Learning to Solve Real-World Physics Puzzles

Alisa Allaire

CMU-RI-TR-23-21

May 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Christopher Atkeson, *chair*
Oliver Kroemer
David Held
Leonid Keselman

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Tasks involving locally unstable or discontinuous dynamics (such as bifurcations and collisions) remain challenging in robotics, because small variations in the environment can have a significant impact on task outcomes. In this thesis, we present a robot system that we developed to evaluate learning algorithms on real-world physical problem solving tasks which incorporate these challenges. For such tasks, learning a single deterministic policy that is robust to slight or imperceptible changes in environment state and dynamics is difficult. Learning such a policy from scratch on the real robot can also be prohibitively expensive. We provide a framework for learning structured exploration policies in simulation based on a mixture of experts (MoE) policy representation. We also present a method for efficiently adapting the policy in the real world. The mixture of experts policy is composed of stochastic sub-policies that allow exploration of multiple distinct regions of the action space (or strategies) and a high-level selection policy to guide exploration towards the most promising regions. We demonstrate that representing multiple strategies promotes efficient adaptation in new environments and strategies learned under different dynamics can still provide useful information about where to look for good solutions.

Acknowledgments

I owe many thanks to my advisor, Chris Atkeson, for his support and guidance over the years. Further, I would like to thank the other members of my committee David Held, Oliver Kroemer, and Leonid Keselman. Their feedback and varied perspectives on this work has been invaluable.

I also would like to thank my lab-mates Mrinal Verghese and Arka Chaudhury for many engaging, thought-provoking, and fun conversations.

Funding

This work was supported by the National Science Foundation under Grant IIS-1849287.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Developing intelligent systems with the efficient and flexible physical reasoning capabilities of humans remains one of the greatest challenges in robotics. One limiting factor is that contact is challenging to leverage and controlling the discontinuous behaviors of colliding objects remains incredibly difficult [21, 26]. Impacts among robots and their surroundings are particularly difficult to model because slight inaccuracies in either initial conditions or model parameters can generate vastly different predictions, even over a small time horizon [4]. These properties can cause significant problems for controlling real-world systems where noisy sensor measurements produce small, possibly unobservable, variations in the environment that have a significant impact on task outcomes.

We are inspired by prior work that proposes simulation-based mechanical puzzles as benchmarks for physical reasoning [6, 8]. These puzzles often involve locally unstable and discontinuous dynamics, emphasizing collisions as multiple objects move and interact over extended periods of time. Unlike most tasks addressed using reinforcement learning, actions can only be taken at the start of the task (setting the initial configuration of objects). There is no possibility of further control or re-planning after the objects start to move, so success depends on reasoning about the task outcome based on the initial state. While effective for evaluating general-purpose, long-term physical reasoning, simulation-based benchmarks neglect properties of real-

world systems such as noisy observations and environment stochasticity that make reasoning difficult.

1.2 Thesis Contributions

One contribution of this work is the development of a robot system to enable evaluation of learning algorithms in a real-world marble run environment which incorporates both locally unstable or discontinuous dynamics as well as properties of real-world systems that make reasoning difficult. Allowing actions only at the beginning of the task, marble run tasks are similar to simulation-based physical reasoning benchmarks but also incorporate stochasticity which can cause varying outcomes for even the same initial state. Additionally, while our robot runs hundreds of trials without human intervention, it can take up to a minute to setup and execute a single trial, so learning algorithms evaluated in this domain must perform well under a limited evaluation budget. We provide a framework for learning structured exploration policies in simulation based on a mixture of experts (MoE) policy representation and a method for efficiently adapting the mixture of experts policy in the real world. We demonstrate that representing multiple strategies promotes efficient adaptation in new environments and strategies learned under different dynamics can still provide useful information about where to look for good strategies.

Chapter 2

The Marble Run Environment

In this chapter, we introduce the marble run environment including our procedure for designing and generating the marble run tasks and an overview of the real-world system, shown in Fig. 2.1. To emphasize the challenges associated with this domain, we evaluate how difficult our marble run tasks are to solve in simulation using randomly selected actions. With a small number of human participants, we also asses how difficult marble run tasks are for humans to solve in the real world.

2.1 Related Work

Many environments for learning physical reasoning have been explored, ranging from physics-based computer games [10, 15] to physical reasoning benchmarks [6, 8, 11, 13]. For evaluating real-world physical reasoning capabilities beyond prediction and question answering, the most common application is contact-rich manipulation tasks [3, 5, 29]. In these domains robots usually take actions and receive feedback at every time-step, which allows re-planning throughout a task and reduces the effects of uncertainty or error. The simulation-based physical reasoning environments Tools [6] and PHYRE [8] allow actions only at the start of a task and are effective benchmarks for general-purpose, long-term physical reasoning. Both PHYRE and Tools define tasks similar to marble run tasks, which all require placing an object in a scene so it interacts with other objects to reach a desired goal state. Unlike the tasks in PHYRE and Tools, our real-world marble run tasks incorporate challenges of the real world

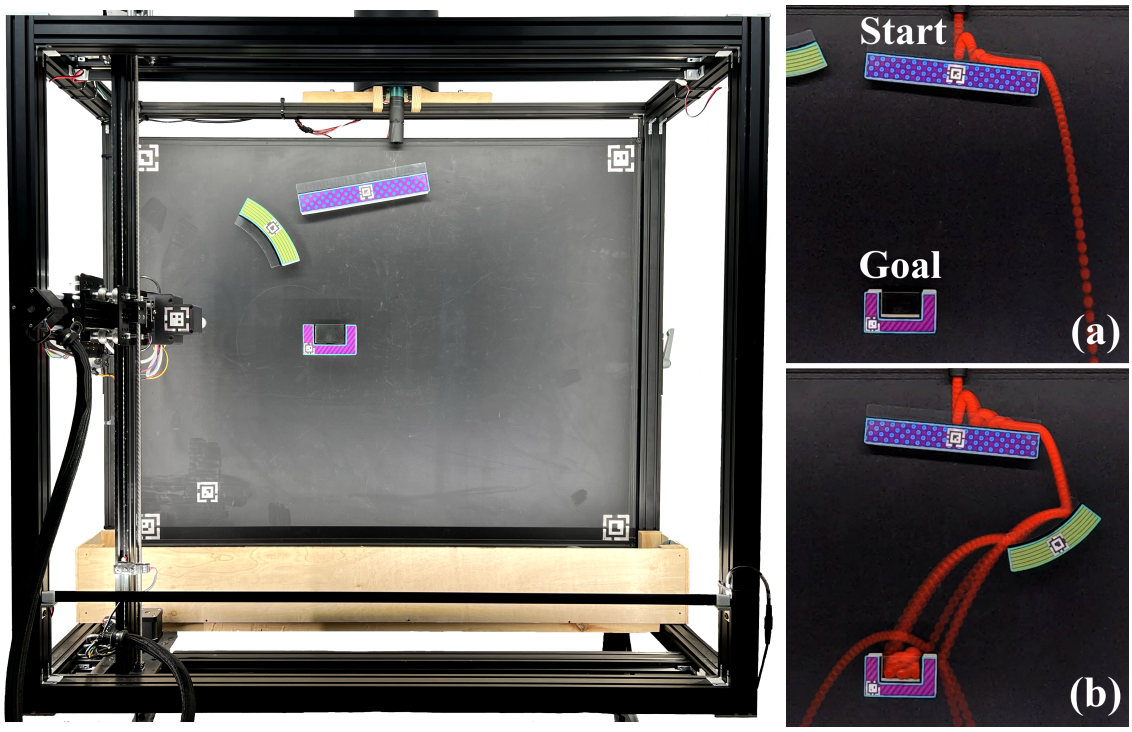


Figure 2.1: The marble run robot autonomously evaluates learning algorithms on real-world marble run tasks. (a-b) An example marble run task (a) and solution (b), where the goal is to place the curved track so the ball lands in the U-shaped goal. Stochasticity of the ball's initial state leads to large variations in outcome for the same action.

including noisy observations and environment stochasticity.

