Learning Exploration Strategies to Solve Real-World Marble Runs

Alisa Allaire[1] and Christopher G. Atkeson[1]

***Abstract*— Tasks involving complex dynamic interactions remain challenging in robotics, because small variations in the environment can have a significant impact on task outcomes. For such tasks, learning a single deterministic policy that performs well across a variety of tasks and uncertain dynamics is difficult, so we focus on structuring exploration with multiple stochastic policies based on a mixture of experts (MoE) policy representation that can be efficiently adapted in the real world. The MoE 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 develop a robot system to evaluate our approach in a real-world physical problem solving domain. After training the MoE policy in simulation, online learning in the real world demonstrates efficient adaptation within just a few dozen attempts. Our results confirm 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 solutions.**

I. INTRODUCTION

Developing intelligent systems with the efficient and flexible physical reasoning capabilities of humans remains one of the greatest challenges in robotics. Tasks involving highly dynamic interactions between multiple objects are particularly difficult because small, possibly unobservable, variations in the environment can have a significant impact on the task outcomes. Prior works have proposed simulation-based physics puzzles as benchmarks for physical reasoning that emphasize reasoning about complex interactions over extended periods of time by allowing actions to be taken only at the start of a task so actions cannot be re-planned on-the-fly throughout a task [1], [2]. Because success relies on reasoning about the outcome based on setting up the initial state, and not controlling anything after that, the tasks in these benchmarks are also sensitive to small changes in initial state. While effective for evaluating general-purpose, long-term physical reasoning, the simulation-based benchmarks neglect other properties of real-world systems such as noisy observations and environment stochasticity that make reasoning about dynamic interactions difficult.

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, shown in Fig. 1. Allowing actions only at the beginning of the task, marble run tasks are similar to the simulation-based physical reasoning benchmarks but also incorporate real-world stochasticity which can cause widely varying outcomes for even the same initial state. Additionally, while our robot runs hundreds of trials without human intervention, it can take as long as a minute to setup and execute a single trial, so learning algorithms evaluated in this domain must perform well under a limited evaluation budget.

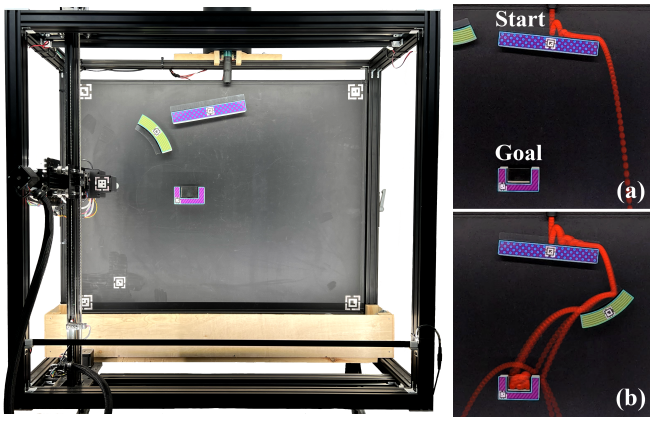


Fig. 1: The marble run robot autonomously evaluates learning algorithms on real-world marble run tasks. (a-b) An example marble run task (a) and solution (b), where the goal is to place the curved track so the ball lands in the U-shaped goal. Stochasticity of the ball's initial state leads to large variations in outcome for the same action.

Due to small changes potentially 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 distinct types of solutions in case the first attempted strategy 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 [3], [4] to train an MoE 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, where the sub-policies represent promising regions of the action space to explore and the high-level policy directs the search towards high-reward regions. Our experiments show that online learning successfully fine-tunes

*This material is based upon work supported by the National Science Foundation under Grant IIS-1849287

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA {aallaire, cga}@cmu.edu

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 and strategies learned in simulation or under different dynamics can still provide useful information about where to look for solutions.

II. RELATED WORK

Many environments for evaluating physical reasoning capabilities of learning algorithms have been explored, ranging from physics-based computer games [5], [6] to specially designed physical reasoning benchmarks [1], [2], [7], [8]. For evaluating real-world physical reasoning capabilities beyond prediction and question answering, the most common application is contact-rich manipulation tasks [9], [10], [11]. 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 [1] and PHYRE [2] 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 like noisy observations and environment stochasticity.

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 [3], [4]. An earlier instantiation of this framework incorporates a similar advantage-weighted policy update [12]. Relative entropy policy search (REPS) [13] and maximum a posteriori policy optimization (MPO) [14] are closely related and similarly derived as a constrained policy search, but using the dual formulation instead.

