Model Learning for Look-Ahead Exploration in Continuous Control

Arpit Agarwal, Katharina Muelling, Katerina Fragkiadaki
Carnegie Mellon University
United States
{arpita1,katharam}@andrew.cmu.edu, katef@cs.cmu.edu

Abstract

We propose an exploration method that incorporates look-ahead search over basic learnt skills and their dynamics, and use it for reinforcement learning (RL) of manipulation policies . Our skills are multi-goal policies learned in isolation in simpler environments using existing multigoal RL formulations, analogous to options or macroactions. Coarse skill dynamics, i.e., the state transition caused by a (complete) skill execution, are learnt and are unrolled forward during lookahead search. Policy search benefits from temporal abstraction during exploration, though itself operates over low-level primitive actions, and thus the resulting policies does not suffer from suboptimality and inflexibility caused by coarse skill chaining. We show that the proposed exploration strategy results in effective learning of complex manipulation policies faster than current state-of-the-art RL methods, and converges to better policies than methods that use options or parametrized skills as building blocks of the policy itself, as opposed to guiding exploration. We show that the proposed exploration strategy results in effective learning of complex manipulation policies faster than current state-of-the-art RL methods, and converges to better policies than methods that use options or parameterized skills as building blocks of the policy itself, as opposed to guiding exploration.

Introduction

In animals, skill composition greatly increases the efficiency to solve new problems (White 1959). The skills act as the "building blocks" out of which an agent can form solutions to new problem configurations as well as entirely new problems (Chentanez, Barto, and Singh 2005). Instead of creating a solution from low-level motor commands for each new challenge, skill composition enables the agent to focus on combining and adjusting higher-level skills to achieve the goal. This principle of hierarchical skill composition has been applied by researchers in Reinforcement Learning (RL) in the hope to achieve a similar efficiency for artificial agents (Sutton, Precup, and Singh 1999; Kulkarni et al. 2015). In this context, skills are often referred to as options or macro-actions (Sutton, Precup, and Singh 1999). Options realize the idea of temporally extended actions that can independently accomplish a sub-goal for a defined set of scenarios. The higher-level policy is then tasked

with obtaining the optimal sequence of options to accomplish the task. The performance of the policy therefore critically depends on the set of options available to the agent. If the option set is poorly chosen, the resulting composed policies will be suboptimal (Dietterich 1999). Many researchers have tried to find a "golden" set of options (Chentanez, Barto, and Singh 2005) to compose hierarchical policies from (Dietterich 1999). However, with a growing number of options the efficiency of learning will suffer. This leads to a **trade-off between flexibility and learning speed**: the fewer options, the faster the learning, the less optimal the resulting composed policy. In light of the above, this paper proposes temporally abstracted look-ahead search for exploration, yet fine-grained action composition for policy representations in the manipulation domain. In other words, a small set of generalized basic manipulation policies, which we call skills or options, are learnt and their coarse transition functions are used to unroll forward a tree search during the exploration loop. Yet, policy search still operates over low-level primitive actions, and thus the resulting policies are not limited to coarse skill compositions, as previous hierarchical reinforcement learning formulations (Sutton, Precup, and Singh 1999). This design choice accelerates learning while at the same time permits flexibility in option (skill) selection: as long as the set of skills and the states they visit sufficiently covers the state space necessary for complex manipulation, skills can be redundant, overlapping, or varying in duration, without loss in performance of the final policy.

In a nutshell, our framework works as follows. We train a set of simple manipulation skills, such as grasp, transfer, and reach. Each of these skills represents a (generalized) policy that can handle a set of related goals and is parametrized by both the current state and the desired goal (see Figure 1). Each skill may involve a different number of objects, thus different skills may have different state representations. For each skill we learn a coarse-grain transition function, that, given a pair of initial and goal states, it predicts the resulting state, after execution of the skill.

During training of a new manipulation task represented by a goal configuration, we perform look-ahead tree search using the coarse-grain learned skill dynamics: at each time step of an episode, we unfold a search tree by sampling *skill-goal* combinations and using the learned neural skill dynamics to predict resulting states (Figure 1). We choose the tree
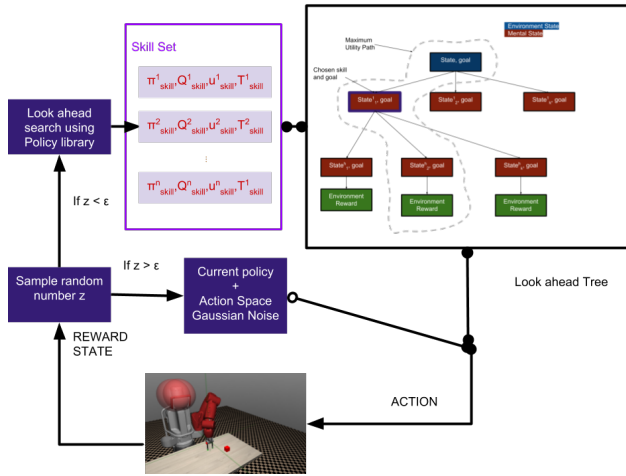
Figure 1: **Overview**. We use learned skill dynamics with deep neural regressors and use them for look-ahead tree search, to guide effective exploration in reinforcement learning of complex manipulation tasks.

