Enhanced Meta Reinforcement Learning using Demonstrations in Sparse Reward Environments

Desik Rengarajan* **Sapana Chaudhary***
Jaewon Kim **Dileep Kalathil** **Srinivas Shakkottai**
Department of Electrical and Computer Engineering, Texas A&M University
{desik,sapanac,jwkim8804,dileep.kalathil,sshakkot}@tamu.edu

Abstract

Meta reinforcement learning (Meta-RL) is an approach wherein the experience gained from solving a variety of tasks is distilled into a meta-policy. The meta-policy, when adapted over only a small (or just a single) number of steps, is able to perform near-optimally on a new, related task. However, a major challenge to adopting this approach to solve real-world problems is that they are often associated with sparse reward functions that only indicate whether a task is completed partially or fully. We consider the situation where some data, possibly generated by a sub-optimal agent, is available for each task. We then develop a class of algorithms entitled Enhanced Meta-RL using Demonstrations (EMRLD) that exploit this information—even if sub-optimal—to obtain guidance during training. We show how EMRLD jointly utilizes RL and supervised learning over the offline data to generate a meta-policy that demonstrates monotone performance improvements. We also develop a warm started variant called EMRLD-WS that is particularly efficient for sub-optimal demonstration data. Finally, we show that our EMRLD algorithms significantly outperform existing approaches in a variety of sparse reward environments, including that of a mobile robot.

1 Introduction

Meta-Reinforcement Learning (meta-RL) is an approach towards quickly solving similar tasks while only gathering a few samples for each task. The fundamental idea behind approaches such as the popular Model Agnostic Meta-Learning (MAML) [9] is to determine a universal meta-policy by combining the information gained over working on several tasks. This meta-policy is optimized in a manner such that, when adapted using a small number of samples gathered for a given task, it is able to perform near optimally on that specific task. While the dependence of MAML on only a small number of samples for task-specific adaptation is very attractive, its also means that these samples must be highly representative of that task for meaningful adaptation and meta-learning. However, in many real-world problems, rewards provided are sparse in that they might only provide limited feedback. For instance, there might be a reward only if a robot gets close to designated way point, with no reward otherwise. Hence, MAML is likely to fail on both counts of task-specific adaptation and optimization of the meta-policy over tasks in sparse reward environments. Making progress towards learning a viable meta-policy in such settings is challenging without additional information.

Many real-world tasks are associated with empirically determined polices used in practice. Such polices could be inexpert, but even limited demonstration data gathered from applying these policies could contain valuable information in the sparse reward context. While the fact that the policy generating the data could be inexpert suggests that direct imitation might not be optimal, supervised

*Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

learning over demonstration data could be used for *enhancing* adaptation and learning. How best should we use demonstration data to enhance the process of meta-RL in the sparse reward setting?

Our goal is a principled design of a class of meta-RL algorithms that can exploit demonstrations from inexpert policies in the sparse reward setting. Our general approach follows two-step algorithms like MAML that employ: (i) Task-specific Adaptation: execute the current meta-policy on a task and adapt it based on the samples gathered to obtain a task-specific policy, and (ii) Meta-policy Optimization: execute task-specific policies on an ensemble of tasks to which they are adapted, and use the samples gathered to optimize the meta-policy from whence they were adapted. Our key insight is that we can enhance RL-based policy adaptation with behavior cloning of the inexpert policy to guide task-specific adaptation in the right direction when rewards are sparse. Furthermore, execution of such an enhanced adapted policy should yield an informative sample set that indicates how best to obtain the sparse rewards for meta-policy optimization. We aim at analytically and empirically capturing this progression of policy improvement starting from task-specific policy adaption to the ultimate meta-policy optimization. Thus, as long the inexpert policy has an advantage, we must be able to exploit it for meta-policy optimization.

Our main contributions are as follows. (i) We derive a policy improvement guarantee result for MAML-like two-step meta-RL algorithms. We show that the inclusion of demonstration data can further increase the policy improvement bound as long as the inexpert policy that generated the data has an advantage. (ii) We propose an approach entitled Enhanced Meta-RL using Demonstrations (EMRLD) that combines RL-based policy improvement and behavior cloning from demonstrations for task-specific adaptation. We further observe that directly applying the meta-policy to a new sparse-reward task sometimes does not yield informative samples, and a warm-start to the meta-policy using the demonstrations significantly improves the quality of the samples, resulting in a variant that we call EMRLD-WS. (iii) We show on standard MuJoCo and two-wheeled robot environments that our algorithms work exceptionally well, even when only provided with just one trajectory of sub-optimal demonstration data per task. Additionally, the algorithms work well even when exposed only to a small number of tasks for meta-policy optimization. (iv) Our approach is amenable to a variety of meta-RL problems wherein tasks can be distinguished across rewards (e.g., whether forward or backward motion yields a reward for a task) or across environment dynamics (e.g., the amount of environmental drift that a wheeled robot experiences changes across tasks). To illustrate the versatility of EMRLD, we not only show simulations on different continuous control multi-task environments, but also demonstrate its excellent performance via real-world experiments on a TurtleBot robot [2]. We provide videos of the robot experiments and code at `https://github.com/DesikRengarajan/EMRLD`.

Related Work: Here, we provide a brief overview of the related works. We leave a more thorough discussion on related works to the Appendix.

Meta learning: Basic ideas on the meta-learning framework are discussed in [15, 34, 8]. Model-agnostic meta-learning (MAML) [9] introduced the two-step approach described above, and can be used in the supervised learning and RL contexts. However, in its native form, the RL variant of MAML can suffer from issues of inefficient gradient estimation, exploration, and dependence on a rich reward function. Among others, algorithms like ProMP [28] and DiCE [11] address the issue of inefficient gradient estimation. Similarly, E-MAML [1, 33] and MAESN [12] deal with the issue of exploration in meta-RL. PEARL [26] takes a different approach to meta-RL, wherein task specific contexts are learnt during training, and interpreted from trajectories during testing to solve the task. HTR [23] relabels the experience replay data of any off-policy algorithm such as PEARL [26] to overcome exploration difficulties in sparse reward goal reaching environments. Different from this approach, we use demonstration data to aid learning and are not restricted to goal reaching tasks.