Policy hierarchies in robot learning are often represented as options, which are temporally extended actions [15], [16]. Without temporal abstraction, options are reduced to components of a mixture distribution that form abstractions in space. We focus on abstractions in space using a mixture of experts policy representation [17]. Hierarchical extensions to both REPS and MPO have been developed [18], [19] and are similar to ours due to 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. Instead, 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 stably learn the mixture distribution's underlying structure, while online learning adapts the sub-policies and distribution over policies to a specific task or environment.

III. THE MARBLE RUN ENVIRONMENT

To demonstrate the challenges associated with the marble run environment, we define a simple task which initially consists of a rectangular track at the top of the environment and a U-shaped goal in the bottom half of the environment. In the initial configuration, a ball released at the top of the environment, above the rectangular track, will not land in the goal, as shown in Fig. 1a. The robot must then find a configuration of the curved track that allows the ball to land in the goal. At first, this task may seem too easy - many humans could find a solution within just a few attempts. However, Fig. 1b shows that, due to slight variations in the environment, an action that is successful once may not be every time.

Therefore, we task the robot with finding actions that are *always* or *nearly always* successful, rather than just finding actions that worked the one time they are tried. 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 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 actions more robust to variance in the ball trajectories.

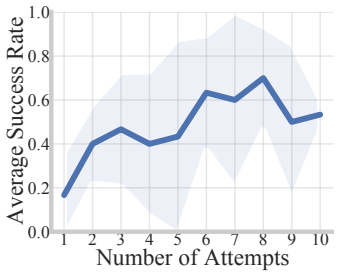


Fig. 2: Average performance of 5 humans on a marble run task over 10 attempts.

IV. FRAMEWORK FOR STRUCTURED EXPLORATION

A. Problem Definition

In this paper, 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. Only one object is allowed to be placed before each trial. The full 3D state of the environment is not observable, so we approximate the environment as 2D and ignore small out-of-plane movement. The state $\mathbf{s}$ is the initial scene configuration, which concatenates each object's position and orientation. The i^{th} object's state is given as $\mathbf{s}_i = [x, y, \sin(\theta), \cos(\theta)]$. Object positions are expressed relative to the ball's initial position, which is fixed across tasks, so the ball's state is not included. We consider tasks with the same number and types of objects so an encoding of object type is not necessary. With tracks magnetically fixed to a panel, we assume the ball is the only

dynamic object. The action is the position and orientation of the additional track placement, $a = [x, y, \sin(\theta), \cos(\theta)]$. 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 $R(\mathbf{s}, \mathbf{a}) = \{1, \text{ if } success; \ -d_{min} \text{ otherwise}\}$, 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, 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 of the estimated reward.

B. 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

$$k \sim \text{Categorical}(\psi(k|\mathbf{s}, \boldsymbol{\phi})) \tag{1}$$

$$\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k), \tag{2}$$

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.

C. Learning a Mixture of Experts Policy from (Simulated) Experience

We generate a dataset representing prior experiences using a 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 $(\mathbf{s}, \mathbf{a}, R(\mathbf{s}, \mathbf{a}))$. We do not expect the mixture policy trained in simulation to transfer perfectly to the real world, but it should provide a good starting point to search for potential solutions.

Expectation Maximization for Mixture of Experts Policies. We derive a training procedure from the expectation maximization (EM) algorithm [20]. 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 $\mathbf{s}$ and parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. 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.

$$J(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a} \sim \mathcal{D}} \Bigg[\sum_{k=1}^{K} w'_k \Big(\log \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k) + \log \psi_k(\mathbf{s}, \boldsymbol{\phi}) \Big) \Bigg] \tag{3}$$

$$w'_k = P(z_k = 1|\mathbf{s}, \mathbf{a}) = \frac{\psi_k(\mathbf{s}, \boldsymbol{\phi}')\pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}'_k)}{\sum_{j=1}^{K} \psi_j(\mathbf{s}, \boldsymbol{\phi}')\pi_j(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}'_j)} \tag{4}$$

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 $J(\boldsymbol{\phi}, \boldsymbol{\theta})$. However, there is not a closed-form solution when ψ and π_k are neural networks with non-linear activations as in this work. 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 [21].

Advantage-Weighted Regression. Using the mixture log-likelihood in (3), we apply advantage-weighted regression (AWR), which weights the log-likelihood with the exponential advantage $\exp(\frac{A^{\pi}(\mathbf{s},\mathbf{a})}{\eta})$ [3], [4], [12]. 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

$$J(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a} \sim \mathcal{D}} \Bigg[\sum_{k=1}^{K} w'_k \exp\left(\frac{A^{\pi'_k}(\mathbf{s}, \mathbf{a})}{\eta}\right) \Big(\log \pi_k(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta}_k) + \log \psi_k(\mathbf{s}, \boldsymbol{\phi}) \Big) \Bigg], \tag{5}$$

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 [19]. We treat η as a fixed hyperparameter, which has been effective in prior work [3], [4]. Only one action is taken and the final episode reward is observed immediately in marble run tasks, 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.

