# Deep Reinforcement Learning with Skill Library: Exploring with Temporal Abstractions and coarse approximate Dynamics Models

Arpit Agarwal

CMU-RI-TR-18-31

Submitted in partial fulfillment of the requirements for the degree of Masters in Robotics Research

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

**Thesis Committee:**
Katerina Fragkiadaki(Co-chair)
Katharina Muelling(Co-chair)
Oliver Kroemer
Devin Schwab

July 2018

# *Abstract*

Reinforcement learning is a computational approach to learn from interaction. However, learning from scratch using reinforcement learning requires exorbitant number of interactions with the environment even for simple tasks. One way to alleviate the problem is to reuse previously learned skills as done by humans. This thesis provides frameworks and algorithms to build and reuse *Skill Library*. Firstly, we extend the Parameterized Action Space formulation using our Skill Library to multi-goal setting and show improvements in learning using hindsight at coarse level. Secondly, we use our Skill Library for exploring at a coarser level to learn the optimal policy for continuous control. We demonstrate the benefits, in terms of speed and accuracy, of the proposed approaches for a set of real world complex robotic manipulation tasks in which some state-of-the-art methods completely fail.

# *Acknowledgements*

I would like to thank my advisors, Katerina Fragkiadaki and Katharina Muelling for their mentorship and constant guidance for the past two years. I am extremely grateful for all the valuable insights and feedback which has helped me to learn about research and robotics.

Prof. Manuela Veloso has provided me support and robot for fueling my research. I am grateful to her for making me part of a wonderful CORAL lab. I would especially like to thank Devin and Anahita for providing inspiration, useful discussions and incredible support throughout my stay at CMU. I would like to thank Rui, Vittorio and Philip for being pillars of support academically and personally.

My roommate and lab-mate, Ashwin, has been a always provided his valuable time in debugging, meta-robotic discussions and has been a source of inspiration. He has been there through ups and downs of my stay at CMU. I can't thank Ashwin enough for being a constant into my CMU life.

I would also like to thank OpenAI for releasing high-quality implementations of SOTA algorithms and benchmarking environment. We have built up over their implementations. Finally, I would like to thank Dr. Oliver for reading my thesis and providing useful insights to improve my work.

# Contents

# List of Figures

# List of Tables

*Dedicated to my parents*

# Chapter 1

# Introduction

## 1.1 Motivation

The introduction of general-purpose function approximators, such as neural networks, with general purpose model-free reinforcement learning algorithms that can learn complex behavioral strategies holds the promise of automating a wide variety of robotic tasks.

There have been recent advances in developing better model-free reinforcement learning algorithms [1–3]. Still the amount of data required for simple tasks remain quite high. Moreover as seen in animals, skill composition greatly increases the efficiency to solve new problems [4]. 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.

Learning and operating over different levels of temporal abstraction is a key challenge in tasks involving long-range planning. The two key advantages are exploration at coarse temporal level and using short-range policies to achieve subgoals. We take advantage of both the points and develop approaches which help us solve complex manipulation tasks.

## 1.2 Thesis Contributions

We look at following questions in this thesis

1. Can we come up with a set of manipulation skills which could be used to perform a variety of challenging real-world manipulation tasks?
2. Can learning parameterized-action space DDPG at a coarser level with pre-trained multi-goal skill library be extended to multi-goal environments with hindsight experience replay? 0
3. Can exploration with look-ahead search at a coarser level using our pre-learned skill library lead to success in multi-goal complex manipulation tasks?

## 1.3 Thesis Outline

We first introduce and review the most important concepts of reinforcement learning and deep reinforcement learning works for manipulation in Chapter 2. In this section we also provide rational of the design choices made to make the work useful and reproducible on the real robot. Next, in Chapter 3 we introduce our *Skill Library* and discuss methods used to learn different skills for manipulation. Chapter 4 introduces multi-goal formulation for parameterized action space reinforcement learning. Chapter 5 uses skills to develop new exploration method and describe look-ahead search in continuous state and action space domains. Next, in Chapter 6 we show the benefits of the introduced *Skill Library* and approaches on suite of benchmark robotic manipulation tasks. The results show that the introduce can tackle complex manipulation tasks and can get attain 100% success in tasks in which previous methods completely fail. Finally, we summarize our work in Chapter 7 and point out interesting promising directions for future work.

# Chapter 2

# Background and Related Work

In this chapter, we will introduce notations which will used throughout the thesis work. We also discuss related works and some of the choices we have made for our system to made it reproducible on a physical system.

## 2.1 Reinforcement Learning

The simplest definition of reinforcement learning is the problem of learning by interacting in an environment with the goal of maximizing numerical reward.

### 2.1.1 Basics and Notations

We consider the problem of RL in Markov Decision Processes(MDP) [5]. A Markov decision process consists of tuple $(\mathcal{S}, \mathcal{A}, r, \mathcal{T}, \rho_0)$. The state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous in our case. In each interaction, the agent starts in a state $s \sim \rho_0$(starting state distribution), chooses an action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$ and transitions into a new state $s'$ with unknown probability distribution

$\mathcal{T}(s'|s,a)$.The environment gives bounded reward $r(s,a)$, using $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ after each interaction.

The goal of the agent is to maximize the sum of discounted expected returns, $E_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, where $s_0 \sim \rho_0, a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{T}(.|s_t, a_t)$

## 2.2 Learning for Robotic Manipulation

In recent years, we have seen tremendous success in continuous control using Deep Reinforcement Learning [6], [7], [2] and [8]. Dexterous manipulation is one of the fundamental challenge in robotics. Researchers have been long seeking ways to enable robots to robustly interact with the environment with varied shapes, sizes and physical properties. However it is hard to achieve generalization with manually designed controllers. If the agent can interact with the environment and adapt the controller with variation in the environment, it can achieve greater generalization. The application of deep RL in complex manipulation tasks has shown good promise in dealing with contacts [9], grasping [10] and reduction in manual engineering for individual tasks [1], [11].

## 2.3 Task Space Control

Most of the Reinforcement learning methods which obtain the best performance are model-free algorithms. However, the model-free algorithms have high sample complexity [6]. Therefore generally it is required to train for the task in simulation and transfer to real world with small amounts of real data. There are some works [11] which have successfully shown the transfer to real world robotic tasks. Transferring the joint states policy learned in simulation is difficult to transfer to real world due to differences in simulation and real world dynamics. However in

most of the real world manipulators, we have end-effector controller implemented by the manufacturer, which works with high-fidelity (upto to a mm in our baxter robot). Therefore policies learned in end-effector space offers direct transfer from simulation to real world.

The end-effector space allows to specify constraints in the workspace which can be used to setup safe experiments on the real robot. There is no direct method to transfer constraints in workspace to constraints in joint space for high DoF robotic arm.

The end-effector space is more intuitive to visualize and specify the goal location as all the object locations are specified in 3D space. Therefore policies learned in end-effector are easy to generalize to multiple target object locations, as compared to joint space policies.

Since we have end-effector controller for different robots given by manufacturers, we can easily transfer the end-effector policies to multiple robots without the need to learn the mapping from joint space to end-effector space for individual robots. Note the actual performance of the policy depends on the end-effector controller implementation on the robot. For example, robot arm may hit objects in the workspace due to overshoot.

Due to above reasons, we learn all the policies in task space rather than joint space.