RL with demonstration: Leveraging demonstrations is an attractive approach to aid learning [13, 36, 21]. Earlier work has incorporated data from both expert and inexpert policies to assist with policy learning in sparse reward environments [21, 14, 36, 17, 27]. In particular, [14] utilizes demonstration data by adding it to the replay buffer for Q-learning, [25] proposes an online fine-tuning algorithm by combining policy gradient and behavior cloning, while [27] proposes a two-step guidance approach where demonstration data is used to guide the policy in the initial phase of learning.

Meta-RL with demonstration: Meta Imitation Learning [10] extends MAML [9] to imitation learning from expert video demonstrations. WTL [40] uses demonstrations to generate an exploration algorithm, and uses the exploration data along with demonstration data to solve the task. GMPS [20]

extends MAML [9] to leverage expert demonstration data by performing meta-policy optimization via supervised learning. Closest to our approach are GMPS [20] and Meta Imitation Learning [10], and we will focus on comparisons with versions of these algorithms, along with the original MAML [9].

Our work differs from prior work on meta-RL with demonstration in several ways. First, existing works assume the availability of data generated by an expert policy, which severely limits their ability to improve beyond the quality of the policy that generated the data. This degrades their performance significantly when they are presented with sub-optimal data generated by an inexpert policy that might be used in practice. Second, these works use demonstration data in a purely supervised learning manner, without exploiting the RL structure. We use a combination of loss functions associated with RL and supervised learning to aid the RL policy gradient, which enables our approach to utilize any reward information available. This makes it superior to existing work in the sparse reward environment, which we illustrate in several simulation settings and real-world robot experiments.

2 Preliminaries

A Markov Decision Processes (MDP) is typically represented as a tuple $< \mathcal{S}, \mathcal{A}, R, P, \gamma, \rho >$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability function, γ is the discount factor, and $\rho \in \Delta(\mathcal{S})$ is the initial state distribution. The value function V^π and the state-action value function Q^π of a policy π are defined as $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s\right]$ and $Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]$. The advantage function A^π is defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. A policy π generates a trajectory τ where $\tau = (s_0, a_0, s_1, a_1, \dots), s_0 \sim \rho, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim P(\cdot|s_t, a_t)$. Since the randomness of τ is specified by P and π, we denote it as $\tau \sim (P, \pi)$. The goal of a reinforcement learning algorithm is to learn a policy that maximizes the expected infinite horizon discounted reward defined as $J(\pi) = \mathbb{E}_{\tau \sim (\pi, P)}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$. It is easy to see that $J(\pi) = \mathbb{E}_{s_0 \sim \rho}[V^\pi(s_0)]$.

The discounted state-action visitation frequency of policy π is defined as $d^\pi(s, a) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s, a_t = a), s_0 \sim \rho$. The discounted state visitation frequency is defined as the marginal $d^\pi(s) = \sum_{a \in \mathcal{A}} d^\pi(s, a)$. It is straightforward to see that $d^\pi(s, a) = \pi(s, a) d^\pi(s)$.

The total variation (TV) distance between two distributions p and q is defined as $D_{\text{TV}}(p, q) = (1/2) \sum_x |p(x) - q(x)|$, the average TV distance between two policies π_1 and π_2, averaged w.r.t. the d^{π_π} is defined as $D^\pi_{\text{TV}}(\pi_1, \pi_2) = \mathbb{E}_{s \sim d^\pi}\left[D_{\text{TV}}(\pi_1, \pi_2)\right]$.

Gradient-based meta-learning: The goal of a meta-learning algorithm is to learn to perform optimally in a new (testing) task using only limited data, by leveraging the experience (data) from similar (training) tasks seen during training. Gradient-based meta-learning algorithms achieve this goal by learning a meta-parameter which will yield a good task specific parameter after performing only a few gradient steps w.r.t. the task specific loss function using the limited task specific data.

Meta-learning algorithms consider a set of tasks $\mathcal{T}$ with a distribution p over $\mathcal{T}$. Each task $i \sim p(\mathcal{T})$ is also associated with a data set $\mathcal{D}_i$, which is typically divided into training data $\mathcal{D}_i^{\text{tr}}$ used for task specific adaptation and validation data $\mathcal{D}_i^{\text{val}}$ used for meta-parameter update. The objective of the gradient-based meta-learning is typically formulated as

$$\min_\theta \; \mathbb{E}_{i \sim p(\mathcal{T})}\left[\mathcal{L}_i\left(\theta - \alpha \nabla_\theta \mathcal{L}_i(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{val}}\right)\right], \tag{1}$$

where $\mathcal{L}_i$ is the loss function corresponding to task i and α is the learning rate. Here, $\theta_i(\theta) = \theta - \alpha \nabla_\theta \mathcal{L}_i(\theta, \mathcal{D}_i^{\text{tr}})$ is the task specific parameter obtained by one step gradient update starting from the meta-parameter θ, and the goal is to find the best meta-parameter which will minimize the meta loss function $\mathcal{L}(\theta) = \mathbb{E}_{i \sim p(\mathcal{T})}[\mathcal{L}_i(\theta_i(\theta))]$

Gradient-based meta-reinforcement learning: The gradient-based meta-learning framework is applicable both in supervised learning and reinforcement learning. In RL, each task i corresponds to an MDP with task specific model P_i and reward function R_i. We assume that the state-action spaces are uniform across the tasks, thus ensuring the first level of task similarity. Task specific data $\mathcal{D}_i$ is the trajectories $\tau_{i,m} = (s_0^m, a_0^m, s_1^m, a_1^m, \dots), s_0 \sim \rho, a_t \sim \pi(s_t, \cdot), s_{t+1} \sim P_i(\cdot|s_t, a_t)$ for $1 \leq m \leq M$ generated according to some policy π. Since the randomness of the trajectory $\tau_{i,m}$ is specified by P_i and π, we denote it as $\tau_{i,m} \sim (P_i, \pi)$. We consider the function approximation setting where each policy π is represented by a function parameterized by $\theta \subset \Theta$ and is denoted as

π_θ. The task specific loss in meta-RL is defined as $\mathcal{L}^i_{\text{RL}}(\theta) = -J_i(\pi_\theta)$. The gradient $\nabla_\theta \mathcal{L}^i_{\text{RL}}(\theta)$ can then be computed using policy gradient theorem.