D. Online Learning

The mixture policy trained offline in simulation encapsulates 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. Expert policies represents promising regions of the action space to explore. The gating network

directs the search towards the most promising regions (or policies).

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 (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. A soft policy update shares information between policies and is important during earlier stages of training to learn a proper division of the state space. With enough sub-policies, we can assume a proper state space division is learned after training on the simulated dataset so benefits of the soft update are diminished.

Instead, 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 (5), where 1 is assigned to the component with the highest probability $P(z_k = 1|\mathbf{s}, \mathbf{a})$. We decompose (5) into separate objective functions for the gating policy and each expert policy. The objective function for policy k is

$$J(\boldsymbol{\theta}_k) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}_k}\Big[\exp\left(\frac{A^{\pi'_k}(\mathbf{s},\mathbf{a})}{\eta}\right)\log\pi_k(\mathbf{a}|\mathbf{s},\boldsymbol{\theta}_k)\Big], \quad (6)$$

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

$$J(\boldsymbol{\psi}) = \mathop{\mathbb{E}}_{\mathbf{s},k\sim\mathcal{D}}\Big[\exp\left(\frac{A^{\psi'}(\mathbf{s},k)}{\eta}\right)\log\psi_k(\mathbf{s},\boldsymbol{\phi})\Big], \quad (7)$$

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 [16].

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 separate 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 $\geq$ 0.5). During online learning, each reward function is updated using both online and offline samples from the corresponding policy.

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 $(\mathbf{s}_k, \mathbf{a}_k, R(\mathbf{s}_k, \mathbf{a}_k)) \sim \mathcal{D}_k$ containing an even mixture of positive and negative samples. The policy π_k is updated with a batch $(\mathbf{s}_k, \mathbf{a}_k, R(\mathbf{s}_k, \mathbf{a}_k)) \sim \mathcal{D}_k$ using (6) and the gating policy ψ is updated using (7) with a batch $(\mathbf{s}, \mathbf{a}, k, R(\mathbf{s}, \mathbf{a})) \sim \mathcal{D}$ sampled over the entire dataset. The batches used for both the policy and reward function updates are composed of a balanced ratio of online-to-offline samples which more aggressively updates the policy than uniform sampling. The ratio is initially set to 0 and linearly increased to 1 so each batch contains only online samples by a specified number of steps. For the expert policy and reward function updates, we increase the ratio to 1 in 25 steps. For the gating network, we increase the ratio more slowly over 100 steps to preserve exploration and reduce the risk of prematurely converging to a sub-optimal strategy.

V. Evaluation

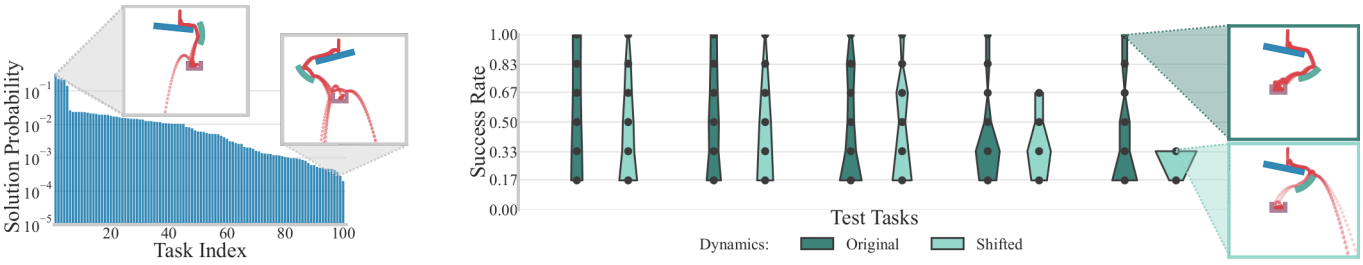
A. Simulation Setup

The simulated marble run environment is built in Box2D[1] 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. 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.

B. Metrics

The average success rate of a policy is used as a performance metric. The success rate refers to the number of

[1]https://box2d.org/



(a) (b)

Fig. 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 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.

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), making it difficult to differentiate performance across different conditions. We use the inter-quartile mean (IQM), which is less sensitive to outliers and stratified bootstrap confidence intervals to report aggregate performance [22].

C. Task Dataset Generation

We randomly generate 100 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 use task generation scripts from the PHYRE code-base to ensure tasks are non-trivial and sufficiently diverse [2]. The tasks are split into 80 training, 10 validation, and 10 test tasks.

Difficulty of Marble Run Tasks. 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. 3a demonstrate that difficulty varies significantly across tasks, with some requiring more than 10,000 actions before finding a solution with random sampling alone.

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. 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 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.