path that leads closer to the desired goal configuration, and the first skill-goal of the path is executed in the real environment. We execute the chosen skill with its goal parameter until termination of that skill or episode. Then, planning is repeated from the newly reached state, akin to model predictive control (Mayne 2014).

Our skill-based look-ahead exploration outperforms $epsilon-$greedy exploration, model-based RL (Sutton 1991) where the fitted dynamics model is used to supply (fake) experience tuples and not for exploration, as well as policy learning over coarse parameterized skills (Hausknecht and Stone 2016), as opposed to low-level action primitives.

In summary, success of our framework depends on the following design choices, that distinguish it from previous works:

- Look-ahead with coarse-grain skill dynamics yet act with fine-grain primitives. Coarse-grain transition dynamics do not suffer from severe model error accumulation when unrolled in time (Oh et al. 2015), as few hops are sufficient to look far into the future. Yet, policies over fine-grained actions — as opposed to pretrained skills — produce smooth behaviour, being as fast and optimal as possible.

- Purposeful skill dynamics. Our dynamics model used for look-ahead is built from basic manipulation skills, as opposed to random exploration and transitions therein (Ha and Schmidhuber 2018). This ensures that learnt dynamics cover important part of the state space. In contrast, dynamics learnt from random exploration alone often miss useful states, and thus their temporal unrolling is less informative.

Our code is available at https://github.com/arpit15/skill-based-exploration-drl

Related work

Exploration - Intrinsic motivation   Effective exploration is a central challenge in learning good control policies (Mohamed and Rezende 2015). Methods such as $\epsilon-$greedy, that either follow the current found policy or sample a random action with a certain probability, are useful for local exploration but fail to provide impetus for the agent to explore different areas of the state space. Exploring by maximizing the agent's curiosity as measured by the error of predictive dynamics (Pathak et al. 2017) expected improvement of predictive dynamics (Schmidhuber 1990), information maximization (Mohamed and Rezende 2015), state visitation density (Bellemare et al. 2016), uncertainty of the value function estimates (Osband et al. 2016), all have been found to outperform $\epsilon-$greedy, but are limited due to the underlying models operating at the level of basic actions. In manipulation, often times there are very few actions that would lead to effective new outcomes and are hard to discover from uninformative landscapes of the reward function.

Multigoal RL - Inverse dynamics   The Horde architecture (Sutton et al. 2011) proposed to represent procedural knowledge in terms of multiple Value Functions (VF) learned in parallel by multiple RL agents, each with its own reward and termination condition. VF would capture time for specific events to happen, or, used for inducing a policy for a specific goal. Work of (Schaul et al. 2015) proposed to represent a large set of optimal VF by a single unified function approximator, parametrized by both the state and goal. This takes advantage of the fact that similar goals can be achieved with similar policies, and thus allows generalization across goals, not only across states. Hindsight Experience Replay (HER) (Andrychowicz et al. 2017) introduces the simple idea that failed executions -episodes that do not achieve the desired goal- achieve some alternative goal, and is useful to book-keep in the experience buffer such "failed" experience as successful experience for that alternative goal. Such state and goal parametrized experience is used to train a generalized policy, where actor and critic networks take as input both the current state and goal (as opposed to the state alone). Thanks to the smoothness of actor and critic networks, achieved goals that are nearby the desired one, implicitly guide learning, instead of being discarded. Our approach builds upon HER, both for learning the basic skills, as well as the final complex manipulation policies, yet we propose novel exploration strategies, instead of ϵ-greedy used in (Andrychowicz et al. 2017).

Generalized (multigoal) policies learn to transition for a set of initial states to a set of related goals, and as such, they are equivalent to *multistep* inverse dynamics models. Many works in the literature attempt to learn inverse models with random exploration (Agrawal et al. 2016) and chain them in time for imitation (Nair et al. 2017). Our work learns skill inverse models using multigoal RL and explicit rewards. We are able to discover more useful and interesting temporally extended inverse dynamics models, which is unclear how to obtain with random exploration. For example, a robot would potentially never learn to grasp guided solely by random

controls or curiosity, or at least, this result has not been obtained till today.