2.2 Task Design and Generation

We focus on the problem of placing an additional track in an existing configuration so that the ball lands in the goal. A marble run task is defined by the initial configuration of all objects in the scene, including the ball, goal, and additional track pieces. In the initial configuration, a ball released at the top of the environment above the rectangular track will miss the goal, as shown in Fig. 2.1a. The robot must then find a configuration of the curved track that allows the ball to land in the goal. An example of one such configuration is shown in Fig. 2.1b.

2.2.1 Problem Definition

The marble run environment frame is defined as a 800 mm × 700 mm square that is co-planar with robot's back panel, as shown in Fig. 2.2b. The full 3D state of the environment is not observable, so we approximate the environment as 2D and ignore small out-of-plane movement. There are not physical barriers to keep the ball within the environment limits, but the ball's position is not tracked outside of the limits. With tracks magnetically attached to a panel, only the ball is considered dynamic.

The state $\mathbf{s}$ is the initial scene configuration, which concatenates each object's x, y position and orientation expressed as $[\sin(\theta), \cos(\theta)]$. Object positions are computed relative to the ball's initial position, which is fixed across tasks and therefore not included in the state. We consider tasks with the same number and types of objects so it is not necessary to include object type in the state. The action $\mathbf{a}$ is the x, y position and orientation of the single moved track piece. If different types of objects could be placed, the object type should also be included in the action. For now, we consider only one object type for the action.

Reward Function

The reward function for a single trial is

$$R(\mathbf{s}, \mathbf{a}) = \begin{cases} 1, & \text{if } \textit{success} \\ -d_{min}, & \text{otherwise} \end{cases} \tag{2.1}$$

where d_{min} is the minimum distance between the ball and the goal along the ball's trajectory. The reward is 1 for successful trials to differentiate between true successes and trials where the ball bounces out of the goal, where $d_{min} = 0$ in both cases. Uncertainty in the real system's initial state can cause the outcome to vary significantly across trials from the same state and action, as in Fig. 2.1b, so the reward is averaged over 6 trials per action taken in the real world or in a stochastic simulator. We empirically found 6 trials provides a good trade-off between minimizing evaluation time and minimizing variance.

2.2.2 Task Generation

We randomly generate marble run tasks with varying initial configurations of a long rectangular track and a U-shaped goal. The objective is to place a curved track so the ball lands in the goal. On the real system, the ball is dropped through a fixed tube so we assume its mean initial position and velocity are the same for every task. The rectangular track is placed near the tube to catch the ball, but varies slightly in x, y, and θ. The goal position is more varied, where the range of x nearly spans the environment width, the range of y spans the region below the rectangular track, and its angle is always 0.

We leverage the task generation scripts from the PHYRE framework [8] to ensure that the sampled tasks (1) have a stable solution (solution that still solves the task if the action is slightly perturbed by $\pm$ 0.5 mm along each axis), (2) are solvable (probability of finding stable solutions among randomly sampled actions exceeds 10^{-5}) and not trivially solved (probability of finding stable solutions is less than 0.5), and (3) have sufficiently diverse solutions (out of the set of possible tasks, tasks with solutions that also solve to over 30% of the other tasks are discarded). We generate a total of 100 tasks that meet these criteria in the simulated environment. The tasks are split into 80 training, 10 validation, and 10 test tasks.

2.3 Real-World System

We designed a custom 4-axis CNC system, which can move in x, y, z, and rotate a suction gripper around z, as the base of our robot. Notably, our system, shown in Fig. 2.2a, consists of low-cost and easily accessible components including the cameras.

2.3.1 Low-Level Control

We use two Raspberry Pi's, each equipped with Protoneer's Raspberry Pi CNC shield, to interface with the CNC stepper motors. The CNC shields handle low-level control of the motors via an on-board Arduino chip running GRBL. GRBL is a open-source software for controlling CNC machines by executing G-code commands read from a file, like a CAD design converted to G-code for 3D printers, or received directly from

another computer, as is the case in our system.

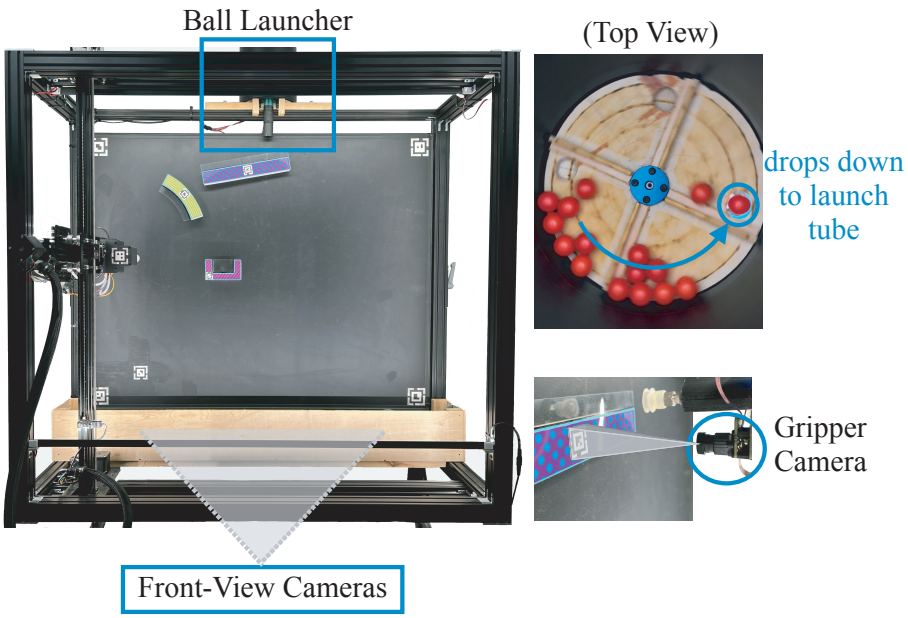
2.3.2 Vision System and Calibration

The vision system consists of two front-view cameras and a gripper mounted camera. One of the front-view cameras and the gripper mounted cameras are Raspberry Pi HQ cameras, which are used to detect and localize the tracks throughout robot operation. The front-view Raspberry Pi cameras supplies images at 1920×1080 resolution and 30 frames per second, while the gripper camera supplies images at 640×480 also at 30 frames per second. We use a GoPro Hero10 as the second front-view camera for high-speed tracking of the ball which records each trial at 4k resolution and 120 frames per second. We scale the video down to 960×540 for post-processing.