D. 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 IV-C. The average success rate after evaluating 20 actions sampled from the mixture policy on the test tasks is reported. For each evaluation step, we sample a policy from the categorical distribution over policies (i.e. the gating network) and use the mean of the selected 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 IV-D, by attempting 100 actions and updating the policy after each attempt. Every 5 attempts, we take an additional evaluation action using the mean of the sampled policy and record the performance. The evaluation steps are used only to report performance and are not used to update the policy. **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 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 IV-C and IV-D, except only a single policy is used so no gating policy is learned.

E. Simulation Experiments

In Fig. 4, we show the Online learning performance in simulation (Fig. 4a) and simulation with different dynamics

(a) Simulation (b) Shifted Dynamics

Fig. 4: Online learning performance in (a) simulation and (b) simulation with 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 average success rate over all evaluation steps is also shown for each task.

than the offline dataset (Fig. 4b). For each simulation environment, we plot the average success rate of the evaluation steps taken every 5 attempts during online learning. Additionally, we show the average success rate computed over all evaluation steps for each task.

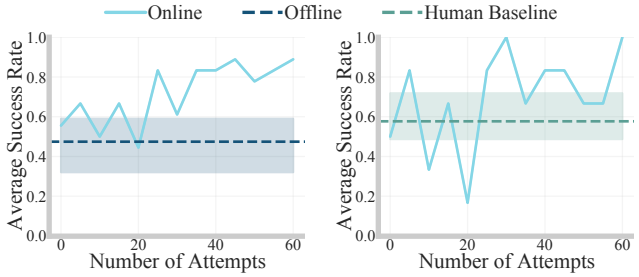
When the dynamics of the test environment match the dynamics of the training environment, the performance gained by representing multiple strategies is less pronounced. The offline performance of the Gaussian policy falls slightly below the MoE policy, likely due the Gaussian policy averaging over multiple solution regions which may include pockets of lower reward regions in 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 thus converge more quickly during online learning.

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. 4b. 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 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 evidenced by the offline policy performance which is less affected by the shifted dynamics than the single Gaussian policy.

For additional videos and results, please visit https://sites.google.com/view/learning-strategies-icra2023/home.

F. Real-World Experiments

Fig. 5a 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. 5b, we compare the performance of the MoE policy to the human performance from Fig. 2 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 performance is likely in the range of 0.8-0.9 for that task.



(a) Real World (b) Test Task 7

Fig. 5: (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.

VI. CONCLUSIONS

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. 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.

REFERENCES

[1] K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning," *Proceedings of the National Academy of Sciences*,

vol. 117, no. 47, pp. 29 302–29 310, 2020. [Online]. Available: https://www.pnas.org/content/117/47/29302

[2] A. Bakhtin, L. van der Maaten, J. Johnson, L. Gustafson, and R. Girshick, "Phyre: A new benchmark for physical reasoning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[3] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," *arXiv preprint arXiv:1910.00177*, 2019.

[4] A. Nair, M. Dalal, A. Gupta, and S. Levine, "Accelerating online reinforcement learning with offline datasets," *CoRR*, vol. abs/2006.09359, 2020. [Online]. Available: https://arxiv.org/abs/2006.09359

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[6] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik, "Learning visual predictive models of physics for playing billiards," in *ICLR*, 2016.

[7] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. 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.

[8] R. Girdhar and D. Ramanan, "CATER: A diagnostic dataset for compositional actions and temporal reasoning," *CoRR*, vol. abs/1910.04744, 2019. [Online]. Available: http://arxiv.org/abs/1910.04744

[9] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *CoRR*, vol. abs/1606.07419, 2016. [Online]. Available: http://arxiv.org/abs/1606.07419

[10] A. Ajay, M. Bauza, J. Wu, N. Fazeli, J. B. Tenenbaum, A. Rodriguez, and L. P. Kaelbling, "Combining Physical Simulators and Object-Based Networks for Control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[11] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[12] G. Neumann and J. Peters, "Fitted q-iteration by advantage weighted regression," in *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21. Curran Associates, Inc., 2009.

[13] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[14] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=S1ANxQW0b

[15] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[16] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[17] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1177–1193, 2012.

[18] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical relative entropy policy search," *Journal of Machine Learning Research*, vol. 17, no. 93, pp. 1–50, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-188.html

[19] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Y. Siegel, N. Heess, and M. A. Riedmiller, "Compositional transfer in hierarchical reinforcement learning," in *Robotics Science and Systems*, 2020. [Online]. Available: https://roboticsconference.org/2020/program/papers/54.html

[20] S.-K. Ng and G. McLachlan, "Using the em algorithm to train neural networks: misconceptions and a new algorithm for multiclass classification," *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 738–749, 2004.

[21] R. M. Neal and G. E. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*, 1998.

[22] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *Advances in Neural Information Processing Systems*, 2021.