The standard meta-RL training is done follows. A task $i \sim p(\mathcal{T})$ (usually a batch of tasks) is sampled at each iterate k of the algorithm. Now, starting with meta-parameter θ_k, the training data $\mathcal{D}_i^{\text{tr}}$ for task adaptation is generated as the trajectories $(\tau_{i,m})_{m=1}^M$, where $\tau_{i,m} \sim (P_i, \pi_{\theta_k})$, and the updated parameter $\theta_{i,k}$ for task i is computed by policy gradient evaluated on $\mathcal{D}_i^{\text{tr}}$. The validation data $\mathcal{D}_i^{\text{val}}$ is then collected as trajectories $(\tau_{i,m})_{m=1}^M$, where $\tau_{i,m} \sim (P_i, \pi_{\theta_{i,k}})$, and the meta-parameter θ_k is updated by policy gradient evaluated on $\mathcal{D}_i^{\text{val}}$. In the next section, we will introduce a modified approach which will leverage the demonstration data for task adaptation and meta-parameter update.

3 Meta-RL using Demonstration Data

Most gradient-based meta-RL algorithms learn the optimal meta-parameter and the task-specific parameter from scratch using on-policy approaches. These algorithms exclusively rely on the reward feedback obtained from the training and validation data trajectories collected through the on-policy roll-outs of the meta-policy and task-specific policy. However, in RL problems with sparse rewards, a non-zero reward is typically achieved only when the task is completed or near-completed. In such sparse rewards settings, trajectories generated according to a policy that is still learning may not achieve any useful reward feedback, especially in the early phase of learning. In other words, since the reward feedback is zero or near-zero, the policy gradient will also be similar, resulting in non-meaningful improvement in the policy. Hence, standard meta-RL algorithms such as MAML, which rely crucially on reward feedback, will not be able to make much progress towards learning a valuable task-specific or meta-policy in sparse reward settings.

Learning the optimal control policy in sparse reward environments has been recognized as a challenging problem even in the standard RL setting, since most state-of-the-art RL algorithms fail to learn any meaningful polices even after a large number of training episodes [25, 17, 27]. One widely accepted approach to overcome this challenge is known as *learning from demonstration*, wherein demonstration data obtained from an expert [25] or inexpert policy [17, 27] is used to aid online learning. The intuitive idea is that, even though the demonstration data does not contain any reward feedback, it can be used to guide the learning agent to reach non-zero reward regions of state-action spaces. This guidance, usually in the direction of the goal/target, is achieved by inferring some pseudo reward signal through supervised learning approaches using demonstration data.

Can we enhance the performance of meta-RL algorithms in sparse reward environments by using demonstration data from sub-optimal experts? Meta-RL in sparse reward environments is significantly more challenging than that of the standard RL setting. This is because the reward feedback serves the dual objectives of adapting the meta-parameter to specific tasks and for updating the meta-parameter itself. We note that demonstration data helps with both of these objectives. Firstly, use of demonstration data to guide task-specific adaptation becomes important because adaptation is achieved in one or a few gradient steps, and policy resulting from each adaptation step might not achieve meaningful reward in a sparse reward setting. Secondly, making use of demonstration data for meta-parameter update is equally important because of the role of meta-policy as a reward-yielding exploratory policy. Intuitively, the meta-policy should yield trajectories that reach in the vicinity of the reward-achieving region of the state-action spaces. This does not happen in sparse reward environments. However, using the guidance from demonstration data, the task-specific policy obtained after the task adaptation may be able to generate trajectories that will reach within the reward-achieving region resulting in performance acceleration of the meta-policy.

For meta-learning with demonstration, we assume that each task i is associated with demonstration data $\mathcal{D}_i^{\text{dem}}$, which contains a trajectory generated according to a demonstration policy π^{dem} in an environment with model P_i. We *do not assume* that π^{dem} is the optimal policy for task i because in many real-world applications $\mathcal{D}_i^{\text{dem}}$ could be generated using an inexpert policy. Our key idea is to enhance task adaptation using the demonstration data by introducing an additional gradient term corresponding to the supervised learning guidance loss. We define the supervised learning loss function for task i as $\mathcal{L}_i^{\text{BC}}(\theta, \mathcal{D}_i^{\text{dem}}) = -\sum_{(s,a)\in\mathcal{D}_i^{\text{dem}}} \log \pi_\theta(a|s)$. We note that, though this loss function is the same as in behavior cloning (BC), we use it directly in the gradient update instead of performing a simple warm start. This approach is known to achieve superior performance than naive BC warm starting in standard RL problem under the sparse reward setting [25, 17, 27]. The

task adaptation step at iteration k, starting with meta-parameter θ_k is now obtained as

$$\theta_{k,i} = \theta_k - \alpha \nabla_\theta \left(\mathrm{w_{rl}} \mathcal{L}_i^{\mathrm{RL}}(\theta; \mathcal{D}_i^{\mathrm{tr}}) + \mathrm{w_{bc}} \mathcal{L}_i^{\mathrm{BC}}(\theta, \mathcal{D}_i^{\mathrm{dem}}) \right) |_{\theta=\theta_k}, \tag{2}$$

where $\mathrm{w_{rl}}$ and $\mathrm{w_{bc}}$ are hyperparameters that control the extant to which RL and demonstration data influence the gradient.

The next question is: how do we use demonstration data in the meta-parameter update? One approach is to use only the demonstration data with a supervised learning loss function for updating the meta-parameter as done in [20]. We conjecture that such a reduction to supervised learning will severely limit the learning capability of the algorithm. Firstly, if demonstration data is obtained from an inexpert policy, this approach will never be able to achieve the optimal performance. This is because the role of the meta-policy as a reward-yielding exploratory policy will be limited by true performance of the inexpert policy. Secondly, the task-specific policies obtained according to (2) may be able to reach within the reward-yielding region of state-action space as we mentioned before. Hence, the validation data $\mathcal{D}_i^{\mathrm{val}}$ collected through the roll-out of the policies obtained after task adaptation might contain extremely valuable reward feedback. Utilizing this data could potentially have a significant impact on improving the learning of the meta-parameter. Thus, in our approach, we update the meta-parameter using the RL loss with policy gradient as follows.

$$\theta_{k+1} = \theta_k - \beta \nabla_\theta \sum_i \mathcal{L}_i^{\mathrm{RL}}(\theta - \alpha \nabla_\theta \left(\mathrm{w_{rl}} \mathcal{L}_i^{\mathrm{RL}}(\theta; \mathcal{D}_i^{\mathrm{tr}}) + \mathrm{w_{bc}} \mathcal{L}_i^{\mathrm{BC}}(\theta, \mathcal{D}_i^{\mathrm{dem}}) \right); \mathcal{D}_i^{\mathrm{val}}) \tag{3}$$

We note that the demonstration data is indeed used in the meta-parameter update implicitly, as its impact can be observed in (3). We found empirically that the double use of the demonstration data, either by adding an additional gradient through a BC loss function, or by replacing $\mathcal{D}_i^{\mathrm{val}}$ with $\mathcal{D}_i^{\mathrm{dem}}$ results in similar or worse performance than the approach described above.