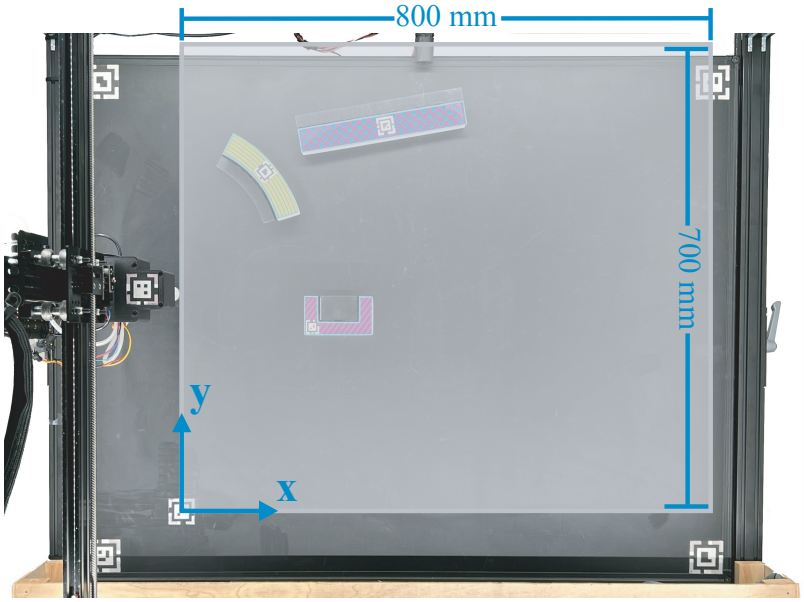
AprilTags [14, 20, 25] mounted on the back panel are used to calibrate the cameras to the coordinate frames of the robot and the marble run environment. The x and y axes of the robot frame are parallel to the x and y axes of the environment, but the robot's frame is offset from the environment frame along the z axis. The 2D environment plane is co-planar with the back panel of the robot. AprilTags marking the four corners of the back panel are used to remove perspective distortion from the camera images, aligning the image planes to the environment plane. The x, y axes of the robot, environment, and image frames are aligned using the 5th AprilTag on the back panel whose center marks the zero point of the environment frame, shown in Fig. 2.2b. Pose estimation with AprilTags is less accurate from farther distances so the front-view Raspberry Pi camera provides an initial estimate of the track poses using the AprilTags located on each track piece. With this initial pose estimate, the robot's gripper can get close enough to a track for its AprilTag to come into view of the gripper camera and a more precise estimate of the track pose is obtained with visual servoing to align the AprilTag to the center of the gripper camera.

2.3.3 Ball Launcher

Balls are held in the bucket mounted directly above the back panel of environment. Balls are dropped one-at-a-time through a tube into to the environment when a hole in the rotating plate at the base of the bucket aligns with the opening of the tube. When the break-beam sensor at the top of the tube detects a ball, the rotating plate



(a) Hardware Setup



(b) Marble Run Environment Frame

Figure 2.2: (a) Hardware setup for the marble run robot. (b) The marble run environment frame is a 800 mm × 700 mm square that is co-planar with robot's back panel.

is stopped so that only one ball is released. The box at the bottom of the back panel collects the majority of balls that would otherwise fall to the floor below the robot. With this design, our robot can run hundreds of trials without human intervention before the bucket runs out of balls and a human must re-fill the bucket. While we only had 500 hundred balls available for our experiments, the bucket can hold closer to 1000 balls at full-capacity.

2.4 Simulation Environment

The simulated marble run environment is built in Box2D (box2d.org) using 2D models of the real marble run tracks extracted from RGB images. The physical parameters (coefficient of restitution, friction, and gravity) are optimized to match data from the real system. The ball's initial position and velocity are assumed fixed but do vary in the real world because the ball's diameter does not match the diameter of the launching tube and its velocity is not explicitly controlled. Noise is added to the ball's initial state to reflect this stochasticity. Specifically, we initialize the ball's initial position and velocity in the simulator by sampling from a normal distribution where the mean and standard deviation are measured from 50 trials on the real system.

For some experiments, we add an additional horizontal gravitation force which acts like wind to represent a shift in dynamics from the training environment to the test environment. We add this force in the same direction that the ball rolls off the rectangular track to prevent cases where wind slows the ball to a stop and the task become unsolvable.

2.5 Evaluation of Marble Run Tasks

2.5.1 Difficulty of Marble Run Tasks in Simulation

We evaluate task difficulty using the stochastic simulation environment by estimating the solution probability for each task as the average success rate of 10,000 randomly sampled actions. The solution probabilities shown in Fig. 2.3a demonstrate varying difficulty across tasks, with some requiring more than 10,000 actions before finding a solution with random sampling alone.