Hierarchical RL   Learning and operating over different levels of temporal abstraction is a key challenge in tasks involving long-range planning. In the context of reinforcement learning, Sutton, Precup, and Singh (1999) proposed the options framework, which involves abstractions over the space of actions. At each step, the agent chooses either an one-step primitive action or a multi-step action policy (option). Each option defines a policy over actions (either primitive or other options) and can be terminated according to a stochastic function. The MAXQ framework (Dietterich 1999) decomposes the value function of a Markov Desicion Process (MDP) into combinations of value functions of smaller constituent MDPs. Work of Kulkarni et al.(2016) learns a policy for scheduling semantically meaningful goals using deep Q networks. Work of Riedmiller et al. (2018) uses simple intrinsic perceptual rewards to learn subtasks and their scheduling for helping learning of extrinsic motivated (non-hierarchical) policies. Chentanez, Barto, and Singh (2005) also explored agents with intrinsic reward structures in order to learn generic options that can apply to a wide variety of tasks. Using a notion of salient events as sub-goals, the agent learns options to get to such events. Other works have proposed parametrized actions of discrete and continuous values as a form of macro-actions (temporally extended actions) to choose from (Hausknecht and Stone 2016; Masson and Konidaris 2015). We compare with the model from Hausknecht and Stone in the experimental section. Other related work for hierarchical formulations include Feudal RL (Dayan and Hinton 1993) which consists of "managers" taking decisions at various levels of granularity, percolating all the way down to atomic actions made by the agent. Daniel et al.(2016) jointly learn options and hierarchical policies over them. Such joint search makes the problem more difficult to solve, moreover, options are not shared across policies of different tasks. We instead capitalize over already known options (skills) to accelerate training of more complex ones.

Hierarchical Planning   Planning has been used with known transition dynamic models of the environment to help search over optimal actions to take. Incorporating macro-actions to reduce the computational cost of long-horizon plans has been explored in He, Brunskill, and Roy(2010). In Vien and Toussaint(2015), the authors integrate a task hierarchy into Monte Carlo Tree Search. These approaches work for discrete state, action and observation spaces, and under known dynamics. Our work instead considers continuous actions and states, and unknown dynamics.

Model-based RL   To address the large sample complexity of model-free RL methods, researchers learn models of the domain, which they use to sample fake experience for policy learning (Sutton 1991), initialize a model-free method (Nagabandi et al. 2017) which is further fine-tuned with real experience to fight the biases of the model, or is combined with a model-free estimation of the residual errors (Chebotar et al. 2017). When the model of the domain is given, Monte Carlo tree search has shown to be very effective for exploration (Silver et al. 2017), and outperforms corresponding model-free RL methods (Guo et al. 2014), even when the latter are allowed to consume a great amount of (simulated) experience.

Exploration using look-ahead search over skill dynamics

We consider a multi-goal Markov Decision Process (MDP) (Schaul et al. 2015), represented by states $s \in \mathcal{S}$, goals $g \in \mathcal{G}$, actions $a \in \mathcal{A}$. At the start of each episode, a state-goal pair is sampled from the initial state-goal distribution $\rho(s_0, g)$. Each goal g corresponds to a reward function $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At each timestep, the agent gets as input the current state s_t and goal g, and chooses an action a_t according to a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$, and obtains reward $r_t = r_g(s_t, a_t)$. The objective of the agent is to maximize the overall reward.

Learning multi-goal skill policies

We endow our agent with an initial set of manipulation skills. We define a skill as a short-horizon generalized action policy that achieves *a set of related goals*, as opposed to a single goal. The goal sets of individual skills are not related to the goals of our final manipulation tasks. Our framework can handle any set of skills, independent of their length, complexity, state and action spaces. Furthermore, skills can be redundant within the set. We trained three skills:

- *reach*, in which the gripper reaches a desired location in the workspace while not holding an object,

- *grasp*, in which the gripper picks up an object and holds it at a particular height,

- *transfer*, in which the gripper reaches a desired location in the workspace while holding an object.

The skills do not share the same state space: each involves different number of objects or it is oblivious to some part of the state space. Control is carried out in task space, by predicting directly dx, dy, dz of the motion of the end-effector and the gripper opening. Task space control allows easier transfer from simulation to a real robotic platform and is agnostic to the exact details of the robot dynamics. The skills do not share the same action space either, e.g., the reaching skill does not control the gripper open/close motion. State and action abstraction allows faster skill training. Details on the particular skill environments and corresponding states, actions and rewards functions are included in the supplementary material.