We now formally present our algorithm called Enhanced Meta-RL using Demonstrations (EMRLD).

Algorithm 1 Enhanced Meta-RL using Demonstrations (EMRLD)

Input: Set $\mathcal{T}$ of N tasks, demonstration data $\mathcal{D}_i^{\mathrm{dem}}$ for task $i = 1, \ldots, N$
Input: Adaptation learning rate α, meta-learning rate β
Initialize meta parameter θ_0
for $k = 0, 1, \ldots$ **do**
  for $i \in \mathcal{T}$ **do**
    Execute the meta policy π_{θ_k} for task i to collect $\mathcal{D}_i^{\mathrm{tr}}$
    Task adaptation: Compute task adapted parameter $\theta_{k,i}$ using $\mathcal{D}_i^{\mathrm{tr}}$ and $\mathcal{D}_i^{\mathrm{dem}}$ according to 2
    Collect data $\mathcal{D}_i^{\mathrm{val}}$ by executing adapted policy $\pi_{\theta_{k,i}}$ for meta-policy update
  end for
  Meta-parameter update: Update meta-policy according to 3 using $\mathcal{D}_i^{\mathrm{val}}$
end for

We now present a theoretical justification of why EMRLD should have a superior performance in the sparse reward setting as compared to other gradient based algorithms that do not use demonstration data. First, we introduce some notation. Let $\pi_k = \pi_{\theta_k}$ be the meta-policy used at iteration k of our algorithm. Also, let $\pi_{k,i}$ be the policy obtained after task-specific adaptation for task i. Recall that $J_i(\pi_{k,i})$ is the value of the policy for the MDP corresponding to task i. Similarly, we can define the state-action value function and advantage function of policy $\pi_{k,i}$ for task i as $Q_i^{\pi_{k,i}}$ and $A_i^{\pi_{k,i}}$, respectively. Also, let $d_i^{\pi_{k,i}}$ be the visitation frequency of policy $\pi_{k,i}$ for task i. Now, we can define the value of the meta-policy π_k over the ensemble of all tasks as $J_{\mathrm{meta}}(\pi_k) = \mathbb{E}_{i \sim p(\mathcal{T})}[J_i(\pi_{k,i})]$.

If the demonstration data has to be useful, it should provide a reasonable amount of guidance. In particular, we would like the task-specific policy adapted using this data to collect feedback that would ensure good meta-policy updates, particularly in the initial stages of meta-training. Since the capability of the demonstration data to guide adaptation will depend on the demonstration policy π_i^{dem} according to which it is generated, we make the following assumption about π_i^{dem}.

Assumption 3.1. During the initial stages of meta-training, $\mathbb{E}_{a \sim \pi_i^{\mathrm{dem}}(s,\cdot)}[A_i^{\pi_{k,i}}(s,a)] \geq \Delta$, for all $s \in \mathcal{S}$ and $i \in \mathcal{T}$.

Assumption 3.1 implies that during the initial stages of meta-training, the demonstration policy can provide a higher advantage on average than the current policy adapted to that task. This is a reasonable assumption, since any reasonable demonstration policy is likely to perform much better than an untrained policy in the initial phase of learning. We also note that a similar assumption was used in learning from demonstration literature [17, 27]

We now present the performance improvement result for EMRLD.

Theorem 3.2. *Let $\pi_k = \pi_{\theta_k}$ be the meta-policy used at iteration k of our algorithm and let $\pi_{k,i}$ be the policy obtained after task adaptation in task i. Let Assumption 3.1 holds for π_k. Then,*

$$J_{meta}(\pi_{k+1}) - J_{meta}(\pi_k) \geq \left(\frac{1}{1-\gamma} \mathbb{E}_{i\sim p(\mathcal{T}),(s,a)\sim d_i^{\pi_{k,i}}} \left[\frac{\pi_{k+1,i}(s,a)}{\pi_{k,i}(s,a)} A_i^{\pi_{k,i}}(s,a) \right] \right.$$
$$\left. - \frac{2C_1}{1-\gamma} \mathbb{E}_{i\sim p(\mathcal{T})} \left[D_{TV}^{\pi_{k,i}}(\pi_{k+1,i}, \pi_{k,i}) \right] \right) + \left(\frac{\Delta}{1-\gamma} - \frac{2C_1}{1-\gamma} \mathbb{E}_{i\sim p(\mathcal{T})} \left[D_{TV}^{\pi_{k+1,i}}(\pi_{k+1,i}, \pi_i^{dem}) \right] \right)$$

Theorem 3.2 presents a lower bound for the meta policy improvement as a sum of two groups of terms. Maximizing the first term in group one with a constraint on its second term will ensure a higher lower bound and hence an improvement in the meta-parameter training. We notice that this is indeed achieved by the TRPO step used in the meta-parameter update. Hence, this first group is the same for any MAML-type of algorithm. The advantage of the demonstration data is revealed in the second group of terms. The term $\Delta/(1-\gamma)$ adds a positive quantity to the lower bound, and this contribution from this second group of term can be maximized by minimizing $\mathbb{E}_{i\sim p(\mathcal{T})}[D_{TV}^{\pi_{k+1,i}}(\pi_{k+1,i}, \pi_i^{\text{dem}})]$. However, this minimization is hard to perform in practice because estimating $D_{TV}^{\pi_{k+1,i}}$ requires sampling the data according to $\pi_{k+1,i}$, and this is not feasible at iteration k. Hence, in practice, we replace that term by $\mathbb{E}_{i\sim\mathcal{T}}[D_{TV}^{\pi_i^{\text{dem}}}(\pi_{k+1,i}, \pi_i^{\text{dem}})]$. This can be easily achieved by including the standard maximum likelihood objective in the adaptation step. Thus, EMRLD both exploits the advantage offered by an RL step, as well as that of behavior cloning for meta-policy optimization.