## 2.4 Exploration using Intrinsic Motivation

Effective exploration in MDPs is an unsolved challenge in learning good control policies. Methods such as $\epsilon-$greedy, that either follow the current found policy or sample a random action with a $\epsilon$ 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 [12, 13] or expected improvement of predictive dynamics [14], exploration guided by information maximization [15], state visitation density [16], uncertainty of the value function estimates [17], all have been found to outperform $\epsilon-$greedy, but are limited due to the underlying models operating at the level of basic actions. However, all the above methods force the agent to explore the whole state space even if the information is unnecessary for the current task. In MDPs with large state and action spaces, it is often times hard to explore the full state space to obtain meaningful policies without an intent.

## 2.5 Hierarchical Reinforcement Learning

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 et al. [5] proposed the options framework, which involves abstractions over the space of actions. At each step, the agent chooses either a 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 [18] decomposes the value function of an MDP into combinations of value functions of smaller constituent MDPs. Work of [19] learns a policy for scheduling semantically meaningful goals using deep Q networks. Singh et al. [20] 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 (temporally extended) actions to choose from [21, 22]. We compare with the model from [21] in the experimental section. Other related work for hierarchical formulations include Feudal RL [23, 24] which consists of "managers"

taking decisions at various levels of granularity, percolating all the way down to atomic actions made by the agent.

## 2.6 Multi-task Learning and Transfer Learning

In [25], authors use simple intrinsic perceptual rewards to learn subtasks and their scheduling for helping learning of extrinsic motivated (non-hierarchical) policies. However our works relaxes an assumption, that state space and action space being same for intentions and tasks, made in this approach. In [26] authors propose to train multiple related tasks and then use the training agents as teachers for the current tasks to obtain the policy. This approach limits the agent to be only as good as the teacher agents, which could be arbitrarily bad. In [27], Y. Bengio propose to create common representation which could used across tasks. However these representations has to be learned for the whole state space which doesn't allow to modularize the problem. In our work we are able to learn the skills independent of the main task to be performed.

# Chapter 3

# Building Manipulation Skill Library

In this chapter, we will introduce our Skill Library and various components of a Skill. We will also show some results regarding the skills which we have learned and their components.

## 3.1 Basic Skills Definition

We endow our agent with an initial library of manipulation skills. We define a skill as a short-horizon behavior policy that achieves *a set of related goals* (as opposed to a single goal). The goal sets ($\mathcal{G}^i$) of individual skills are not related to the goals of our final complex manipulation task. The proposed framework can handle any set of skills independent of their length, complexity, state and action spaces. The skills may not share the state and action space with the main task. This allows us to easily learn the skill in simpler environment as opposed to more complicated main task to solve as done in [28].

A skill consist of policy $\pi^i$, state-action value function $Q^{\pi i}$, success probability prediction $p^{\pi i}$ and coarse dynamics model $\mathcal{T}^{\pi i}_{\text{coarse}}$. Formally, we define a skill $\Omega^i = (\pi^i, Q^{\pi i}, p^{\pi i}, \mathcal{T}^{\pi i}_{\text{coarse}})$, where

$$\pi^i : \mathcal{S}^i \times \mathcal{G}^i \to \mathcal{A}^i \tag{3.1}$$

$$Q^{\pi i} : \mathcal{S}^i \times \mathcal{G}^i \times \mathcal{A}^i \to \mathbb{R} \tag{3.2}$$

$$p^{\pi i} : \mathcal{S}^i \times \mathcal{G}^i \to [0, 1] \tag{3.3}$$

$$\mathcal{T}^{\pi i}_{\text{coarse}} : \mathcal{S}^i \times \mathcal{G}^i \to \mathcal{S}^i_{\text{terminal}} \tag{3.4}$$

The state space $\mathcal{S}^i$, goal space $\mathcal{G}^i$, action space $\mathcal{A}^i$ and terminal state space $\mathcal{S}^i_{\text{terminal}}$ are continuous and skill-specific.

Our skill library is a collection of individual skills $\mathscr{L} = \{\Omega^1, \Omega^2, \ldots \Omega^N\}$. In the Section 3.2 we describe the algorithm used to learn actor $\pi^i$ and critic $Q^i$ for skills. In the following Section 3.3, we describe the data collection and learning process for learning coarse skill dynamics $\mathcal{T}^i_{\text{coarse}}$ and success probability $p^i$.

## 3.2 Skill Policy and Critic Learning

Each skill policy and critic is trained using Deep Deterministic policy gradient [6] and Hindsight Experience Replay [1] (HER). In HER, the basic formulation of MDP Section 2.1.1 is modified to take goal as a parameter in $\pi, \rho_0, r$ along with usual inputs. This formulation is particularly useful in binary or sparse reward environment in which the agent obtains useful learning signals only if the agents manages to reach the current episode goal. The basic idea of HER is to use a goal different than the goal sampled at the start of the episode to evaluate transitions obtained in the current episode.

Training is carried out with off-policy deep deterministic policy gradients (DDPG) [6]. This allows us to decouple exploration and policy learning. The agent maintains actor $\pi$ and action value (critic) $Q : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \mathbb{R}$ function approximators. The actor is learned by taking gradients with loss function $\mathcal{L}_{actor} = -\mathbb{E}_s Q(s, g, \pi(s, g))$ 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 $\gamma$ is the reward discount factor. Exploration is carried out choosing random action with $\epsilon$ probability and adding Gaussian noise to the action predicted by current policy.

Note we use open-source implementations from OpenAI [29] for training DDPG agent. We build up everything done in this thesis over basic DDPG implementation from OpenAI.

## 3.3 Coarse-level Skill Dynamics and Success Probability Prediction

After learning the policy ($\pi$) for common skills, we build coarse dynamics model ($\mathcal{T}^{\pi i}_{\text{coarse}}$) which maps starting state $s_0$ and goal $g$ to the resulting terminal state $s_{\text{terminal}}$ for each skill. We also learn success probability predictor $p^{\pi i}$ to predict the probability of success of a given skill $\Omega^i$ in starting state $s_0 \sim \mathcal{S}$ and given goal parameters $g \sim \mathcal{G}$. In the following sections we describe the method for data collection and learning methods used to train the 2 components. Later in Section 5.1, we use these components for exploration to solve complex tasks.

### 3.3.1 Data Collection

For each skill, we sampled 100,000 different starting state and goal locations from the environment in which skills were learned and ran the current learned policy

to obtain the terminal states which led to successful completion of the task. We saved the data is 2 formats - $(s_0, g, s_{\text{terminal}})$ and $(s_0, g, 1)$. We used the first data format for learning course dynamics model and second format as the positive data for learning success probability. For extracting the negative data for learning success probability, we sampled 100000 starting state and goals from environments in which skill doesn't succeed. We saved the data as $(s_0, g, 0)$ and combined with positive data $(s_0, g, 1)$ to train a neural network classifier as described in Section 3.3.3.

### 3.3.2 Coarse Dynamics Model Learning

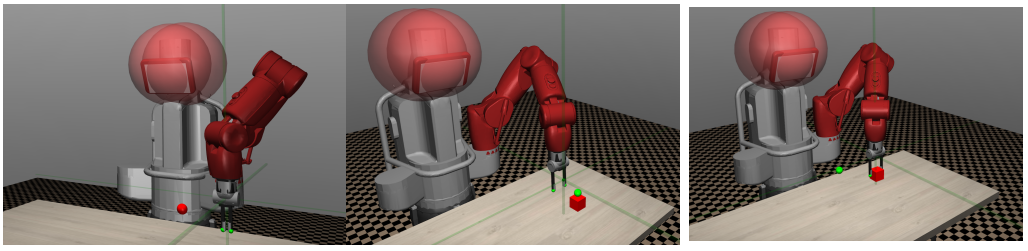
We represent the $\mathcal{T}^{\pi i}_{\text{coarse}}$ with 3-layer fully connected neural network with 1000 neurons in each dense layer and layer normalization [30] after hidden layer. The neural network is trained with l2 loss and Adam optimizer with learning rate of 1e-5 for each skill.