Each skill is trained using Hindsight Experience Replay (Andrychowicz et al. 2017) (HER) and off-policy deep deterministic policy gradients (DDPG) (Lillicrap et al. 2015) with standard ϵ-greedy exploration (Andrychowicz et al. 2017). This allows us to decouple exploration and policy learning. The agent maintains actor $\pi : \mathcal{S} \times \mathcal{G} \to \mathbb{A}$ and action-value (critic) $Q : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ function approximators. The actor is learned by taking gradients with

respect to the loss function $\mathcal{L}_a = -\mathbb{E}_s Q(s, g, \pi(s, g))$ and the critic minimizes TD-error using TD-target $y_t = r_t + \gamma Q(s_{t+1}, g, \pi(s_{t+1}, g))$, where γ is the reward discount factor. Similar to (Andrychowicz et al. 2017), we use a binary reward function r_g which is 1 if the resulting state is within a specified radius from the goal configuration g, and 0 otherwise. Exploration is carried out by adding ϵ normal stochastic noise (Andrychowicz et al. 2017) to actions predicted by the current policy.

HER, alongside the intended experience tuples of the form $(s_t, a_t, rg_t, g, s_{t+1})$, given the resulting state s_T of an episode of length T, it adds additional experience tuples in the buffer, by considering s_T to be the intended goal for the experience collected during the episode, namely, adds tuples of the form $(s_t, a_t, r(s_T)_t, s_T, s_{t+1})$. All the tuples in the experience buffer are use to train the actor and critic networks using the aforementioned reward functions. For more details, please refer to (Andrychowicz et al. 2017).

Learning coarse-grain skill dynamics and success probabilities

Multi-goal skill policies are used to obtain **general and purposeful forward dynamic models, that cover rich part of the state space and which are not easy to learn from random exploration.** For the ith skill, we learn

- a **coarse** transition function of the form $\mathcal{T}_{coarse}^i : (s, g) \rightarrow s_{final}$ which maps an initial state s and goal g to a resulting final state s_{final}, and

- a success probability function $u^i(s, g) \rightarrow [0, 1]$, that maps the current state and goal to the probability that the skill will actually achieve the goal.

We learn $\mathcal{T}_{coarse}$ and u after each skill is trained. Data tuples are collected by sampling initial states and goals and running the corresponding skill policy. The collected data is used to train deep neural regressors for each skill, a three layer fully connected network that takes as input a state and a goal configurations and predicts the final state reached after skill execution, and the probability of success. The detailed architecture of the dynamics neural networks is included in the supplementary. Each manipulation skill is represented by (generalized) policy π, action-value function Q, transition dynamics and probability of success: $\mathcal{K} = \{\Omega^i = (\pi^i, Q^i, u^i, \mathcal{T}_{coarse}^i), i = 1 \cdots N\}$, where N is the number of skills and $\mathcal{K}$ the skill set. For us $N = 3$. Our coarse skill dynamics model learns to predict the outcome of skill execution (on average 25 timesteps long) instead of predicting the outcome of low-level actions. This allows to plan over longer horizons without severe dynamics error accumulation. Although HER with ϵ-greedy exploration successfully learns generalized basic skills guided solely by binary reward functions, it fails to learn more complex skills, such as, *put object A inside container B*, that require longer temporal horizon. Next, we describe how we use knowledge of coarse skill dynamics $\mathcal{T}_{coarse}$ and success probabilities u for exploration during training of such complex manipulation policies.

Exploration with look-ahead search

We will use again DDPG with HER to train manipulation policies over low-level actions, but with a better exploration method. DDPG, being an off-policy algorithm, allows use to decouple learning and exploration. We exploit this decoupling and place our look-ahead search at the exploration step. With ϵ probability we use the proposed look-ahead search to select the next action to try a_t, otherwise we follow the current policy learned thus far $\pi(s_t, g)$, where s_t denotes the current state. We vary ϵ to be close to 1 in the beginning of training, and linearly decay it to 0.001.
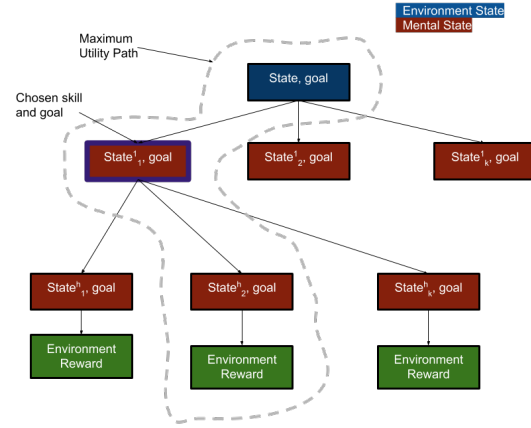

Figure 2: **Look-ahead Search:** From the current environment state, we sample skill identities and skill sub-goals and (mentally) unfold a look-ahead tree using learned skill dynamics. We select the first skill and sub-goal of the path with maximum utility and execute it in the "real world".