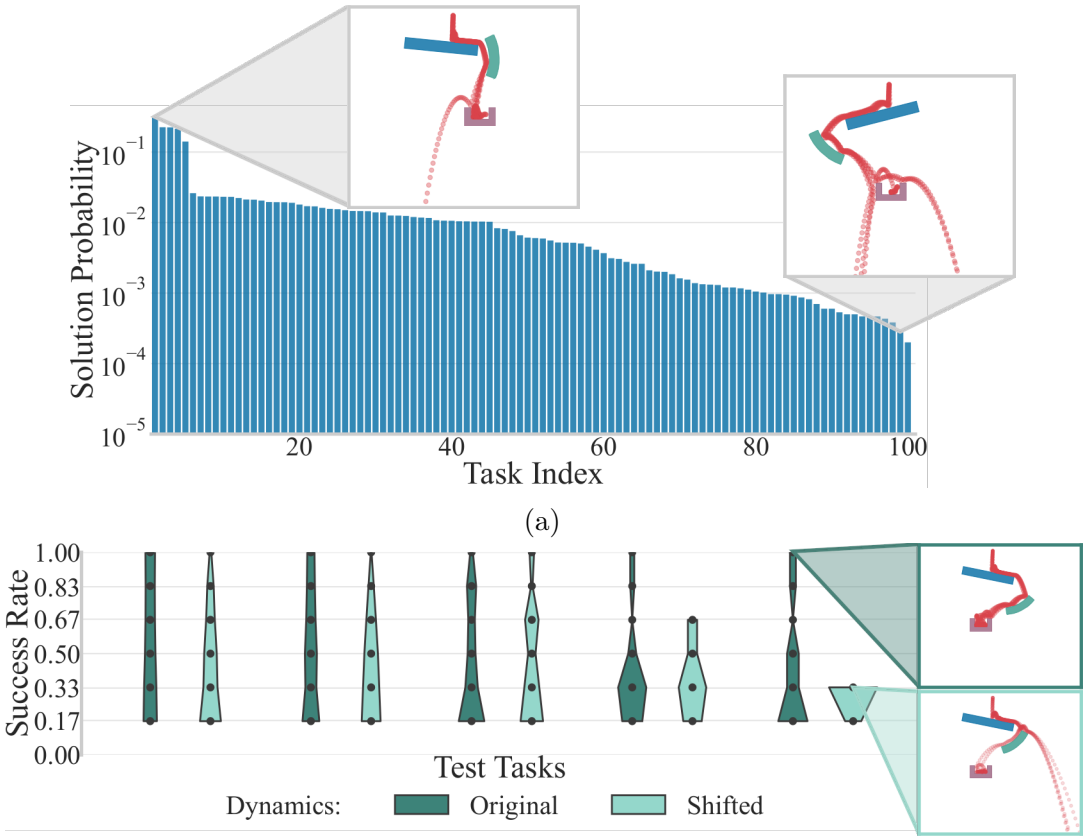


Figure 2.3: (a) Solution probabilities per task estimated over 10,000 random actions in simulation. Tasks increase in difficulty from left to right. (b) Success rate distribution of actions with success rate > 0 out of 10,000 randomly sampled actions. Results are shown for 5 test tasks, sorted in order of increasing difficulty. The width at each point corresponds to the proportion of occurrences with a success rate of that value. Results are also shown for different simulation dynamics (shifted) where we apply a horizontal wind-like force in the same direction the ball rolls off the initial rectangular track. Depending on the environment dynamics, it may be more difficult to find actions with high success rates for some tasks.

The average success rate that can be achieved for each task depends on the environment dynamics and may be considerably less than 1 on difficult tasks, which is shown in Fig. 2.3b. When a wind-like force is introduced in the environment, lower success rates under shifted dynamics indicate some tasks are more difficult to solve. If finding actions with success rates of 1 is possible, such actions may be rare, difficult

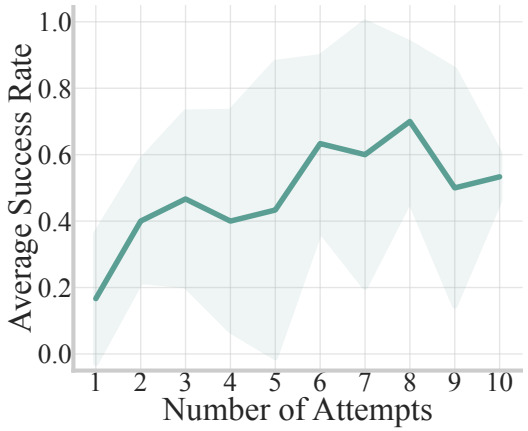


Figure 2.4: Average performance of 5 humans on a marble run task over 10 attempts.

to reproduce, or occur by chance. Estimating the success rate with more than 6 trials per action would reduce the occurrence of finding high success rate actions by chance, but with increased run-time.

2.5.2 Human Performance on Real-World Marble Run Tasks

While this task appears easy given that many humans could find a solution within just a few attempts, Fig. 2.1b shows that, due to slight variations in the environment, an action that is successful once may not be successful every time. Preliminary experiments demonstrate that even for a simple task, finding solutions that are robust to stochasticity in the environment is challenging even for humans. Fig. 2.4 shows the average performance of 5 humans who were asked to make 10 attempts to solve a marble run task. Initially, all participants were able to find actions that were somewhat successful. Small adjustments to the initial object placement usually produced slightly improved performance, but nearly all participants eventually required switching strategies with more drastic changes to the object placement or angle in order to find more robust actions. We task the robot with finding actions that are *always* or *nearly always* successful, rather than just finding actions that worked once.

Chapter 3

Learning Exploration Strategies from Simulated Experiences

Due to small changes having major effects on task outcome and the real time duration of marble run tasks, we focus on learning a structured exploration policy in simulation that can be efficiently adapted in the real world. Directly learning a deterministic policy is difficult due to this parameter sensitivity and the sim-to-real gap. We choose a policy representation that is stochastic to support exploration and captures multiple types of solutions in case the strategy that is optimal in simulation is not applicable in the real world. Specifically, we use a mixture of experts (MoE) policy representation that is composed of multiple Gaussian sub-policies and a high-level selection policy and represents multiple strategies to achieve the same or similar goals. Our proposed approach extends advantage-weighted regression [16, 22] to train a mixture of experts policy from simulated experiences. While we do not expect the mixture policy trained offline in simulation to transfer perfectly to the real world, it should provide a good starting point to perform an online search for solutions, which is described in more detail in Chapter 4.

3.1 Related Work

We extend advantage-weighted regression (AWR) to mixture of expert policies. AWR formulates a constrained policy search problem as weighted supervised regression

on the actions, allowing the policy to be easily updated with both online data and offline data [16, 22]. An earlier instantiation of this framework incorporated a similar advantage-weighted policy update [18]. Relative entropy policy search (REPS) [23] and maximum a posteriori policy optimization (MPO) [1] are closely related and similarly derived as a constrained policy search, but using the dual formulation for optimizing constrained objective functions.

Policy hierarchies in robot learning are often represented as options, which are temporally extended actions [7, 24]. In our work, actions are only applied at the start of the task and options become a set of initial actions which we represent as a mixture of experts [28]. Hierarchical extensions to both REPS and MPO have been developed [9, 27] and our approach is similar reflecting the underlying similarities between REPS, MPO, and AWR. They focus on learning hierarchies incrementally from scratch which is hard and requires imposing additional constraints to learn distinct and diverse sub-policies. We formulate the problem in a simpler way by assuming access to a datatset of prior experiences to initialize the mixture policy using batched supervised regression. We show that a simulator generates useful training data in this domain to learn the mixture distribution's underlying structure, while online learning adapts the sub-policies and distribution over policies to a specific task or environment.

3.2 Mixture of Experts (MoE) Policy Representation

To represent knowledge learned from past experiences, we learn a stochastic policy parameterized as a probabilistic mixture of experts (MoE) that maps an input state to a multi-modal distribution of actions. The MoE consists of K Gaussian "expert" policies $\{\pi_k\}_{\forall k \in \{1,\dots,K\}}$ and a "gating" policy ψ that predicts a categorical distribution over the experts such that

$$k \sim \text{Categorical}(\psi(k|\mathbf{s}, \boldsymbol{\phi})) \tag{3.1}$$

$$\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k), \tag{3.2}$$

where $\boldsymbol{\phi}$ and $\boldsymbol{\theta}_k$ are learned parameters for the neural networks representing ψ and π_k. k is the expert model index sampled from the categorical distribution predicted by ψ and $\mathbf{a}$ is the action sampled from the Gaussian distribution with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$ represented by θ_k.