We can further improve the performance of EMRLD by including a behavior cloning warm starting step before performing the update (2). We simplify this warm start to a one step gradient as, $\theta_{\tilde{k},i} = \theta_k - \alpha \nabla_\theta \mathcal{L}_i^{\text{BC}}(\theta, \mathcal{D}_i^{\text{dem}})|_{\theta=\theta_k}$, and then do the task adaptation as in (2) starting with $\theta_{\tilde{k},i}$. We call this version of our algorithm as EMRLD-WS. Such a warm start is likely to provide more meaningful samples than directly rolling out the meta-policy to obtain samples for task-specific adaptation. In the next section, we will see empirically how our design choices for EMRLD and EMRLD-WS enable them to learn policies that provide higher rewards using only a small amount of (even sub-optimal) demonstration data.

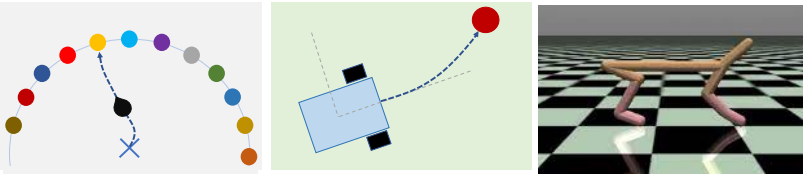


Figure 1: Point2D Navigation (first), Two Wheeled Locomotion (second) and HalfCheetah (last).

4 Experimental Evaluation

We evaluate the performance of EMRLD based on whether the meta-policy it generates is a good initial condition for task-specific adaptation in sparse-reward environments over (i) Tasks already seen in training and (ii) New unseen tasks. We seek to validate the conjecture that in the sparse-reward setting, EMRLD should be able to leverage even demonstrations of inexpert policies to attain high test performance over previously unseen tasks. We do so with regard to two classes of tasks, namely,

• Tasks that differ in their reward functions: Simulation experiments on Point2D Navigation [9], TwoWheeled Locomotion [12], and HalfCheetah [38, 35]

• Tasks that differ in the environment dynamics: Real-world experiments using a TurtleBot, which is a two-wheeled differential drive robot [2].

4.1 Experiments on simulated environments

Sparse multi-task environments We present simulation results for three standard environments shown in Figure 1 and described below. We train over a small number of tasks that differ in their reward functions. We generate unknown tasks for test by randomly modifying the reward function.

Point2D Navigation is a 2 dimensional goal-reaching environment. The states are the (x, y) location of the agent on a 2D plane. The actions are appropriate 2D displacements (dx, dy). Training tasks are defined by a fixed set of 12 goal locations on a semi-circle of radius 2. The agent is given a zero reward everywhere except when it is a certain distance near the goal location, making the reward function highly *sparse*. Within a single task, the objective of the agent is to reach the goal location in the least number of time steps starting from origin. Test tasks are generated by sampling any point on the semicircle as the goal.

TwoWheeled Locomotion environment is a goal-reaching with sparse rewards, similar to Point2D Navigation. However, the robot is constrained by the permissible actions (limits on angular and linear velocity) and trajectories feasible based on the turning radius of the robot. Here, our training tasks are a fixed set of 24 goal locations on a semi-circle of radius 2, while test goals are sampled randomly. Further details on state-space and dynamics are provided in the Appendix.

HalfCheetah Forward-Backward consists of two tasks in which the HalfCheetah agent learns to either move in the forward (task 1) or backward (task 2) directions with as high velocity as possible. The agent gets a reward only after it has moved a certain number of units along the x-axis in the correct direction, making the rewards sparse. Training and test are under the same two tasks.

Optimal data and sub-optimal data We provide a limited amount of demonstration data in the form of *just one trajectory per task for guidance*. Optimal data consists of (s, a) transitions generated by an expert policy trained using TRPO. Sub-optimal data is generated by an inexpert, partially trained TRPO policy with induced action noise and truncated trajectories as shown in Figure 2.

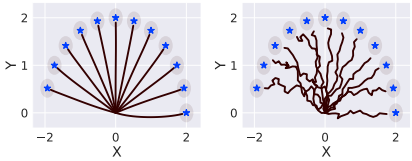


Figure 2: Optimal and Sub-optimal demonstrations for Point2D Navigation.

Baselines We compare the performance of our algorithm against the following gradient based meta-reinforcement learning algorithms: (i) **MAML:** [9] The standard MAML algorithm for meta-RL (ii) **Meta-BC:** A variant of [10]; this is a supervised learning/behavior cloning version of MAML, where the maximum likelihood loss is used in the adaptation as well as the meta-optimization steps. (iii) **GMPS:** Guided meta policy search [20], which uses RL for gradient based adaptation, and uses demonstration data for supervised meta-parameter optimization. The implemention of our algorithms and baselines is based on a publicly available meta-learning code base [3] licensed under the MIT License.

Performance with optimal demonstration data: We illustrate the training and testing performance of the different algorithms trained and tested with optimal data in Figure 3. The top row of Figure 3 shows the average adapted return across training tasks of the meta-policy during training iterations. The bottom row of Figure 3 shows the average return of the trained meta-policy adapted across testing tasks over adaptation steps. We see that our algorithms out perform the others by obtaining the highest average return, and are able to quickly adapt to testing tasks with just one adaptation step and one trajectory of demonstration data. Additionally, our algorithms demonstrate a nearly-monotone improvement in average return demonstrating stable learning. Meta-BC fails and has unstable training performance as the amount of demonstration data available per task is very small. Training over only a small number of tasks further hampers the performance of Meta-BC. MAML and GMPS fail to learn due to sparsity of the environment as the purely RL adaptation step incurs almost zero reward, and hence, negligible learning signal. Furthermore, GMPS is hampered in the meta update step due to availability of only a small amount of demonstration data per-task.