Our look-ahead search works as follows: At each episode step, we unfold a search tree by sampling (with replacement) at each tree level, for each branch, a set of K skill identities (in our case one of *reach*, *grasp* or *transfer*), and corresponding K skill sub-goals, where K is the branching factor of the search. For each sampled skill identity and sub-goal, we use the learned success probabilities of each sampled skill and sub-goal combination and prune improbable transitions (line 13 in Algorithm 2). For the remaining skill/subgoal combinations we (mentally) transition to the resulting final state following the learned skill dynamics function. After unfolding the tree for a prespecified number of steps, we choose the path with the maximum total reward defined as the sum of the transition rewards(reward for going from one node to other connnected node) of the skill executions as measured by the skill critic networks Q^i and proximity of the final state to the desired goal configuration: $R = \sum_{(s, k \in \mathcal{K}, g_k) \in \text{path to leaf node}} Q^k(s, g_k) + r_{\text{final}}$, where r_{final} is the negative of the Euclidean distance between the final state and the desired goal configuration g. We execute the first (skill, sub-goal) tuple on the chosen maximum utility path in the current ("real") environment until the skill terminates, i.e., until the skill sub-goal is achieved or maximum

Algorithm 1 HER with look-ahead search exploration(HERLASE)

1: **Input:**
2:     skill set $\mathcal{K}$
3:     reward function $r : -\mathbb{1}[f_g(s) = 0]$
4:     $\epsilon \leftarrow 1$, skill terminated $\leftarrow$ true
5: Initialize π, Q, Replay buffer B
6: **for** episode = 1, M **do**
7:     Sample a goal g and starting state s_0
8:     **while** episode not done **do**
9:         **if** random(0,1) $< \epsilon$ **then**
10:             **if** skill terminated **then**
11:                 $\Omega^i, g^i \leftarrow$ TreeSearch(s_t, g)
12:                 $(\pi^i, Q^i, u^i, \mathcal{T}^i_{\text{coarse}}) \leftarrow \Omega^i$
13:             **end if**
14:             $a_t = \pi^i(s_t, g^i)$
15:         **else**
16:             $a_t = \pi(s_t, g)$ + Gaussian noise
17:         **end if**
18:         $s_{t+1}, r_t,$ terminal = execution(a_t)
19:         skill terminated $\leftarrow$ checkSkillTermination(s_{t+1})
20:     **end while**
21:     Create hindsight experience with $g' = s_T$
22: **end for**

Algorithm 2 TreeSearch

1: **Input:**
2:     maxHeight H, branchingFactor B
3:     goal g, initial state s_t^{real}, skill set $\mathcal{K}$
4: Initialize
5:     root $\leftarrow (s_t^{\text{real}}, 0, 0)$
6:     openlist $\leftarrow$ addRoot
7:     leafNodelist $\leftarrow \{\}$
8: **while** all path explored **do**
9:     s, currHeight = getTopLeafNode(openlist)
10:     sampled Set = sample skills and goal parameters(B)
11:     **for** $\Omega^i, g_i \in$ sampled Set **do**
12:         $(\pi^i, Q^i, u^i, \mathcal{T}^i_{\text{coarse}}) \leftarrow \Omega^i$
13:         **if** $u^i(s, g_i) > 0.5$ **then**
14:             nextState $\leftarrow \mathcal{T}^i_{coarse}(s, g_i)$
15:             $a_i \leftarrow \pi^i(s, g_i)$
16:             transitionReward $\leftarrow Q^i(s, g_i, a_i)$
17:             **if** currHeight+1 $<$ H **then**
18:                 AddToLeafNodelist(nextState,
                            transitionReward, currHeight+1)
19:             **else**
20:                 AddNodeToOpenlist(nextState,
                            transitionReward, currHeight+1)
21:             **end if**
22:         **end if**
23:     **end for**
24: **end while**
25: bestPath = getBestPath(leafNodeList)
26: Return first skill and goal parameter of bestPath

---

skill episode length is reached or goal is reached. The experience in terms of tuples $(s_t, g, a_t, r_t, s_{t+1})$ populate the experience replay buffer.

Note that the learned skill dynamics $\mathcal{T}_{coarse}$ and u may not match the dynamics in the current environment. The reason could be both due to approximation error of the neural network regressors and the difference in the environment dynamics, e.g., task environments may contain additional objects on which the gripper can collide, which were not present during skill learning. Our look-ahead exploration is described in Algorithm 2 and visualized in Figure 2. The complete exploration and reinforcement learning method is described in Algorithm 1.

Experiments

We test the proposed method in the MuJoCo simulation environment (Todorov, Erez, and Tassa 2012) using a seven degree of freedom Baxter robot arm with parallel jaw grippers in the following suite of manipulation tasks:

- **Pick and Move** (Figure 3a) The robot needs to reach towards the object, grasp it and move to a target 3D location.

- **Put A inside B** (Figure 3b) The robot needs to reach towards the object, grasp it and put it inside the container.