### 3.3.3 Success Probability Predictor

We represent the $p^{\pi i}$ with a 2 layer fully connected neural network with 50 and 10 neurons in dense layers followed by layer normalization. The output activation was sigmoid to predict the probability of success. The neural network is trained with binary cross-entropy loss and Adam optimizer with learning rate of 1e-4 for each skill.

## 3.4 Results

We learned 3 basic skills - transit, grasping and transfer to perform manipulation tasks. Next we describe the skill objectives and their goal parameters.



(A) Transit Skill (B) Grasping Skill (C) Transfer Skill

FIGURE 3.1: The skill trained for manipulation tasks are shown in this figure. (A) Shows the environment in which the end-effector has to move to target location shown in red. (B) Shows the environment in which the object has to grasped and raised to the target shown in green. (C) Shows the environment in which the object has to moved close to the green target location.

### 3.4.1 Transit Skill

The state space of the skill is the 3D location of the end-effector and goal of the skill was 3D target where the end-effector should transit to. The actions were $\Delta x, \Delta y, \Delta z$ of the end-effector in task space due to benefits mentioned in Section 2.3 The objective of this skill is to move the end-effector of the 7 DoF baxter robot arm from random start location in the workspace to random goal location in the workspace. The reward function is binary, i.e. the agent obtains reward 0 when the end-effector is within the tolerance 3cm of the target location and -1 otherwise. Figure 3.1a shows the environment.

### 3.4.2 Grasp Skill

The state space of this skill includes the Cartesian position of the end-effector, its velocities, position and velocity of gripper, objects pose and its velocities. We use a single starting state in which object is grasped. The state space includes the relative position of object which has to be transported in the environment. actions are 4-dimensional with first 3-dimension as the Cartesian motion of the end-effector and last dimension controls the opening and closing of the gripper. The goal location is given as a 3D Cartesian location above the object with height

varying from 5cm to 8cm. The objective of the skill is to move the end-effector of baxter arm from random start location close to (5cm) object, grasp the object and raise it to the desired height. The reward function is binary, i.e. the agent obtains reward 0 when the object is within the tolerance 3cm and -1 otherwise. Figure 3.1b shows the environment.

### 3.4.3 Transfer Skill

The state space and action space of this skill is same as the grasp skill. The goal location is given as a 3D Cartesian location in the workspace of the baxter arm. The objective of the skill is to move the object, grasped between the gripper, from random location in workspace to random 3D target location in workspace. Figure 3.1c shows the environment.

# Chapter 4

# Parameterized action space MDPs with skills library

In Chapter 3, we built a skill library which has a policy to perform meaningful operation in the environment. In this chapter, we explore the use of skill library as primitive actions and learning meta-controller over those primitive actions. This allows us to learn with temporal abstraction which is essential for long range planning and control.

Some of the prior works [19], [31] in deep RL, learned a 2-level hierarchy, in which the top level predicts Q-value over discrete choices or directly predicts the goal parameters as a state which are then used by the low level. However discretization in robotic controller leads to loss in precision of control or explosion in the number of discrete actions for the manager and prediction of goal doesn't allow to group the controllers with similar capabilities with multiple goal locations into a simple skill. Therefore to take advantage of temporal abstraction without incorporating the above problems, we formalize the problem of learning meta-controller as a Markov Decision Process with parameterized action space [22]. Note that parameterized action space formulation has been used in [21] and [32] with deep neural network

function approximators. We investigate the advantages of various improvements made in original DDPG [6] formulation with continuous actions to parameterized action space DDPG with off-policy learning algorithm[21]. We show that benefits carry over directly without much modification.

The overview of the approach is shown in Figure 4.1. We briefly describe the original formulation in Section 4.1 and the modification in Section 4.2

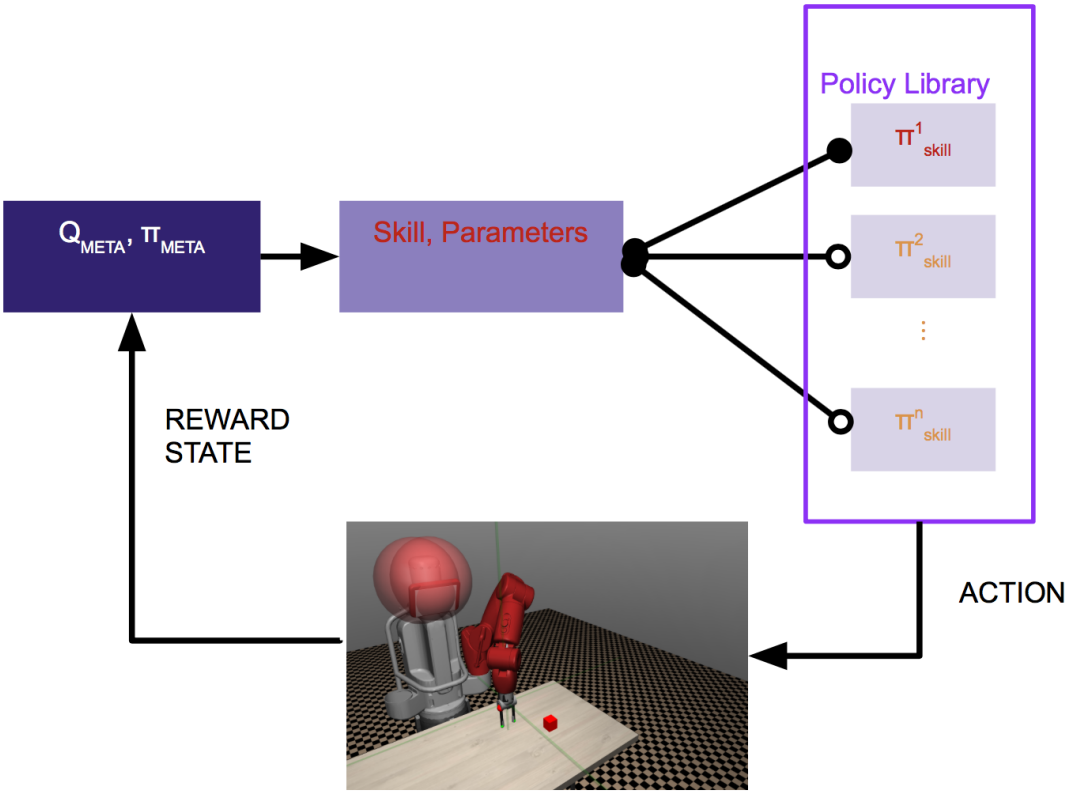



FIGURE 4.1: **Overview of Multi-goal PAS MDP**. We learn a meta-controller which predict the probability and continuous parameter over skills in our skill library.