3.3 Expectation Maximization for Mixture of Experts Policies

We derive a training procedure from the expectation maximization (EM) algorithm [19]. The EM algorithm estimates the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta} = \{\boldsymbol{\theta}_k\}_{k \in 1:K}$ that maximize the complete log-likelihood of the selection variables $\{z_k\}$ and actions $\mathbf{a}$ given the state $\mathbf{s}$. The selection variable z_k is 1 if the k^{th} expert generates or predicts the action $\mathbf{a}$ and 0 otherwise. The E-step calculates the expected log-likelihood, given as $J(\boldsymbol{\phi}, \boldsymbol{\theta})$ below, where w'_k is the probability z_k is one given the state and action.

$$J(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a} \sim \mathcal{D}} \left[\sum_{k=1}^{K} w'_k \Big(\log \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k) + \log \psi_k(\mathbf{s}, \boldsymbol{\phi}) \Big) \right] \tag{3.3}$$

$$w'_k = P(z_k = 1|\mathbf{s}, \mathbf{a}) = \frac{\psi_k(\mathbf{s}, \boldsymbol{\phi}') \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}'_k)}{\sum_{j=1}^{K} \psi_j(\mathbf{s}, \boldsymbol{\phi}') \pi_j(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}'_j)} \tag{3.4}$$

The superscript $'$ indicates w_k is computed using the current parameter estimates and is not involved in the gradient calculation. In the standard EM algorithm, the M-step analytically computes parameters which maximize $J(\boldsymbol{\phi}, \boldsymbol{\theta})$. There is not a closed-form solution when ψ and π_k are neural networks with non-linear activations as in this work so we instead use a generalized EM algorithm, where the M-step performs a gradient step to move $J(\boldsymbol{\phi}, \boldsymbol{\theta})$ closer to the maximum [17].

3.4 Advantage-Weighted Regression for Mixture of Experts Policies

Using the mixture log-likelihood in (3.3), we apply advantage-weighted regression (AWR), which weights the log-likelihood with the exponential advantage $\exp(\frac{A^{\pi}(\mathbf{s},\mathbf{a})}{\eta})$ [16, 18, 22]. The advantage $A^{\pi}(\mathbf{s},\mathbf{a}) = R(\mathbf{s},\mathbf{a}) - V^{\pi}(\mathbf{s})$ is a measure of improvement based on how the reward of an action compares to the average reward observed from a state under the current policy. When the advantage is negative, the weights approach zero and filter out poorly performing actions. The resulting advantage-weighted objective function is

$$J(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\sum_{k=1}^{K} w'_k \exp\left(\frac{A^{\pi'_k}(\mathbf{s},\mathbf{a})}{\eta}\right)\Big(\log \pi_k(\mathbf{a}|\mathbf{s},\boldsymbol{\theta}_k) + \log \psi_k(\mathbf{s},\boldsymbol{\phi})\Big)\right], \quad (3.5)$$

where η is a Lagrange multiplier associated with constraining the policy to stay close to the behavior policy π_β that represents the distribution of data seen so far. Dual gradient descent can be used to estimate η, but this requires estimating π_β from the data [27]. We treat η as a fixed hyperparameter, which has been effective in prior work [16, 22]. In marble run tasks, the final episode reward is observed immediately after taking each action, so the value function is the average reward of actions sampled from the current policy at a specified state. During offline training, we pre-compute the per-state values as the average reward of actions observed in the dataset at each state.

3.5 Summary of Offline Learning Algorithm

We generate a dataset of prior experiences using the simulated marble run environment by randomly sampling actions on a set of training tasks until 500 successful and unsuccessful actions are found for each task. Each sample is stored in the dataset $\mathcal{D}$ as a state-action-reward tuple $(\mathbf{s}, \mathbf{a}, R(\mathbf{s},\mathbf{a}))$. In algorithm 1, we summarize the procedure to train the mixture of experts policy from this dataset of simulated experiments.

Algorithm 1: Offline Learning from Simuated Experiments

Input: Offline dataset $\mathcal{D}$

// Train MoE policy

for each epoch **do**

  for batch in $\mathcal{D}$ **do**

    // Do soft policy update (Eq. 3.5)

    for each policy k **do**

      $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\phi}, \boldsymbol{\theta})$

    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha \nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}, \boldsymbol{\theta})$

// Sort samples into subsets according to w'_k (Eq. 3.4)

$\{\mathcal{D}_1, \ldots, \mathcal{D}_K\} \leftarrow \mathcal{D}$

// Train reward functions

for each epoch **do**

  for each policy k **do**

    for batch in $\mathcal{D}_k$ **do**

      $\boldsymbol{\omega}_k \leftarrow \boldsymbol{\omega}_k - \alpha \nabla_{\boldsymbol{\omega}_k} \mathcal{L}_{\text{MSE}}(\boldsymbol{\omega}_k)$

Return: $\boldsymbol{\phi}$, $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$, $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_K\}$, $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$

Chapter 4

Adapting Exploration Strategies to the Real World with Online Learning

The mixture policy trained offline in simulation represents prior knowledge about what strategies might work well in different contexts. When new environments or task configurations are encountered, the offline policy is a starting point from which to perform an online search for robust solutions in the current context. The expert policies represent promising regions of the action space to explore and the gating network, which is responsible for selecting which expert policy to use, directs the search towards high-reward regions. Our experiments show that online learning successfully fine-tunes the mixture of experts policy within a few dozen attempts in the real world and even exceeds human performance on a test task. Our results demonstrate that representing multiple strategies promotes efficient adaptation in new environments. Strategies learned in simulation or under different dynamics can still provide useful information about where to look for solutions.

4.1 Decomposing the Objective Function with Hard Policy Updates

As a supervised learning algorithm, advantage-weighted regression is easily adapted to online learning by incorporating online samples into policy updates. The objective function defined in (3.5) requires indiscriminately optimizing all sub-policies over batches sampled across the entire dataset with each update step, where samples are weighted according to responsibilities w'_k. This soft policy update shares information between policies and is important during early stages of training to learn a better division of the state space. We empirically observe that after pre-training the above soft updates become less effective during online learning. We assume this is because a locally optimal division of the state space is learned during pre-training.

To compensate for the declining effectiveness of soft updates, we perform hard policy updates during online learning by updating expert policies independently using only data associated with each policy. Performing policy updates using only the most relevant samples helps sub-policies quickly specialize to the current task.

The individual update rules can be derived by setting w'_k to 0 or 1 in (3.5), where 1 is assigned to the component with the highest probability $P(z_k = 1|\mathbf{s}, \mathbf{a})$. We decompose (3.5) into separate objective functions for the gating policy and each expert policy. The objective function for policy k is

$$J(\boldsymbol{\theta}_k) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}_k}\Big[\exp\left(\frac{A^{\pi'_k}(\mathbf{s},\mathbf{a})}{\eta}\right)\log\pi_k(\mathbf{a}|\mathbf{s},\boldsymbol{\theta}_k)\Big], \tag{4.1}$$

where $\mathcal{D}_k$ is a subset of the dataset generated by or associated with the k^{th} policy. We also decompose the advantage function. The advantage function for the k^{th} policy is defined as $A^{\pi'_k}(\mathbf{s},\mathbf{a}) = R_{\boldsymbol{\omega}_k}(\mathbf{s},\mathbf{a}) - \mathbb{E}_{\mathbf{a}\sim\pi'_k}[R_{\boldsymbol{\omega}_k}(\mathbf{s},\mathbf{a})]$. Similarly, the gating policy's objective function is

$$J(\boldsymbol{\psi}) = \mathop{\mathbb{E}}_{\mathbf{s},k\sim\mathcal{D}}\Big[\exp\left(\frac{A^{\psi'}(\mathbf{s},k)}{\eta}\right)\log\psi_k(\mathbf{s},\boldsymbol{\phi})\Big], \tag{4.2}$$

and its advantage function is $A^{\psi'}(\mathbf{s},k) = \mathbb{E}_{\mathbf{a}\sim\pi'_k}[R_{\boldsymbol{\omega}_k}(\mathbf{s},\mathbf{a})] - \mathbb{E}_{k\sim\psi'}[\mathbb{E}_{\mathbf{a}\sim\pi'_k}[R_{\boldsymbol{\omega}_k}(\mathbf{s},\mathbf{a})]]$. The gating policy's advantage function provides an estimate of how actions from

expert k compare to other experts. A similar advantage function is defined in hierarchical reinforcement learning as the advantage over options for determining option termination criteria [7].