- **Stack A on top of B** (Figure 3c) The robot needs to reach towards the red object, grasp it and put it on top of the purple object.

- **Take A out of B** (Figure 3d) The robot needs to reach towards the object inside the container, grasp it and take it out of the container. The objective of this environment to grasp the objective in cluster container, then move

(a) Pick and and Move   (b) Put A inside B

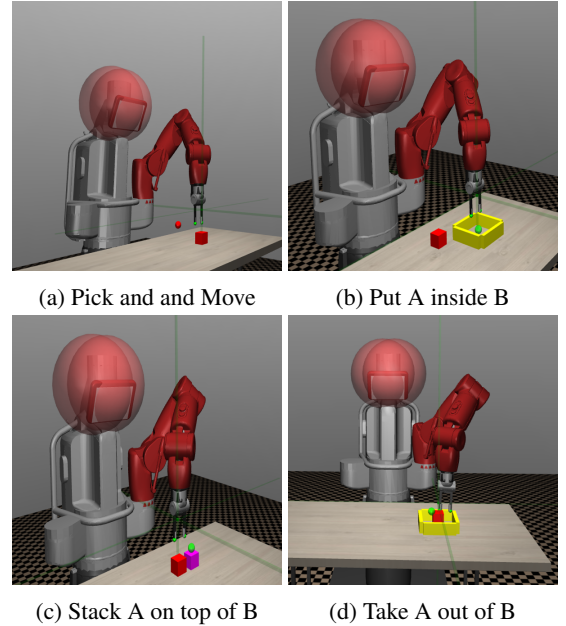(c) Stack A on top of B   (d) Take A out of B

Figure 3: Suite of robot manipulation tasks with baxter robot with end-effector control and parallel jaw gripper.

out of container and move it to any 3D location in the workspace.

All the tasks in our benchmark suite require long temporal horizon policies, which is a challenge when learning from sparse rewards.

1. How well the proposed exploration strategy performs over ϵ-greedy?

2. How out learnt policies compare with policies assembled directly over macro-actions or options?

3. What is the impact of the chosen skill set to the proposed method?

4. What happens when the dynamics are quite different in the new environment?

We evaluate our method against the following baselines:

1. Hindsight Experience Replay (*HER*), described in (Andrychowicz et al. 2017). In this method, exploration is carried out by sampling noise from an ϵ-normal distribution and adding it to the predicted actions of the policy learned so far. This is a state-of-the-art model-free RL method for control.

2. Parameterized action space (*PAS*), described in (Hausknecht and Stone 2016). This approach uses the learned skills as macro-actions by learning a meta-controller which predicts probability over those skills, as well as the goal of each skill.

3. Parameterized action space + Hierarchical HER (*HER-PAS*), as described in (Levy, Platt, and Saenko 2017). We extend the model of (Hausknecht and Stone 2016) by creating additional hindsight experience at macro-level, i.e., in the parameterized action level (skill and sub-goal). Specifically, we replace the goal state in the macro-action transitions collected during the episode which failed to achieve goal g with the $g' = s_T$ and evaluate all the low level actions, chosen using the skill and sub-goal, with the new reward function $r_t = r_{g_{\text{new}}}(s_t, a_t)$.

4. Parameter Space Noise + HER(*HER+ParamNoise*), described in (Plappert et al. 2017), exploration is carried out in parameter space similar to evolutionary strategies.

Success plots for the different tasks for our method and baselines are shown in Figures 4. Our method significantly outperforms the baselines in terms of final performance and sample complexity of environment interactions. In "put A inside B" task 3b *HER* is not able to succeed at all. PAS show success early in training but converges to sub-optimal performance. This is expected due to the restriction imposed on the policy space by to the hierarchical structure.

Sensitivity to skill set selection We use skills (macro-actions) for exploration, and not for assembling a hierarchical policy, as done in the option framework (Sutton, Precup, and Singh 1999). Therefore, we expect our method to be less sensitive to the skill selection, in comparison to previous methods (Sutton, Precup, and Singh 1999). To quantify such flexibility we conducted experiments using three different skill sets: faster convergence. To test this hypothesis, we

(a) Pick and Move   (b) Put A inside B

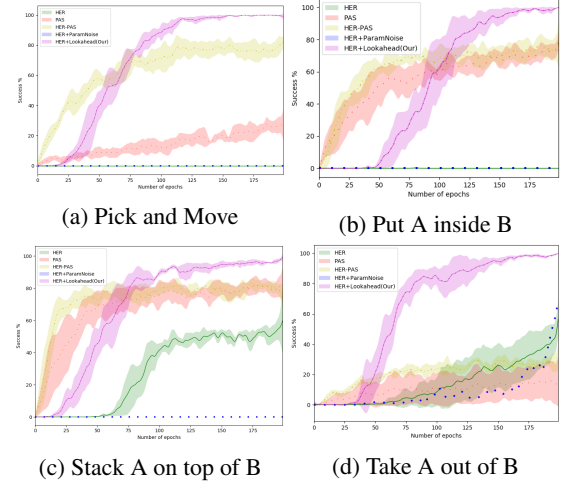(c) Stack A on top of B   (d) Take A out of B