## 4.1 Parameterized Action Space

The formulation of the MDP remains the same as defined in Section 2.1.1 with action space being hybrid combination of discrete and continuous parameters. There are finite set of discrete actions $A_d = \{a_1, a_2, \ldots, a_n\}$, and each $a \in A_d$ has

a set of continuous parameters $X_a \subset \mathbb{R}^{m_a}$. An action is a tuple (a,x) where a is a discrete action and x are the parameters for that action. The action space is defined as follows.

$$\mathcal{A} = \bigcup_{a \in A_d} \{(a, x) | x \in X_a\}$$

## 4.2 Multi-goal Parameterized Action Space DDPG

In the earlier formulation, the authors [21] used the skill with an implicit global location encoded in reward function for ex: the fixed location where object should be placed. However, as done in [1], we extend the original parameterized action space formulation to multi-goal parameterized action space formulation (MPAS MDP) in which we learn generalized policy and critic for multiple goal locations. This extension leads to faster and higher success than origin formulation single goal formulation. The modifications from the original formulations introduced in Section 2.1.1 are as follows:

$$\mathcal{S} \subset \mathbb{R}^S \tag{4.1}$$

$$\mathcal{A} = \bigcup_{a \in A_d} \{(a, x) | x \in X_a\}, \text{where } X_a \subset \mathbb{R}^{m_a} \tag{4.2}$$

$$\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \tag{4.3}$$

$$r : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R} \tag{4.4}$$

We call our method Multi-goal parameterized action space MDP or MPAS MDP.

We show results using the introduced approach in Chapter 6 for which we use 3 skills, namely transit, grasping and transfer. Transit skill has 3 continuous parameters denoting the target location of the end-effector. Grasping skill has 1 continuous parameters denoting the height up to which the object has to raised.

Transfer skill has 3 continuous parameters denoting the location where object has to be transported.

# Chapter 5

# Using Skill Library for Exploration

In the last chapter, we used the temporal abstraction for learning the policy for the task. However learning over fixed set of goal-parameterized skills restrict the meta-controller to a subset of all the possible policies which can be learned as noted in [18] and may lead to sub-optimal performance. Therefore in this chapter we introduce another method for using skill library and add short horizon search at coarse level to explore a high dimensional complex continuous control problem. The Figure 5.1 gives the overview of our approach. In the following sections describe how we build our look ahead search tree and use it for exploration.

## 5.1 Look-Ahead Search Tree

During lookahead search, we use the current state-goal pair and sample skills $\Omega^i$ from the library L and goal parameters of the skill. We built a tree with root node as current real state $s_0^r$ and add an edge to next node with imagined state $s_1^{\text{imag}} = \mathcal{T}^i_{\text{coarse}}(s_0^r, g^1_{sampled}), g^1_{sampled} \in \mathcal{G}^i$, total reward $r_1 = Q^i(s_0^r, g^1_{\text{sampled}}, \pi^i((s_0^r, g^1_{\text{sampled}}))$

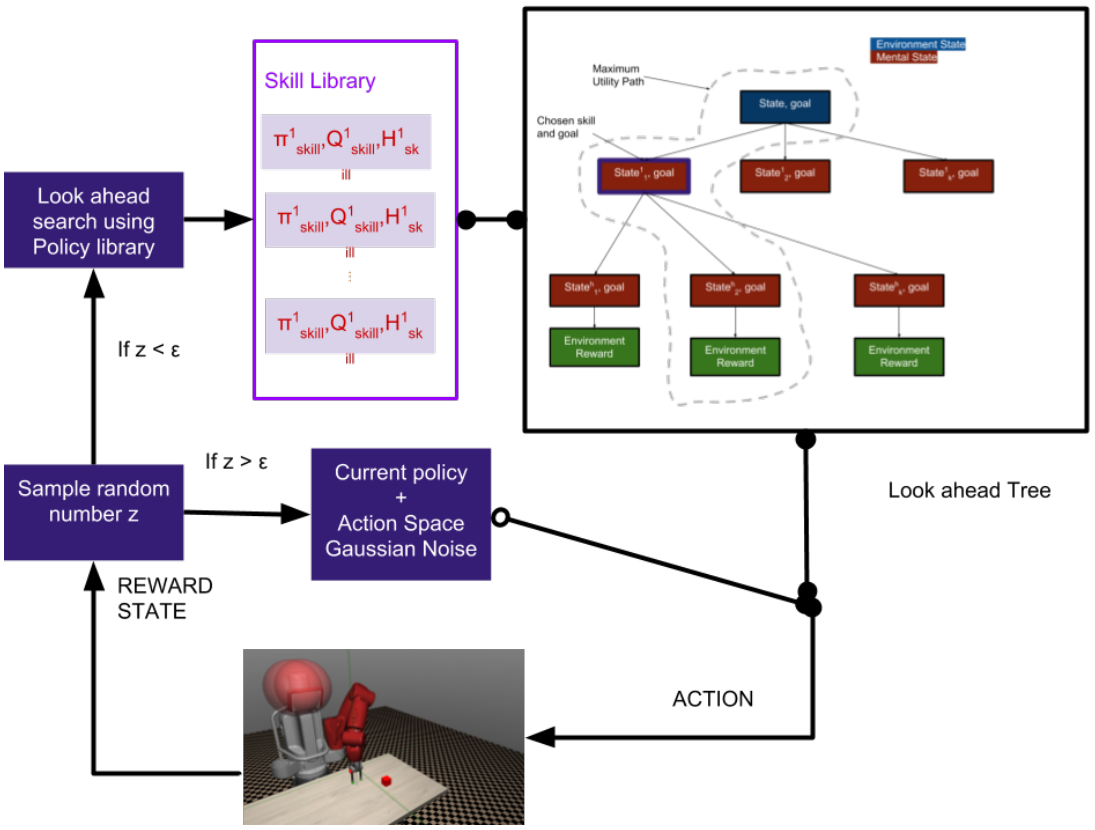



FIGURE 5.1: **Overview of Learning with exploration using skill Library**. We use learned skill dynamics encoded in deep neural regressors for lookahead tree search, in order to aid effective exploration in reinforcement learning of complex manipulation tasks.

and probability of success $p_1 = p^i(s_0^r, g^1_{\text{sampled}})$. We repeat addition of edges with this imagined state $s_1^{\text{imag}}$ if the $p_1 > 0.5$ or maximum height of the tree is not reached. The pictorial representation of building look ahead search tree is shown in Figure 5.2. We describe the skill sampling, tree pruning and next state extraction using skill library below.

**Sampling skills and goals:** At each node we sample K=5 goals and parameters. Since we are learning in task space, we could bias sampling of goal parameters of transit and grasp skill in the vicinity (3cm) of the object ($g \sim$ object location + Uniform$(-0.03, 0.03)$). Note that the information related to object can be grounded in terms of sparse rewards based on sensory input as done [28]. We also tried to

