbf{w}$ and state-action features (successor features) to compute the immediate rewards (Q values). All the hidden layers use ReLU non-linearities. The final dot product layers of the immediate reward and the Q value produce raw values without any non-linearity.

B. Algorithm Details

We describe the reinforcement learning procedure of the SR model in Algorithm 1. This training method follows closely with previous work on deep Q-learning [35] and deep SR model [24]. Similar to these two works, replay buffer and target network are used to stabilize training.

C. Action Space

The set of plausible actions in a scene is determined by the variety of objects in the scene. On average each scene has 53 objects (a subset of them are interactable) and the agent is able to perform 80 actions. Here we provide an example scene to illustrate the interactable objects and the action space.

Scene #9: 16 *items*, 23 *receptacles* (at 11 unique locations), and 15 *containers* (a subset of *receptacles*)

Figure 8. Screenshot of Scene #9

items: *apple*, *bowl*, *bread*, *butter knife*, *glass bottle*, *egg*, *fork*, *knife*, *lettuce*, *mug 1-3*, *plate*, *potato*, *spoon*, *tomato*

receptacles: *cabinet 1-13*, *coffee machine*, *fridge*, *garbage can*, *microwave*, *sink*, *stove burner 1-4*, *table top*

containers: *cabinet 1-13*, *fridge*, *microwave*

actions: 80 actions in total, including 11 Navigation actions, 15 Open actions, 15 Close actions, 14 Pick Up actions, 23 Put actions, Look Up and Look Down.

We have fewer Navigation and Pick Up actions than the number of receptacles and items respectively, as we merge some adjacent receptacles to one location (navigation destination). We also merge picking up items from the same object category into one action. This reduces the size of the action space and speeds up learning. An important simplification that we made is to treat the Navigation actions as "teleports", which abstracts away from visual navigation of the agent. The actual visual navigation problem can be solved as an independent subroutine from previous work [54]. As discussed in Sec. 3.2, not all actions in the set can be issued given a certain circumstance based on affordance. We use the PDDL language to check if the preconditions of an action are satisfied before the action is sent to THOR for execution.

D. Tasks

We list all the tasks that we have evaluated in the experiments in Table 2. In summary, we evaluated tasks from three levels of difficulty, with 10 easy tasks, 8 medium tasks, and 7 hard tasks.

References

[1] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Algorithm 1 Reinforcement Learning for Successor Representation Model

procedure RL-TRAINING
  Initialize replay buffer $\mathcal{D}$ to size N
  Initialize an SR network θ with random weights $\theta = [\theta_{int}, \theta_{cnn}, \theta_{mlp}, \theta_r, \theta_q, \mathbf{w}]$
  Make a clone of θ as the target network $\tilde{\theta}$
  for $i = 1$: *#episodes* **do**:
    Initialize an environment with random configuration
    Reset exploration term $\epsilon = 1.0$
    while not terminal **do**
      Get agent's observation and internal state s_t from the environment
      Compute $Q_{s_t,a} = f(s_t, a; \theta)$ for every action a in action space
      With probability ϵ select a random action a_t; otherwise, select $a_t = \arg\max_a Q_{s_t,a}$
      Execute action a_t to obtain the immediate reward r_t and the next state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
      Sample a random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
      Compute $\tilde{r}_j$, ϕ_{s_j,a_j}, and ψ_{s_j,a_j} using θ for every transition j
      Compute gradients that minimize the mean squared error between r_j and $\tilde{r}_j$
      Compute $\phi_{s_{j+1},a}$, $\psi_{s_{j+1},a}$, and $\tilde{Q}_{s_{j+1},a}$ using $\tilde{\theta}$ for every transition j and every action a
      if s_{j+1} is a terminal state **then**:
        Compute gradients that minimize the mean squared error between ψ_{s_j,a_j} and ϕ_{s_j,a_j}
      else:
        Compute gradients that minimize the mean squared error between ψ_{s_j,a_j} and $\phi_{s_j,a_j} + \gamma\psi_{s_{j+1},a'}$
        where $a' = \arg\max_a \tilde{Q}_{s_{j+1},a}$
      end if
      Perform a gradient descend step to update θ
    end while
    Anneal exploration term ϵ
    Soft-update target network $\tilde{\theta}$ using θ
  end for
end procedure

Table 2. List of Tasks from Three Levels of Difficulty

Scene	Easy	Medium	Hard
1	open / close *fridge*	put *lettuce, tomato* and *glass bottle* to the *sink*	find *bowl* and put in *sink*
2	open / close *cabinet*	put *apple, egg* and *glass bottle* to the *table top*	find *plate* and put in *cabinet*
3	open / close *microwave*	put *glass bottle, lettuce* and *apple* to the *table top*	find *lettuce* and put in *fridge*
4	open / close *cabinet*	put three *mug*s to the *fridge*	find *glass bottle* and put in *microwave*
5	open / close *fridge*	-	-
6	open / close *fridge*	-	-
7	open / close *cabinet*	put three *mug*s to the *table top*	-
8	open / close *fridge*	put *potato, tomato* and *apple* to the *sink*	find *lettuce* and put on *table top*
9	open / close *microwave*	put three *mug*s to the *table top*	find *glass bottle* and put in *fridge*
10	open / close *cabinet*	put *glass bottle, bread* and *lettuce* to *fridge*	find *bowl* and put in *sink*