Figure 4: The success plot for each manipulation task in our suite. For evaluation, we freeze the current policy and sample 20 random initial states and goals at each epoch (1 epoch = 16 episodes of environment interaction).
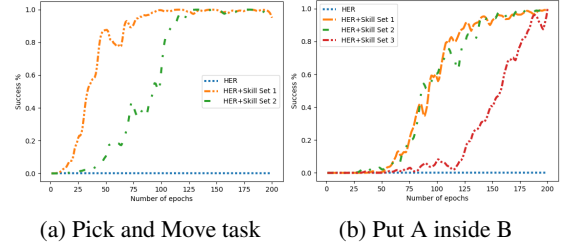

(a) Pick and Move task   (b) Put A inside B

Figure 5: **Sensitivity of our method to skill set selection.** We show the success plots for "Pick and move" and "Put A inside B" tasks. Look-ahead exploration leads to slower learning when essential skills are missing. However, it is still better than random exploration.(1 epoch = 16 episodes of environment interaction).

used 3 skill set $\mathcal{K}_1 = \{\text{transit, grasp, transfer}\}$, $\mathcal{K}_2 = \{\text{grasp, transfer}\}$ and $\mathcal{K}_3 = \{\text{transit, transfer}\}$. We tested our method with $\mathcal{K}_2$ on the "Pick and Move", and "Put A inside B" tasks and show results in Figure 5a. Exploring with all the skills ($\mathcal{K}_1$) leads to faster convergence of policy learning than using $\mathcal{K}_2$. This is easily explained as the agent needs to learn the transit skill via interactions generated by other skills and the current policy. We did not observe any slower learning for "Put A inside B" task using $\mathcal{K}_2$, as shown in Figure 5b. The reason for this is that the transit skill is similar to the transfer skill, which is present. Learning of "Put A inside B" task using $\mathcal{K}_3$ is slower (Figure 5b). This is due to the fact that grasping is a critical skill in completing this task. However our method is still able to learn the task from scratch, while *HER* fails to do so.

Sensitivity to model errors We quantify sensitivity of our method to model errors, i.e., the accuracy of dynamics skill models. We created a perturbation in our "Put A in-

side B" task by making a wall insurmountable by the robot arm, shown in Figure 6a. The skill dynamics that concern end-effector states near the large wall would be completely wrong, since this wall was absent during learning of our bacis skills and therefore their dynamics. However, the proposed look-ahead exploration is still beneficial, as seen in Figure 6b. Our method succeed to learn task, while *HER* with random exploration fails to do so. We used our full skill set in this experiment, namely, transit, transfer and grasp skills.

(a) Put A inside B with 1 insurmountable wall

(b) Success Curve
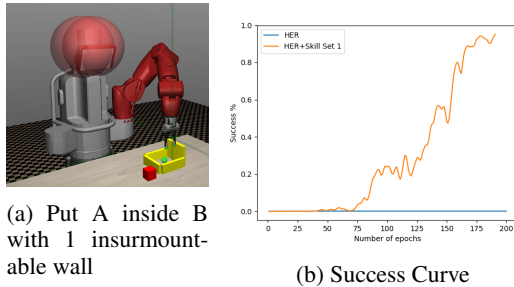
Figure 6: **Sensitivity of our method to model errors.** In (A) we show the perturbed version of "Put A inside B" task in which a wall is very high and cannot be crossed by the agent. In (B) we show that even with wrong coarse dynamics model our approach works better than random exploration. (1 epoch = 16 episodes of environment interaction).

Scalability Our method trades time of interacting with the environment with time to "think" the next move by mentally unfolding a learned model of dynamics. The "thinking" time depends on the speed of the look-ahead search. We experimented with different branching factors for the tree search.

With ϵ-greedy the agent takes 0.4 seconds per episode (50 steps), with branching factor (bf) equal to 5 the agent takes 17 seconds, with bf=10 it takes 71 seconds, and with bf=15 it takes 286 seconds. However, for complex tasks in our benchmark suite, we did not observe empirical advantages from larger branching factors, so all the reported results use bf=5. The sampling process and pruning could be implemented in parallel on GPU, which will render our tree search much more efficient. We are currently exploring learning-based ways to guide the tree unfolding.