4.2 Learning an Approximate Reward Function

During online learning, the advantage is estimated using learned reward functions. Learning a single function to approximate a multi-modal, discontinuous reward function is difficult, so different learned reward functions are associated with each expert policy. Each approximate reward function $R_{\boldsymbol{\omega}_k}(\mathbf{s}, \mathbf{a})$ is trained using only data generated by the associated expert. During offline training of the policy, the best divisions of the state-action space are not yet known, so we estimate the advantage directly from sampled rewards to avoid bias. After the mixture policy is trained offline, we perform a hard assignment of samples to the most likely policy indicated by responsibilities w'_k and $R_{\boldsymbol{\omega}_k}(\mathbf{s}, \mathbf{a})$ is pre-trained over the corresponding subset of data, $\mathcal{D}_k$, by minimizing the mean-squared-error (MSE) between predicted and observed rewards. An equal number of positive and negative samples is used in each update for both pre-training the reward functions and during online learning. Samples are considered positive if at least half the trials for the action are successful (i.e. success rate ≥ 0.5). During online learning, each reward function is updated using both online and offline samples from the corresponding policy.

4.3 Summary of Online Learning Algorithm

We assume the online learning phase for each new task is initialized with the same offline policy and dataset. At each iteration of online learning, an expert policy and action are sampled from the mixture policy ($k \sim \psi,\ \mathbf{a} \sim \pi_k$). The action is executed and stored in the dataset as a tuple $(\mathbf{s}, \mathbf{a}, k, R(\mathbf{s}, \mathbf{a}))$. The learned reward function $R_{\boldsymbol{\omega}_k}(\mathbf{s}, \mathbf{a})$ associated with the current expert k is then updated with a batch of training points $(\mathbf{s}_k, \mathbf{a}_k, R(\mathbf{s}_k, \mathbf{a}_k)) \sim \mathcal{D}_k$ containing an even mixture of positive and negative samples. The policy π_k is updated with a batch of training points $(\mathbf{s}_k, \mathbf{a}_k, R(\mathbf{s}_k, \mathbf{a}_k)) \sim \mathcal{D}_k$ using (4.1) and the gating policy ψ is updated using (4.2)

with a batch of training points $(\mathbf{s}, \mathbf{a}, k, R(\mathbf{s}, \mathbf{a})) \sim \mathcal{D}$ sampled over the entire dataset.

Algorithm 2: Online Learning

Input: Pretrained parameters $\boldsymbol{\phi}$, $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$, $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_K\}$, and offline dataset $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$

for each action attempt **do**

Sample policy $k \sim \text{Categorical}(\psi(k|\mathbf{s}, \boldsymbol{\phi}))$

Sample action $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k)$,

Store new sample $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{(\mathbf{s}, \mathbf{a}, R(\mathbf{s}, \mathbf{a}))\}$

`// Update reward function` k

for each update step **do**

Sample batch from $\mathcal{D}_k$ `// online and offline samples`

$\boldsymbol{\omega}_k \leftarrow \boldsymbol{\omega}_k - \alpha \nabla_{\boldsymbol{\omega}_k} \mathcal{L}_{\text{MSE}}(\boldsymbol{\omega}_k)$

`// Update policy` k

for each update step **do**

Sample batch from $\mathcal{D}_k$ `// online and offline samples`

$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\theta}_k)$ `// from Eq.` 4.1

`// Update gating network`

for each update step **do**

Sample batch from complete dataset $\mathcal{D}$ `// online and offline samples`

$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha \nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi})$ `// from Eq.` 4.2

Return: $\boldsymbol{\phi}$, $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$, $\{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_K\}$, $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$

4.4 Implementation Details

Network Architecture

The sub-policies, gating network, and reward function are all represented as multi-layer perceptions (MLP) with 2 hidden layers, where each layer has 256 units and is followed by ReLU activations. For the sub-policies, the output layer splits into two heads for the mean and covariance. To estimate the covariance, the network outputs Cholesky factors A such that $\Sigma = AA^T$, where A is lower triangular and the diagonals of A are $A_{ii} \leftarrow \exp(A_{ii}) + \epsilon$. The Gumbel-Softmax activation, which provides a differentiable approximation of samples drawn from a categorical distribution [12], is applied to the output layer of the gating network. All networks are optimized using the Adam optimizer.

Gumbel-Softmax

The Gumbel-Softmax activation has a temperature parameter τ that adjusts how closely the expected value of the samples from the Gumbel-Softmax approximate samples from the categorical distribution. As $\tau \to \infty$, the expected value of the Gumbel-Softmax distribution approaches a uniform distribution, and the expected value converges to the categorical distribution as $\tau \to \infty$. During learning, there is a trade-off between small temperatures, where samples are closer to one-hot and the variance of the gradients is large, and large temperatures, where samples are closer to uniform and the variance of the gradients is small. It is standard practice to start at a high temperature and gradually decrease the temperature to a small but non-zero value [12]. We initially set τ to 10 and anneal it to 2.5 throughout offline training using the schedule $\tau = \max(2.5, \tau \exp(-rt))$ where t is the current epoch and r controls the annealing rate. During online learning, we keep τ fixed at 2.5.

Advantage Scaling

During offline learning, the advantage scale factor η used for advantage-weighted regression (Eq. 3.5) is set to 0.1. During online learning, we use hard policy updates and decompose the advantage function for the gating network and sub-policy updates,

as shown in Eq.4.2 and 4.1. For the gating network η is set to 0.3 and for sub-policy updates η is 0.1. We also clip the advantage weights to a maximum value of 100 for the sub-policy updates and and 20 for the gating network updates to prevent gradients from exploding.

Ratio of Online-to-Offline Samples

The batches used for the policy and reward function updates are composed of a balanced ratio of online-to-offline samples, providing more aggressive updates than uniform sampling. The ratio is initially set to 0 and linearly increased to 1 over N steps, where each batch contains only online samples by the N^{th} step. For the expert policy and reward function updates, we increase the ratio to 1 over 25 steps. For the gating network, we increase the ratio more slowly over 100 steps to preserve exploration and reduce the risk of premature convergence to a sub-optimal strategy. Additional hyperparameters are provided in Table 4.1.