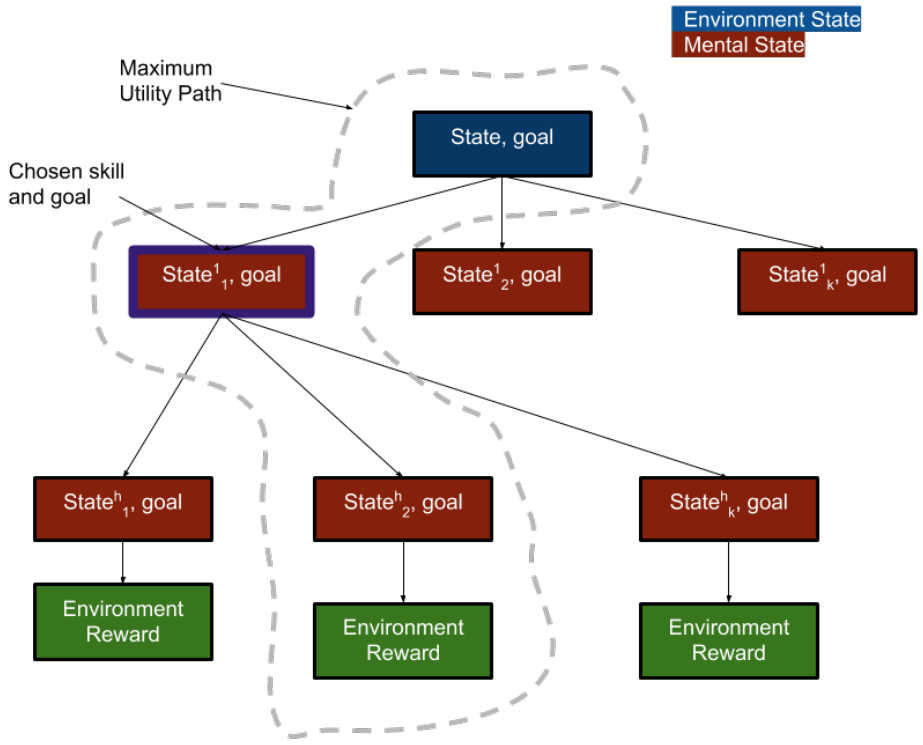



FIGURE 5.2: **Look ahead Search:** The approach uses skill library to build a finite depth and branching factor look ahead tree from the current environment state. We select the first skill and goal parameter of the path with maximum utility.

use mental meta-critic (without any environment interaction of the test environment) to rank the sampled skill and expand only if the value is above a threshold. However the results weren't promising. Therefore we stuck with random sampling of skills and biased goal sampling.

**Pruning tree based on skill success predictor** As defined in the definition of skills Equation 3.4, for each skill we are given $p^{\pi i}$ which gives us the probability of success of the skill using the learned skill policy $\pi^i$. We evaluated sampled skill and goal parameters according to above sampling process and stopped expansion if the success of the skill in achieving the sampled goal was less than the threshold (0.5 in our experiments).

### 5.1.1 Next State Extraction

Here, we describe the process of getting the next coarse level state with the sampled skill and goal parameter to extend the look-ahead tree.

**Using skill memories:** We collected the dataset (memories) for each skill as described in Section 3.3.1. An example tuple in the dataset is represented as (s,g,$s_{\text{terminal}}$). Then we choose the skill memory tuple($\hat{t} = (\hat{s}, \hat{g}, \hat{s}_{\text{terminal}})$) which is closest to the current state $s_t$ and the sampled goal parameter in terms of euclidean distance. We designated the next state as the terminal state ($\hat{s}_{\text{terminal}}$) of closest tuple. Note the state space of different skills are not of same dimension. We took euclidean distance of the dimension which were relevant only for the skill. We kept all the other state variables to be same as the starting state.

**Using skill coarse dynamics model:** The above approach for next state prediction is useful when we have small state space. Therefore to make our approach more generalizable, we used skill coarse level dynamics function $\mathcal{T}^{\pi i}_{\text{coarse}}$ to predict the next state. Given the sampled skill i and goal parameters g, the coarse level dynamics model prediction $s_{t+1} = \mathcal{T}^{\pi i}_{\text{coarse}}(s_t, g)$

| Skill | Memory based retrieval | Neural network based prediction |
|---|---|---|
| Transit | 1.9cm | 1.6cm |
| Grasping | 2.8cm | 1.2cm |
| Transfer | 4.6cm | 2cm |

TABLE 5.1: Comparison between methods for next state prediction, namely, memory based next-state retrieval and neural network coarse dynamics regression model

Formally our approach is described in Algorithm 1. We choose the utility of the path as sum of critic values of each individual skill and leaf state euclidean distance from the global goal location. More formally,

$$U_{\text{path}} = \sum_{(s_t^{\text{imag}}, i, g_i, c) \in \text{path nodes}} \gamma^c Q^{\pi i}(s_t^{\text{imag}}, g_i) + \gamma^C r(s_T^{\text{imag}}, g_i)$$

**Algorithm 1** LookAheadSearch($s_t^{\text{real}}$, $g$)

**Given:** maxHeight, branchingFactor
Initialize
  root $\leftarrow$ ($s_t^{\text{real}}$,0, 0)
  openlist $\leftarrow$ addRoot
  leafNodelist $\leftarrow$ {}
**while** all path explored **do**
  $s$, currHeight = getTopLeafNode(openlist)
  sampled Set = sample skills and goal parameters
  **for** $\Omega^i, g_i \in$ sampled Set **do**
    $(\pi^i, Q^{\pi i}, p^{\pi i}, \mathcal{T}^{\pi i}_{\text{coarse}}) \leftarrow \Omega^i$
    prob $\leftarrow p^i(s, g_i)$
    **if** prob $> 0.5$ **then**
      nextState $\leftarrow \mathcal{T}^i_{coarse}(s, g_i)$
      $a_i \leftarrow \pi^i(s, g_i)$
      totalReward $\leftarrow Q^i(s, g_i, a_i)$

      **if** currHeight+1 $<$ maxHeight **then**
        addToLeafNodelist(nextState, totalReward, currHeight+1)
      **else**
        AddNodeToOpenlist(nextState, totalReward, currHeight+1)
      **end if**
    **end if**
  **end for**
**end while**
bestPath = getBestPath(leafNodeList)
Return first skill and goal parameter of bestPath

## 5.2 Exploration using Look-Ahead Search

In this section we describe how we used look ahead search we built in Section 5.1 to explore at a coarse level and perform complex manipulation tasks. The Figure 5.1 gives an overview of our algorithm. We used the formulation of multi-goal MDP as used in [1] to learn the manipulation tasks with exploration from look-ahead search. The full approach is defined formally in Algorithm 2.

We start with a given skill library $\mathcal{L}$, binary reward function and $\epsilon = 1$. The training loop consists of generating experience using behavior policy and training the current policy using off-policy deterministic gradients. With our look-ahead

**Algorithm 2** HER with lookahead search exploration

Given:
    skill library L
    reward function $r : -\mathbb{1}[f_g(s) = 0]$
    $\epsilon \leftarrow 1$
    skill terminated $\leftarrow$ true
Initialize $\pi, Q$, Replay buffer B
**for** episode = 1, M **do**
  Sample a goal g and starting state $s_0$
  **while** episode not done **do**
    **if** random(0,1) $> \epsilon$ **then**
      **if** skill terminated **then**
        $\Omega^i, g^i \leftarrow$ LookAheadSearch($s_t$, $g$)
      **end if**
      $a_t = \pi^i(s_t)$
    **else**
      $a_t = \pi(s_t)$ + Gaussian noise
    **end if**
    $s_{t+1}$, $r_t$, terminal = execution($a_t$)
    skill terminated $\leftarrow$ checkSkillTermination($s_{t+1}$)
  **end while**
  Create hindsight experience with $g' = s_T$
**end for**

exploration, the behavior policy builds a look ahead search tree from the current state $s_t$ with probability $\epsilon$ and chooses the current policy plus Gaussian noise with probability $1 - \epsilon$. The look-ahead search returns the suggested skill $\Omega^i$ and its goal parameters $g^i$. If the skill is used generating experience, we choose the action using $\pi^i$ and check if the skill goal is reached or skill has been executed w (=10 in our experiments) times for exploration. We built hindsight experience at the end of each episode and store it our replay buffer $B$ as usual.