Conclusion - Future work

We proposed an exploration method that uses coarse-time dynamics of basic manipulation skills for effective lookahead exploration during learning of manipulation policies. Our empirical findings suggest that the proposed look-ahead exploration guided by learned dynamics of already mastered skills, can effectively reduce sample complexity when mastering a new task, to the extent that skills that are impossible with naive exploration, are possible with the proposed dynamics-based look-ahead exploration, suggesting an avenue for curriculum learning of manipulation policies, by continually expanding the skill set. The resulting policies are still in the space of primitive low-level actions, which allows flexibility on choosing skills, and on the resulting reactive policy learner.

If the proposed exploration strategies are used in environments that have different dynamics than the environments used to train the basic skills (e.g., there is a wall present, or the object sizes are very different), learning is slower, and our exploration scheme offers less advantages due to model errors. The fact that our dynamics are not updated online is a limitation of our method, and an avenue for future work. All the experiments performed in our paper are based on the centroid estimates of the objects of interest. It would be interesting to explore policies and models learned directly from visual features in future work.

References

Agrawal, P.; Nair, A.; Abbeel, P.; Malik, J.; and Levine, S. 2016. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR* abs/1606.07419.

Andrychowicz, M.; Crow, D.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 5055–5065.

Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. *CoRR* abs/1606.01868.

Chebotar, Y.; Hausman, K.; Zhang, M.; Sukhatme, G. S.; Schaal, S.; and Levine, S. 2017. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *CoRR* abs/1703.03078.

Chentanez, N.; Barto, A. G.; and Singh, S. P. 2005. Intrinsically motivated reinforcement learning. In Saul, L. K.; Weiss, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems 17*. MIT Press. 1281–1288.

Daniel, C.; Van Hoof, H.; Peters, J.; and Neumann, G. 2016. Probabilistic inference for determining options in reinforcement learning. *Machine Learning* 104(2-3):337–357.

Dayan, P., and Hinton, G. E. 1993. Feudal reinforcement learning. In Hanson, S. J.; Cowan, J. D.; and Giles, C. L., eds., *Advances in Neural Information Processing Systems 5*. Morgan-Kaufmann. 271–278.

Dieterich, T. G. 1999. Hierarchical reinforcement learning with the MAXQ value function decomposition. *CoRR* cs.LG/9905014.

Guo, X.; Singh, S.; Lee, H.; Lewis, R. L.; and Wang, X. 2014. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3338–3346.

Ha, D., and Schmidhuber, J. 2018. World models. *CoRR* abs/1803.10122.

Hausknecht, M., and Stone, P. 2016. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

He, R.; Brunskill, E.; and Roy, N. 2010. Puma: Planning under uncertainty with macro-actions. In *AAAI*.

Kulkarni, T. D.; Whitney, W.; Kohli, P.; and Tenenbaum, J. B. 2015. Deep convolutional inverse graphics network. *CoRR* abs/1503.03167.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. B. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR* abs/1604.06057.

Levy, A.; Platt, R.; and Saenko, K. 2017. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Masson, W., and Konidaris, G. 2015. Reinforcement learning with parameterized actions. *CoRR* abs/1509.01644.

Mayne, D. Q. 2014. Model predictive control: Recent developments and future promise. *Automatica* 50(12):2967 – 2986.

Mohamed, S., and Rezende, D. J. 2015. Variational information maximisation for intrinsically motivated reinforcement learning. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc. 2125–2133.

Nagabandi, A.; Kahn, G.; Fearing, R. S.; and Levine, S. 2017. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR* abs/1708.02596.

Nair, A.; Chen, D.; Agrawal, P.; Isola, P.; Abbeel, P.; Malik, J.; and Levine, S. 2017. Combining self-supervised learning and imitation for vision-based rope manipulation. *CoRR* abs/1703.02018.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. P. 2015. Action-conditional video prediction using deep networks in atari games. *CoRR* abs/1507.08750.

Osband, I.; Blundell, C.; Pritzel, A.; and Roy, B. V. 2016. Deep exploration via bootstrapped DQN. *CoRR* abs/1602.04621.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. *CoRR* abs/1705.05363.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Riedmiller, M. A.; Hafner, R.; Lampe, T.; Neunert, M.; Degrave, J.; de Wiele, T. V.; Mnih, V.; Heess, N.; and Springenberg, J. T. 2018. Learning by playing - solving sparse reward tasks from scratch. *CoRR* abs/1802.10567.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In Bach, F., and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1312–1320. Lille, France: PMLR.

Schmidhuber, J. 1990. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, 222–227. Cambridge, MA, USA: MIT Press.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L. R.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T. P.; Hui, F. X.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550:354–359.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, 761–768. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1):181 – 211.

Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.* 2(4):160– 163.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 5026–5033. IEEE.

Vien, N. A., and Toussaint, M. 2015. Hierarchical monte-carlo planning. In *AAAI*, 3613–3619.

White, R. W. 1959. Motivation reconsidered: The concept of competence. *Psychological review* 66(5):297.