Parameter	Value
Policy hidden sizes	256-256
Offline learning rate	0.0003
Offline epochs	500
Online learning rate	0.00003
Online update steps (per attempt)	1000
Batch size	1024
Weight decay	0.0001
Number of policies	20
Gating hidden sizes	256-256
Offline learning rate	0.0003
Offline epochs	500
Online learning rate	0.00003
Online update steps (per attempt)	10
Weight decay	0.0001
Batch size	1024
Gumbel softmax temp. (τ)	10
τ anneal rate (r)	0.00003
Reward hidden sizes	256-256
Offline Learning rate	0.0003
Offline epochs	2000
Online Learning rate	0.0003
Online update steps (per attempt)	2000
Batch size	1024
Weight decay	0.0001

Table 4.1: Hyperparameters

4.5 Evaluation

4.5.1 Metrics

The average success rate is used as a performance metric, where the success rate refers to the number of successful trials out of 6 taken for each action. We find aggregating performance across tasks computed using the arithmetic mean can be dominated by outlier tasks (i.e. very easy or very difficult tasks). We use the inter-quartile mean (IQM), which is less sensitive to outliers and stratified bootstrap confidence intervals to report aggregate performance [2].

4.5.2 Method Comparisons

Offline Mixture of Experts [Offline]

We evaluate the mixture of experts policy's performance after training offline on the simulated dataset, as described in Section 3.5. Offline performance is reported as the average success rate of 20 actions sampled from the mixture policy and evaluated on the test tasks. Actions are selected by sampling an expert policy from the categorical distribution over policies (i.e. the gating network) and then using the mean of the sampled policy as the action to evaluate.

Online Mixture of Experts [Online]

The mixture of experts policy is updated with online learning, as described in Section 4.3, by attempting 100 actions and updating the policy after each attempt. Every 5 attempts, we take an additional action to evaluate the mean of the current expert policy and record the performance. These evaluation actions are used only to report performance and are not used in the policy update.

Simulation Performance Baseline [Sim Baseline]

As a performance baseline for simulation-based experiments, we rank 10,000 randomly sampled actions for each test task using a perfect model of the evaluation environment (i.e. the simulator). The average success rate of the top 5 actions ranked by the

model is reported. As the number of sampled actions approaches infinity, the baseline performance would represent the best performance that could possibly be achieved on the simulated test tasks. The baseline's reported performance may be lower than the true best performance because it is limited to ranking 10,000 actions per task.

Single Gaussian Policy [Single]

To emphasize the importance of representing multiple strategies, we compare the MoE policy performance to that of a single Gaussian policy trained using the same procedures as in Sections 3.5 and 4.3, except only a single policy is used so no gating network is learned.

4.5.3 Experiments

Online Learning in Simulation

In Fig. 4.1, we show the average success rate of evaluation steps taken every 5 attempts during online learning in simulation. The bar charts show the per-task success rates averaged over all evaluation steps. When the dynamics of the training and test environments match, the performance gained by representing multiple strategies is less pronounced. The offline performance of the single Gaussian policy falls slightly below the MoE policy, likely due to the single Gaussian policy averaging over multiple solution regions which may include pockets of lower reward regions between them. The MoE policy can represent distinct solution regions as separate policies which allows the policies to fit more closely to the high reward regions and converge more quickly during online learning.

Online Learning in Simulation under Different Dynamics

The benefits of representing multiple strategies are more easily observed when the dynamics of the test environment do not match the dynamics of the training environment, as in Fig. 4.2. At the start of convergence, after around 25 attempts, the MoE continues to increase beyond the performance of the single Gaussian policy. This indicates that the MoE policy is more capable of escaping local optima by switching between different candidate solutions. Escaping local optima is especially important

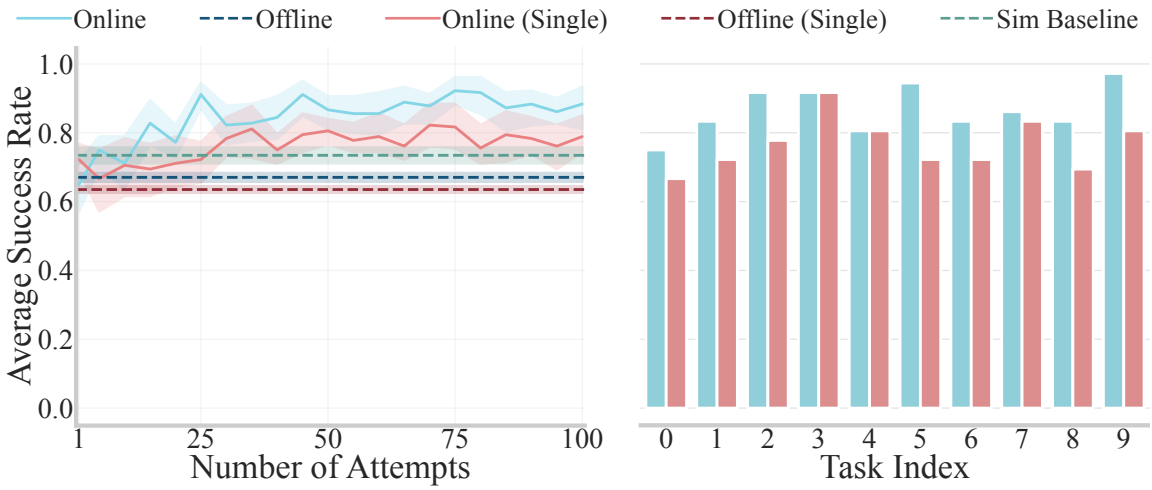


Figure 4.1: Online learning performance in simulation. Average success rate vs. number of attempts (per task) is shown for the evaluation steps taken every 5 attempts during online learning. The bar charts show the average success rate over all evaluation steps for each task.

under different dynamics, because the best strategies for a task will change depending on the dynamics. The relative robustness of the MoE policy to shifts in dynamics is further shown by the offline policy performance which is less affected by the shifted dynamics than the single Gaussian policy.

Online Learning in the Real World

Fig. 4.3a shows offline and online MoE policy performance on the real system, evaluated on a random subset of 5 test tasks. We also limit online learning to just 60 attempts to reduce run-time. Despite the mismatched dynamics between the simulation environment and the real world, the MoE policy achieves an average success rate just over 0.8 within a few dozen attempts, which is consistent with the simulation results. In Fig. 4.3b, we compare the performance of the MoE policy to the human performance from Fig. 2.4 which was evaluated on test task 7. We plot the average performance from the last 5 attempts as the human baseline. The MoE policy starts off with around the same performance as the human baseline, but eventually exceeds human performance. By the end of online learning, the average performance of the MoE policy is hovering between 0.7 and 1 so the asymptotic