Note the key differences with the approach described in [33] are as follows:

- We don't assume a fixed hierarchy at the start of learning a task MDP
- We do fixed height look-ahead search and only take $1^{st}$ skill and its corresponding goal of the best path. We then execute this skill to achieve sampled skill goal or finite number of time-steps, after which we re-plan.

- With $1 - \epsilon$ we sample using the current low level learned policy instead of sampling a skill and its goal using look-ahead search.

# Chapter 6

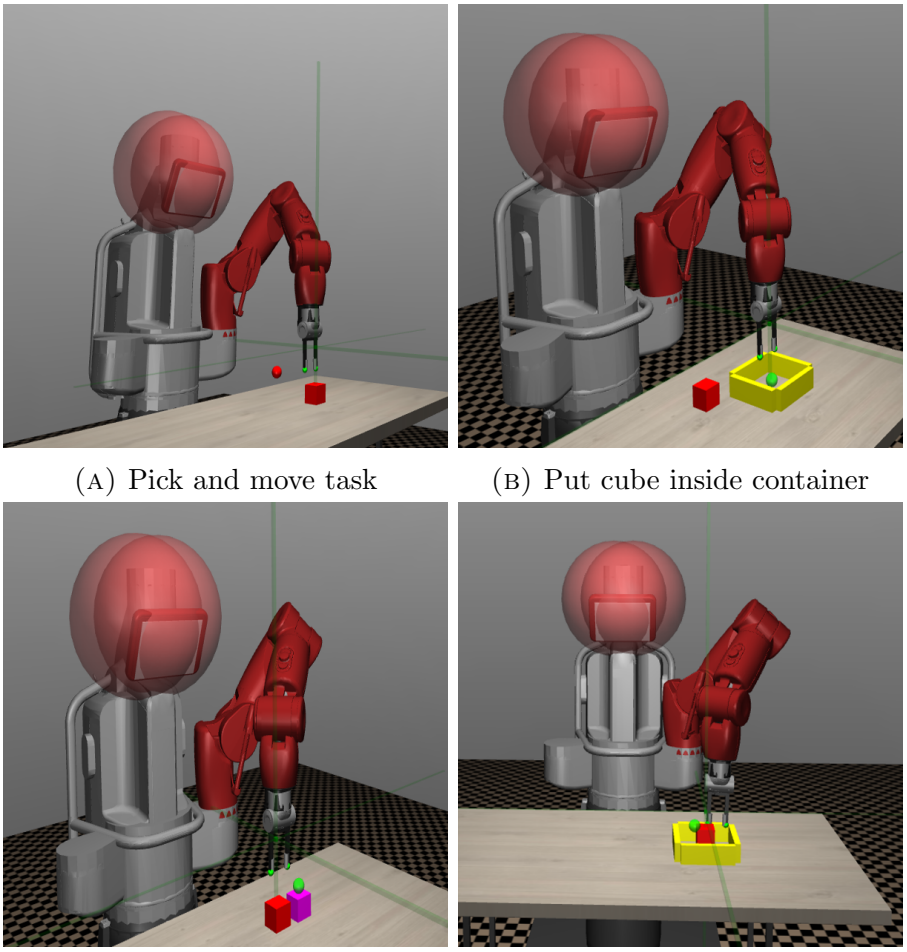
# Experimental Setup and Results

In this chapter, we evaluate the approaches proposed in Chapter 4 and Chapter 5 for a variety of complex robotic manipulation tasks. In Section 6.1, we describe the details about the suite of manipulation tasks we built with baxter robot containing 7 DoF arm and parallel jaw gripper to test our approach. We used MuJoCo [34] simulation software for all our experiments.

## 6.1 Manipulation Tasks

The environments are similar to multi-goal environments proposed in [25]. We made this environment to make the results comparable with other works who use more famous OpenAI environments. Another reason for choosing baxter as our robot is due to its availability of platform in our lab for performing transfer to real world experiments which is left as a future work.

In all the environment, actions are 4-dimensional with first 3-dimension as the Cartesian motion of the end-effector and last dimension controls the opening and closing of the gripper. We apply the same action 75 simulation steps(with $\Delta t =$



(A) Pick and move task (B) Put cube inside container

(C) Put red cube over purple cube (D) Take cube out of container

FIGURE 6.1: Suite of robot manipulation tasks with baxter robot with end-effector control and parallel jaw gripper.

0.002). The state space includes the Cartesian position of the end-effector, its velocities, position and velocity of gripper, objects pose and its velocities.

We use a single starting state in which object is grasped. The state space includes the relative position of object which has to be transported in then environment. The goal location is given as a 3D Cartesian location where the object has to be transported to.

### 6.1.1 Pick and move object to a target location in 3D

This environment is a close replica of [25] FetchPickAndPlace environment made for baxter robot. The objective to move the object from arbitrary location in the workspace to the 3D target location in the workspace. The starting location of the gripper and object are sampled randomly in the workspace of the robot. The reward function is binary, i.e. the agent obtains reward 0 when the object is within the tolerance 3cm and -1 otherwise. The pictorial representation of then environment is shown in Figure 6.1a

### 6.1.2 Put object A inside container B

The objective is to go, grab the object from any arbitrary location in the workspace and move it inside the container. The goal is specified as the center of the container which is kept fixed across all the episodes. The binary reward function gives the agent 0 if the object is within 5cm of the target location and z of the object is less 2cm from the ground and -1 otherwise. The pictorial representation of then environment is shown in Figure 6.1b

### 6.1.3 Put object A on object B

The objective is to go, grab the object from random location and put it over another object B, whose location is fixed. The goal is specified as the center of the top surface of the object B. The binary reward function gives the agent 0 if the object is within 3cm of the target location and object A is in contact of object B and -1 otherwise. The pictorial representation of then environment is shown in Figure 6.1c

### 6.1.4 Take object A out of container B

The objective is to take reach to the object (random starting location inside the container), grab it and move it out of the container B. The goal is specified as a 3D location (randomly sampled) on the top of the container. The agent gets reward 0 if the object is within 3cm of the target location and -1 otherwise. The pictorial representation of then environment is shown in Figure 6.1d

## 6.2 Implementation Details

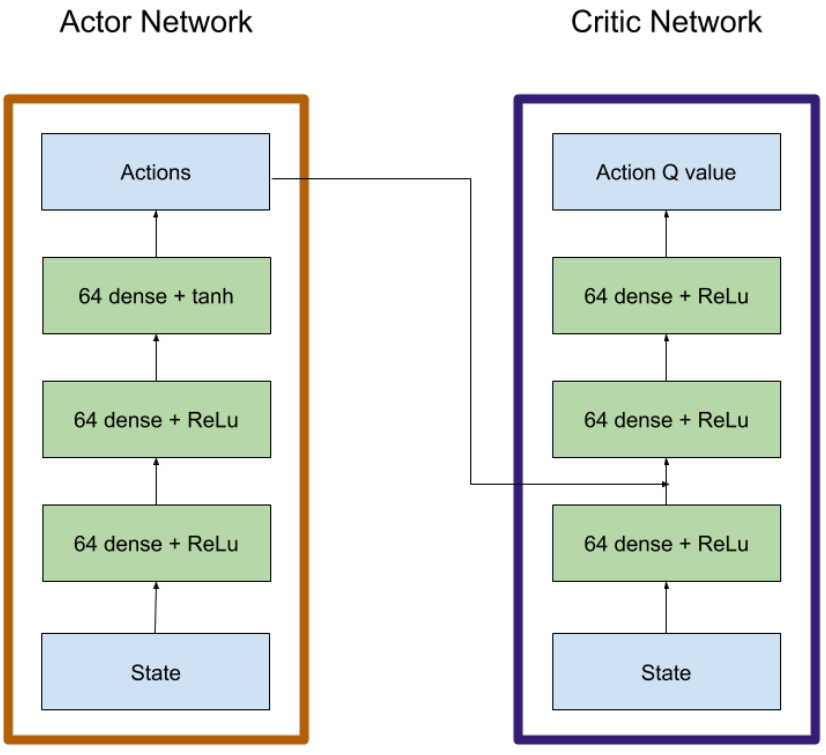



FIGURE 6.2: **Neural Network Structure:**The actor network takes as input the state and outputs actions between [-1,1] which is then remapped to the range corresponding to each dimension in action space. The critic predicts expected discounted future returns if action is executed in state