Performance with sub-optimal demonstration data: EMRLD uses a combination of RL and imitation, which is valuable when presented with sub-optimal demonstrations. For the Point2D Navigation environment, we collect sub-optimal data for each task using a partially trained agent with induced action noise, and truncate the trajectories short of the reward region. Hence, pure imitation cannot reach the goal. For the TwoWheeled Locomotion environment, we collect data in a

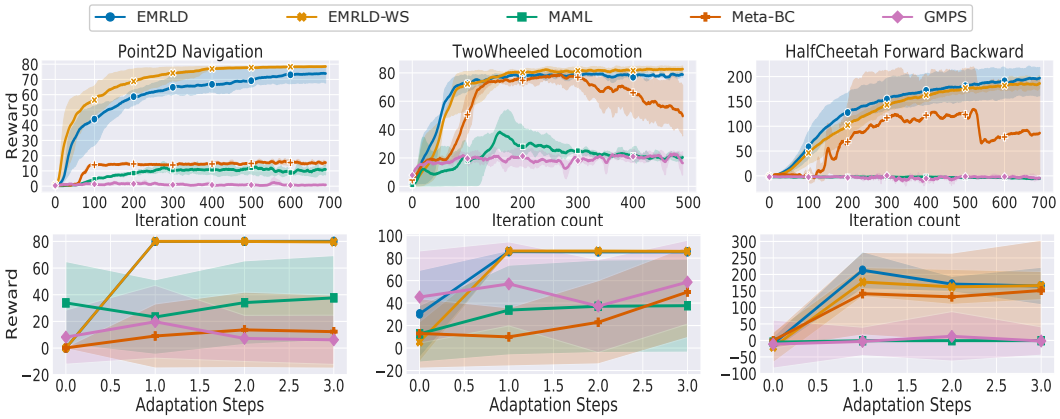


Figure 3: Training (top) and test (bottom) plots on 2D Navigation, Wheeled locomotion and Half Cheetah with **optimal demonstration data**. For training curves, a solid line corresponds to the mean over 3 seeds and the shaded region corresponds to the standard deviation over them. For the test plots, a solid line corresponds to the mean performance over all testing tasks, and the shaded region corresponds to the standard deviation over them.

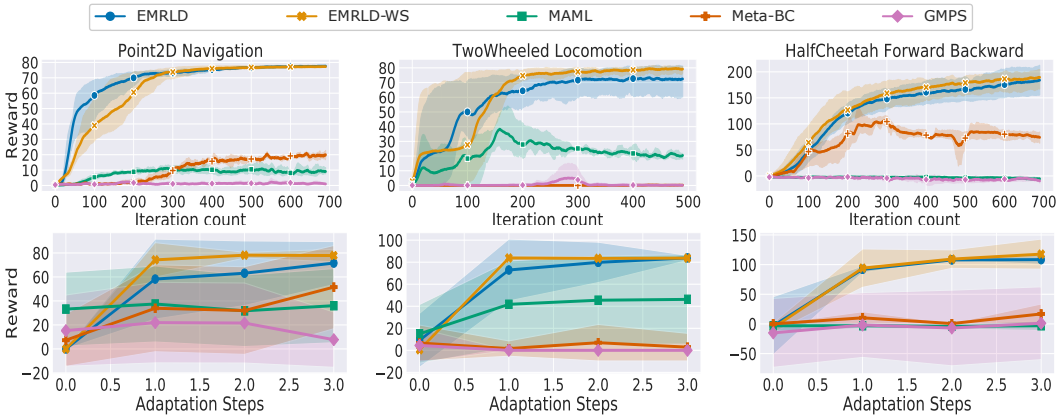


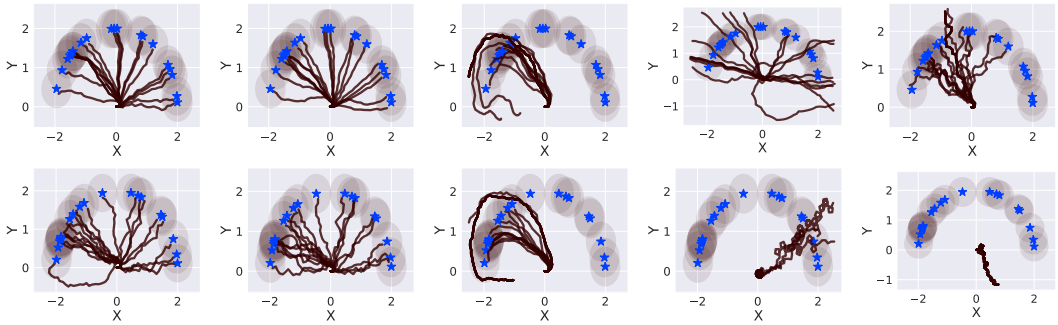
Figure 4: Training (top) and adaptation (bottom) for 2D Navigation, Wheeled locomotion and Half Cheetah with **sub-optimal demonstration data**. Notation is similar to Figure 3.

similar fashion for all tasks, but remove state-action pairs at the beginning of each trajectory. Since the first few state-action pairs contain information on how to orient the two-wheeled agent towards the goal, this truncation eliminates the possibility of direct imitation being successful. Similarly, in HalfCheetah we use a partially trained policy and truncate trajectories before they reach the reward bearing region. Figure 4 illustrates that EMRLD outperforms all the baselines and is quickly able to adapt to unseen tasks, emphasizing the benefit of its RL component. Meta-BC and GMPS fail because they are restricted by the optimality of the data, and the absence of crucial information greatly impacts their performance. MAML again fails due to the sparsity of the reward.

We conclude by presenting in Figure 5, sets of trajectories generated during testing in the TwoWheeled-Lococmotion environment when provided with optimal or sub-optimal demonstration data. The variants of EMRLD clearly outperform the others, showing their strength in the sparse reward setting.

4.2 Real-world Experiments on TurtleBot

We demonstrate the ability of EMRLD variants to adapt to sparse-reward tasks when they differ in environment dynamics and have sparse reward feedback. We do so via performance evaluation in the real-world using a TurtleBot shown in Figure 6 (left). We first we modify the *TwoWheeled*



(a) EMRLD (b) EMRLD-WS (c) MAML (d) Meta-BC (e) GMPS

Figure 5: Trajectories after one step of adaptation using optimal demonstration data (top), and after 3 steps of adaptation using sub-optimal data (bottom) for 20 random goal points in *TwoWheeled Locomotion*. The star indicates the goal, and reward is available only in the shaded region.