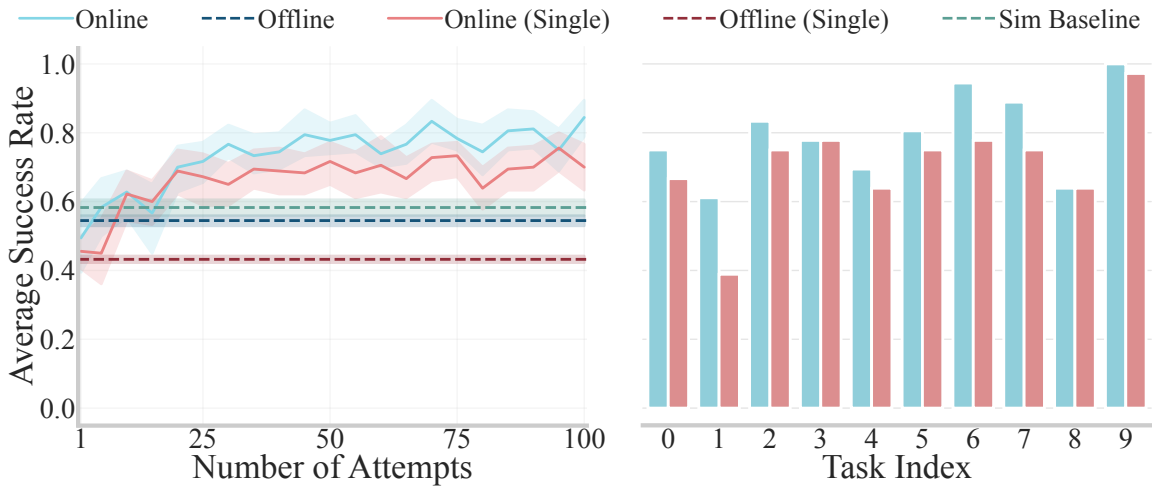


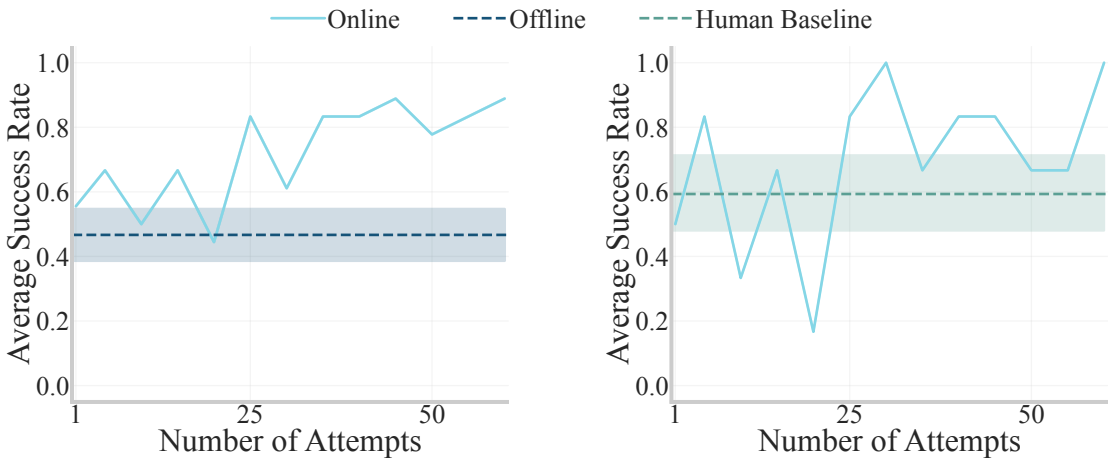
Figure 4.2: Online learning performance in simulation with a wind-like force producing different dynamics than the offline dataset. Average success rate vs. number of attempts (per task) is shown for the evaluation steps taken every 5 attempts during online learning. The bar charts show the average success rate over all evaluation steps for each task.

performance is likely in the range of 0.8-0.9 for that task.

4.6 Conclusion and Future Work

4.6.1 Summary

We present a method using a mixture of experts policy to represent multiple strategies for solving marble run tasks. Our experiments demonstrate that, even when trained offline on simulated data, online learning quickly adapts the policy to solve new marble run tasks in the real world. Ultimately, we expect that achieving human-level physical reasoning will require elements of multi-strategy learning. Finally, by developing a robot system to evaluate the proposed approach on real-world marble run tasks, this work emphasizes the importance of enabling experimental evaluation in domains that involve complex dynamic interactions in the physical world.



(a) Real World (b) Test Task 7

Figure 4.3: (a) Online learning performance in the real world evaluated on a subset of 5 test tasks. (b) Comparison to human performance for a single test task.

4.6.2 Future Work

Although configurations of objects across tasks vary, the marble run tasks are limited in the types and numbers of objects used and only one action is allowed. Future work should consider more complex tasks that involve varying types and numbers of objects and allow multiple sequential actions to be taken. A pixel or graph-based state representation would provide more informative encoding of object shape and support varying numbers of objects are present which should aid in generalization across increasingly complex tasks. Finally, we would like to explore additional domains which incorporate complex, dynamic interactions and thus are similarly sensitive to initial conditions and model parameters. Some examples include footstep planning over challenging terrain or dynamic cooking tasks.

Bibliography

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1ANxQW0b. 3.1

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021. 4.5.1

[3] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL http://arxiv.org/abs/1606.07419. 2.1

[4] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018. 1.1

[5] Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua B Tenenbaum, Alberto Rodriguez, and Leslie P Kaelbling. Combining Physical Simulators and Object-Based Networks for Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. 2.1

[6] Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020. ISSN 0027-8424. URL https://www.pnas.org/content/117/47/29302. 1.1, 2.1

[7] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 3.1, 4.1

[8] Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. In *Advances in*

Neural Information Processing Systems, volume 32, 2019. 1.1, 2.1, 2.2.2

[9] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17 (93):1–50, 2016. URL http://jmlr.org/papers/v17/15-188.html. 3.1

[10] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *ICLR*, 2016. 2.1

[11] Rohit Girdhar and Deva Ramanan. CATER: A diagnostic dataset for compositional actions and temporal reasoning. *CoRR*, abs/1910.04744, 2019. URL http://arxiv.org/abs/1910.04744. 2.1

[12] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2016. URL https://arxiv.org/abs/1611.01144. 4.4, 4.4

[13] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2.1

[14] Maximilian Krogius, Acshi Haggenmiller, and Edwin Olson. Flexible layouts for fiducial tags. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2019. 2.3.2

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. 2.1

[16] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *CoRR*, abs/2006.09359, 2020. URL https://arxiv.org/abs/2006.09359. 3, 3.1, 3.4, 3.4

[17] Radford M. Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, 1998. 3.3

[18] Gerhard Neumann and Jan Peters. Fitted q-iteration by advantage weighted regression. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009. 3.1, 3.4

[19] Shu-Kay Ng and G.J. McLachlan. Using the em algorithm to train neural networks: misconceptions and a new algorithm for multiclass classification. *IEEE Transactions on Neural Networks*, 15(3):738–749, 2004. 3.3

[20] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011. 2.3.2

[21] Mihir Parmar, Mathew Halm, and Michael Posa. Fundamental challenges in deep learning for stiff contact dynamics. *CoRR*, abs/2103.15406, 2021. URL https://arxiv.org/abs/2103.15406. 1.1

[22] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019. 3, 3.1, 3.4, 3.4

[23] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. 3.1

[24] 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. 3.1

[25] John Wang and Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016. 2.3.2

[26] Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. Modeling and control of legged robots. In *Springer handbook of robotics*, pages 1203–1234. Springer, 2016. 1.1

[27] Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Y. Siegel, Nicolas Heess, and Martin A. Riedmiller. Compositional transfer in hierarchical reinforcement learning. In *Robotics Science and Systems*, 2020. URL https://roboticsconference.org/2020/program/papers/54.html. 3.1, 3.4

[28] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193, 2012. 3.1

[29] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. In *Proceedings of Robotics: Science and Systems (RSS)*, 2019. 2.1