For representing all the actor and critic we used a neural network with 3 fully connected layers with 64 neurons each. For our methods and baselines we trained DDPG and build over [29]. The complete architecture is shown in Figure 6.2.

| **Hyperparameter** | **Value** |
|---|---|
| Soft target network update parameter $\tau$ | $10^{-3}$ |
| Optimizer | Adam [35] |
| Batch Size | 128 |
| Actor Learning rate | $10^{-4}$ |
| Critic Learning rate | $10^{-3}$ |
| Replay buffer size | $10^{6}$ |
| Discount factor $\gamma$ | 0.98 |
| Uncorrelated noise $\sigma$ | 0.2 |

TABLE 6.1: Hyperparameters for experiments

We tried variation in learning rate and neural network based on related works on DDPG. However it doesn't lead to faster or better performance.

## 6.3 Results

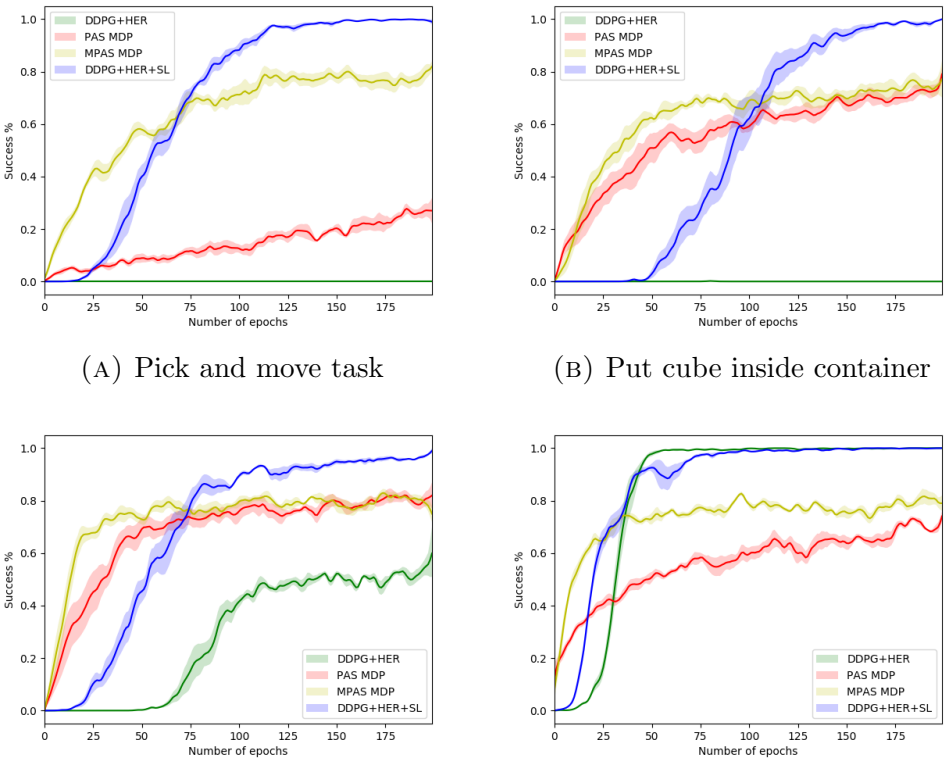
*Baseline 1 (DDPG+HER):* In this baseline, we trained a single DDPG agent from scratch as was done in previous work in [1].

*Baseline 2 (PAS MDP):* In this baseline, we trained a parameterized action space DDPG agent over our fixed skill set as was done in previous work in [21].

*Approach 1 (MPAS MDP):* This is the approach described in Section 4 which is an extension of parameterized action to multi-goal setting.

*Approach 2 (DDPG+HER+SL):* This is the approach described in Section 5 in which we used our look-ahead search exploration using Skill Library and learned DDPG with lower-level actions.

The success percentage plot for all the environments in our test suite are shown in Figure 6.3. As expected for all the environments learning using normal DDPG +




(A) Pick and move task

(B) Put cube inside container

(C) Put red cube over purple cube

(D) Take cube out of container

FIGURE 6.3: The success plot for each manipulation task in our suite. For evaluation, we freeze the current policy and sample 20 random starts and goals at each epoch (1 epoch = 16 episodes of environment interaction).

HER (baseline 1) succeeds only for simple tasks like take cube out of container and is slower to show improvements than any other method. Learning with temporal abstractions using parameterized action space without hindsight experience replay [21] is able to learn complex tasks like Picking and moving object to a target location (Figure 6.3a) and Picking object & placing it inside container (Figure 6.3b). With improvements proposed in Chapter 4 over parameterized action space we are able learn faster as multi-goal setting provides automatic curriculum and learning signals even when the agent is not able to desired goal. As shown in Figure 6.3d, approach 1 outperforms baseline 2.

However as we can see from Figure 6.3c, even when all the methods can learn the policies trained using parameterized action space converge to sub-optimal policy

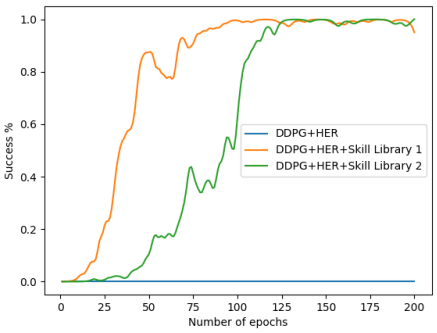
and leads to success only about 80% of the time during evaluation. This brings us to our approach 2, as proposed in Chapter 5, in which we let the possible policy set to be defined over low level actions and use look-ahead search to explore efficiently in the large state space. As shown in Figure 6.3a and Figure 6.3b, the exploring at coarse level and doing look-ahead leads to success. While exploring randomly doesn't succeed at all for complex manipulation tasks. Similar results are shown for Picking and putting object inside container tasks in which approach 2 learns and converges to optimal policy as compared to DDPG+HER which fails completely to achieve any success.

### 6.3.1 Reduced Skill Library

There might be situations in which we might not have the complete set of skills in our skill library. Therefore we did experiments with skill library which consists of fewer skills. We call Skill Library 1 consisting of transit, grasp and transfer. We also construct two other libraries, Skill Library 2 consisting of {grasp, transfer} and Skill Library 3 consisting of {transit, transfer}.