Locomotion sparse reward environment by fixing the goal, and changing the dynamics by inducing a residual angular velocity which mimics drifts in the environment. This environmental drift is what differentiates each task. In other words, for a given task, the environment would cause the robot to drift in some specific unknown direction. We train on a set of 9 tasks with different angular velocity values (i.e., 9 different driving environments). We use one trajectory of demonstration data per task collected using an expert policy trained using TRPO. Note that all the training and data collection is done in simulation. The results are shown in Figure 6 (middle), where we see that the EMRLD variants clearly outperform the others.

For testing, we consider the environment where the Turtlebot experiences a fixed but unknown residual angular velocity representing environmental drift. Thus, we bias the angular velocity control of the TurtleBot by some amount unknown to the algorithm under test. We first execute the meta policy on the TurtleBot in the real world to collect 5 trajectories. We also provide one trajectory of simulated demonstration data. We use these samples to adapt the meta-policy, and execute the adapted task-specific policy on the TurtleBot. The results are shown in Figure 6 (right), where the origin is at $(0, 0)$ and the goal is indicated by a star. It is clear that the variants of EMRLD are the best at quickly adapting to the drift in the environment and are successful with just one step of adaptation.

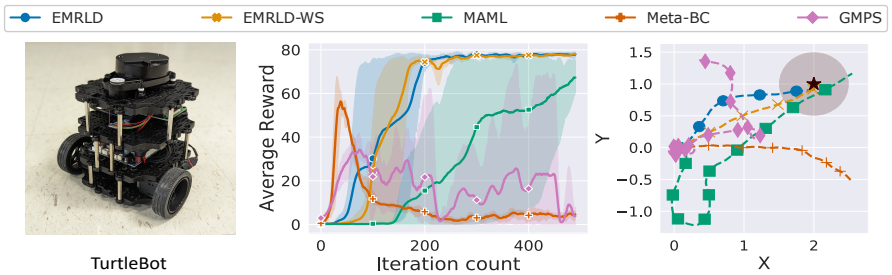


Figure 6: The TurtleBot (left) is a small, two-wheeled robot. We train the algorithms in simulation (middle) and obtain trained meta-policies. The meta-polices are adapted under an unknown drift in the real world to control the TurtleBot to attain a given goal point (right).

5 Conclusion

We studied the problem of meta-RL algorithm design for sparse-reward problems, in which demonstration data generated by a possibly inexpert policy is also provided. Our key observation was that simple application of an untrained meta-policy in a sparse-reward environment might not provide meaningful samples, and guidance provided by imitating the inexpert policy can greatly assuage this effect. We first showed analytically that this insight is accurate and that meta-policy improvement might be feasible as long as the inexpert demonstration policy has an advantage. We then developed two meta-RL algorithms, EMRLD and EMRLD-WS that are enhanced by using demonstration data.

We show through extensive simulations, as well as real world robot experiments that EMRLD is able to offer a considerable advantage over existing approaches in sparse reward scenarios.

6 Limitations and Future Work

EMRLD inherits the limitations of the gradient-based meta-RL approaches like MAML namely on-policy training, and data collection and gradient computation during test adaptation. A limitation specific to our proposed algorithms is the assumption on availability of task specific demonstration data. However, we reiterate that for a small number of train tasks, this assumption is quite practical, further, our framework allows for this data to be sub-optimal.

A possible future direction to explore is the context based meta-RL (that does't require gradient computation during testing) with demonstration data. Another future work direction is to explore usage of demonstration data in off-policy meta-RL algorithms.

7 Acknowledgement

This work was supported in part by the National Science Foundation (NSF) grants NSF-CAREER-EPCN-2045783 and NSF ECCS 2038963, and U.S. Army Research Office (ARO) grant W911NF-19-1-0367. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

[2] Robin Amsters and Peter Slaets. Turtlebot 3 as a robotics education platform. In *Robotics in Education - Current Research and Innovations, Proceedings of the 10th RiE*, Advances in Intelligent Systems and Computing, pages 170–181, 2019.

[3] Sèbastien M R Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. learn2learn: A library for Meta-Learning research. August 2020.

[4] Kai Arulkumaran, Dylan R Ashley, Jürgen Schmidhuber, and Rupesh K Srivastava. All you need is supervised learning: From imitation learning to meta-rl with upside down rl. *arXiv preprint arXiv:2202.11960*, 2022.

[5] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

[6] Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422, 2019.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[8] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[10] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017.

[11] Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018.

[12] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *arXiv preprint arXiv:1802.07245*, 2018.

[13] Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, pages 3223–3230, 2018.

[14] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. 2017.

[15] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[16] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

[17] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International Conference on Machine Learning*, pages 2469–2478. PMLR, 2018.

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

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

[20] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[22] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[23] Charles Packer, Pieter Abbeel, and Joseph E Gonzalez. Hindsight task relabelling: Experience replay for sparse reward meta-rl. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[25] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[26] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

[27] Desik Rengarajan, Gargi Vaidya, Akshay Sarvesh, Dileep Kalathil, and Srinivas Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration. In *International Conference on Learning Representations*, 2022.

[28] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

[29] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[30] Jüergen Schmidhuber, Jieyu Zhao, and MA Wiering. Simple principles of metalearning. *Technical report IDSIA*, 69:1–23, 1996.

[31] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[33] Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

[34] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[36] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[37] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[38] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 380–390. Springer, 2009.

[39] Tony Z Zhao, Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine. Offline meta-reinforcement learning for industrial insertion. *arXiv preprint arXiv:2110.04276*, 2021.

[40] Allan Zhou, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn. Watch, try, learn: Meta-learning from demonstrations and reward. *arXiv preprint arXiv:1906.03352*, 2019.

Ethics Statement and Societal Impacts

Our work considers the theory and instantiation of meta-RL algorithms that were trained and tested on simulation and robot environments. No human subjects or human generated data were involved. Thus, we do not perceive ethical concerns with our research approach.

While reinforcement learning shows much promise for application to societally valuable systems, applying it to environments that include human interaction must proceed with caution. This is because guarantees are probabilistic, and ensuring that the risk is kept within acceptable limits is a must to ensure safe deployments.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Section 1
   (b) Did you describe the limitations of your work? [Yes] Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Section 3
   (b) Did you include complete proofs of all theoretical results? [Yes] Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Supplementary Material (`https://github.com/DesikRengarajan/EMRLD`)
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Section 4
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Section 4
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Appendix
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 3.2

We will use the well known Performance Difference Lemma [16] in our analysis.