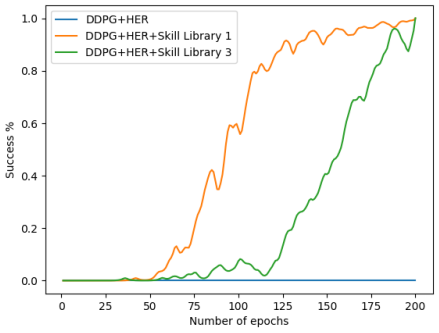
In the first experiment we used Skill Library 2 which consists of grasping and transfer skill only and tested on 2 environments Pick'n'move and Put cube inside container tasks. As shown in Figure 6.4a, exploration with all the skills (Skill Library 1) shows faster convergence than Skill Library 2. This is due not having transit skill causes agent to spend sometime learning the transit skill using interactions generated by other skills and current policy. However we didn't see the slower convergence for Put cube inside container task, as seen in Figure 6.4b

We also tried with another Skill Library 3 for Put cube inside container task in which we don't have grasping skill. The results are shown in Figure 6.4c. Convergence with Skill Library 3 is slower than Skill Library 1 showing that grasping is necessary for this tasks.




(A) Pick and move task


(B) Put cube inside container

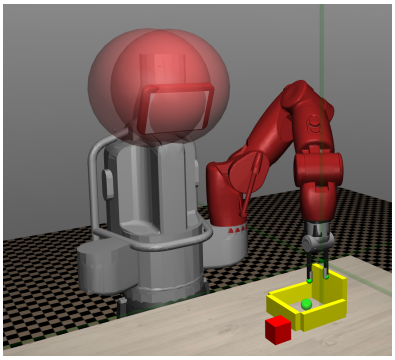



(C) Put cube inside container

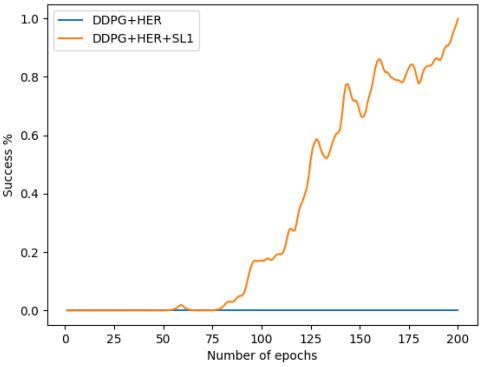
FIGURE 6.4: The success plot for Pick'n'move task and Put cube inside container task. The plots show the effect of having different skills in the Skill Library. Skill Library 1 = {transit, grasp, transfer}, Skill Library 2 = {grasp, transfer} and Skill Library 3 = {transit, transfer}. The assertion is that if we have limited Skill Library, with some essential skills missing, than the speed of convergence will be slower than having all the essential skills. However it is still better than random exploration.(1 epoch = 16 episodes of environment interaction).

### 6.3.2 Affect of Large Unmodelled Dynamics in Environment

One other question is related to the coarse dynamics model learned for each skill. What if that model is wrong? We do not assume anything the test task while learning skills. Therefore skill coarse dynamics model could be wrong. To test this we created a perturbation in our Put cube inside container task by making 1 wall insurmountable by the robot arm. The environment is shown in Figure



(A) Put cube inside container with 1 insurmountable wall




(B) Success Curve

FIGURE 6.5: This Figure is used to test the approach if the coarse dynamics model of the skill is bad in the current test task to be learned. In (A) we show the perturbed version of Put cube inside container in which 1 wall is very high and can't 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).

6.5a. Therefore the skill dynamics involving end-effector states through the large wall would be completely wrong. However using Skill Library is still beneficial as seen in Figure 6.5b and shows to perform the task in which DDPG+HER with random exploration completely fails. We used Skill Library with all the learned skills namely, transit, transfer and grasping.

# Chapter 7

# Conclusion and Future Work

## 7.1 Summary

Leveraging prior experience and long range planning are some of the hottest open problems in today's reinforcement learning. In this thesis, we explore the problem of learning using previously learned low-level skills. We give framework for learning skill library and show its benefit in learning in 2 proposed approaches. Given the Skill library, we explore the learning multi-goal parameterized action space meta-controller and exploration with low level policies.

In this thesis, we first formalize the necessary skill library framework and discuss how various components could be put together for robotic manipulation. We show the generalization of the skill library framework and learned skill library for manipulation to perform multiple long-horizon complex real-world manipulation tasks. The second question tackled is that learning with goal-parameterized skills library as primitive action can be more efficient than learning low-level controller and could be used solve tasks which were not possible before. We then developed a new approach for exploration which uses the benefits of temporal abstraction and

achieves optimal performance. The third part used skill library and coarse dynamics model to perform look-ahead search at skill level and used the suggested skill to perform exploration in the environment to achieve meaningful experience in the environment. Again using this approach, we are able to outperform state-of-the-art methods in model-free learning on our suite of real-world robotic manipulation environments. The last method is particularly interesting as it allows to change the skill library arbitrarily while learning which is not possible in parameterized action space learning. Our exploration method is orthogonal learning algorithm and thus could easily be combined with advances in off-policy model-free policy learning algorithms. Finally, we believe that with reuse of previously learned skills and potential look-ahead search benefits could lead to adaptation of reinforcement learning in robotics for tasks which require long-horizon sequences of action to achieve success.

## 7.2 Future Work

There are multiple interesting directions for extending the currently proposed approaches: multi-goal parameterized action space and exploration using look-ahead search. It would be interesting to explore the combination of the approaches. One of the most interesting direction, in terms of parameterized action space, could be to explicitly introduce a soft action selection step in the actor and then feed it into the critic for calculating the gradients for learning. Other interesting directions could be different architectural choices like feeding actions into first layer with state to critic as done in [36] and/or predicting Q values for each action and propagating gradients back only for the action which was chosen[37].

Other useful direction connected to our second approach is to unfold the search tree based more informed sampling using concepts of entities [38] present in the environment. Another interesting direction could be reuse of previously built

search tree either by bootstrapping Q-value to allow longer unfolding [39] or by keeping statistics of actions at each node in the search tree [40].

# Bibliography


[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[3] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

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

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[7] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. *arXiv preprint arXiv:1803.05580*, 2018.

[8] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2750–2759, 2017.

[9] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[10] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *International Symposium on Experimental Robotics*, pages 173–184. Springer, 2016.

[11] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[12] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[13] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[14] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

[15] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015.

[16] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[17] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[18] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303, 2000.

[19] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[20] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[21] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[22] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. 2016.

[23] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

[24] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494, 2016.

[25] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[26] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[27] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[28] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

[29] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

[30] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[31] Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

[32] Maciej Klimek, Henryk Michalewski, Piotr Mi, et al. Hierarchical reinforcement learning with parameters. In *Conference on Robot Learning*, pages 301–313, 2017.

[33] Devin Schwab and Soumya Ray. Offline reinforcement learning with task hierarchies. *Machine Learning*, 106(9-10):1569–1598, 2017.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[37] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Yang Zheng, Lei Han, Haobo Fu, Xiangru Lian, Carson Eisenach, Haichuan Yang, Emmanuel Ekwedike, Bei Peng, Haoyue Gao, Tong Zhang, Ji Liu, and Han Liu. Parameterized deep q-networks learning: Playing online battle arena with discrete-continuous hybrid action space, 2018. URL https://openreview.net/forum?id=Sy_MK3lAZ.

[38] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*, 2017.

[39] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[40] Timothy Yee, Viliam Lisỳ, and Michael H Bowling. Monte carlo tree search in continuous action spaces with execution uncertainty. In *IJCAI*, pages 690–697, 2016.