Lemma A.1 (Performance difference lemma, [16]). *For policies any two policies π_1 and π_2,*

$$J(\pi_1) - J(\pi_2) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi_1}, a\sim\pi_1(s,\cdot)}\left[A^{\pi_2}(s,a)\right], \tag{4}$$

where $J(\pi_j) = \mathbb{E}_{s_0\sim\rho}\left[V^{\pi_j}(s_0)\right] = \frac{1}{(1-\gamma)}\mathbb{E}_{s\sim d^{\pi_j}, a\sim\pi_j(s,\cdot)}\left[R(s,a)\right]$, for $j = 1, 2$.

Proof of Theorem 3.2. Recall the following notations: π_k is the meta-policy used at iteration k of our algorithm, $\pi_{k,i}$ is the policy obtained after task-specific adaptation for task i, $d_i^{\pi_{k,i}}$ is the state-visitation frequency of policy $\pi_{k,i}$ for task i, and $J_i(\pi_{k,i})$ is the value of the policy for the MDP corresponding to task i. The value of the meta-policy π_k is defined as $J_{\text{meta}}(\pi_k) = \mathbb{E}_{i\sim p(\mathcal{T})}[J_i(\pi_{k,i})]$.

We can obtain a performance difference lemma for the meta-policies as follows.

$$\begin{aligned}
(1-\gamma)\left(J_{\text{meta}}(\pi_{k+1}) - J_{\text{meta}}(\pi_k)\right) &= (1-\gamma)\left(\mathbb{E}_{i\sim p(\mathcal{T})}\left[J_i(\pi_{k+1,i})\right] - \mathbb{E}_{i\sim p(\mathcal{T})}\left[J_i(\pi_{k,i})\right]\right) \\
&= (1-\gamma)\mathbb{E}_{i\sim p}\left[J_i(\pi_{k+1,i}) - J_i(\pi_{k,i})\right] \\
&= \mathbb{E}_{i\sim p(\mathcal{T})}\left[\mathbb{E}_{s\sim d_i^{\pi_{k+1,i}}, a\sim\pi_{k+1,i}(s,\cdot)}\left[A_i^{\pi_{k,i}}(s,a)\right]\right] \\
&= \mathbb{E}_{i\sim p(\mathcal{T}), s\sim d_i^{\pi_{k+1,i}}, a\sim\pi_{1,i}(s,\cdot)}\left[A_i^{\pi_{k,i}}(s,a)\right],
\end{aligned} \tag{5}$$

where the third equality follows from Lemma A.1. Staring from (5), we get

$$\begin{aligned}
&(1-\gamma)\left(J_{\text{meta}}(\pi_{k+1}) - J_{\text{meta}}(\pi_k)\right) = \mathbb{E}_{i\sim p(\mathcal{T}), s\sim d_i^{\pi_{k+1,i}}, a\sim\pi_{k+1,i}(s,\cdot)}\left[A_i^{\pi_{k,i}}(s,a)\right] \\
&- \mathbb{E}_{i\sim p(\mathcal{T}), s\sim d_i^{\pi_{k,i}}, a\sim\pi_{k+1,i}(s,\cdot)}\left[A_i^{\pi_{k,i}}(s,a)\right] + \mathbb{E}_{i\sim p(\mathcal{T}), s\sim d_i^{\pi_{k,i}}, a\sim\pi_{k+1,i}(s,\cdot)}\left[A_i^{\pi_{k,i}}(s,a)\right] \\
&= \sum_i p(i)\sum_s d_i^{\pi_{k,i}}(s)\sum_a \pi_{k+1,i}(s,a)A_i^{\pi_{k,i}}(s,a) \\
&\qquad + \sum_i p(i)\sum_s \left(d_i^{\pi_{k+1,i}}(s) - d_i^{\pi_{k,i}}(s)\right)\sum_a \pi_{k+1,i}(s,a)A_i^{\pi_{k,i}}(s,a)
\end{aligned} \tag{6}$$

We with now consider the second term of equation 6:

$$\begin{aligned}
&\sum_i p(i)\sum_s \left(d_i^{\pi_{k+1,i}}(s) - d_i^{\pi_{k,i}}(s)\right)\sum_a \pi_{k+1,i}(s,a)A_i^{\pi_{k,i}}(s,a) \\
&\quad = \sum_i p(i)\sum_s d_i^{\pi_{k+1,i}}(s)\sum_a \pi_i^{\text{dem}}(s,a)\,A_i^{\pi_{2,i}}(s,a) \\
&\qquad + \sum_i p(i)\sum_s d_i^{\pi_{k+1,i}}(s)\sum_a \left(\pi_{k+1,i}(s,a) - \pi_i^{\text{dem}}(s,a)\right)A_i^{\pi_{k,i}}(s,a) \\
&\qquad - \sum_i p(i)\sum_s d_i^{\pi_{k,i}}(s)\sum_a \left(\pi_{k+1,i}(s,a) - \pi_{k,i}(s,a)\right)A_i^{\pi_{k,i}}(s,a) \\
&\qquad - \sum_i p(i)\sum_s d_i^{\pi_{k,i}}(s)\sum_a \pi_{k,i}(s,a)A_i^{\pi_{k,i}}(s,a) \\
&\quad \overset{(a)}{\geq} \Delta - \sum_i p(i)\sum_s d_i^{\pi_{k+1,i}}(s)\sum_a \left|\pi_{k+1,i}(s,a) - \pi_i^{\text{dem}}(s,a)\right|\left|A_i^{\pi_{k,i}}(s,a)\right| \\
&\qquad - \sum_i p(i)\sum_s d_i^{\pi_{k,i}}(s)\sum_a \left|\pi_{k+1,i}(s,a) - \pi_{k,i}(s,a)\right|\left|A_i^{\pi_{k,i}}(s,a)\right| \\
&\quad \overset{(b)}{=} \Delta - 2C_1\mathbb{E}_{i\sim p(\mathcal{T})}\left[D_{TV}^{\pi_{k+1,i}}\left(\pi_{k+1,i}, \pi_i^{\text{dem}}\right)\right] - 2C_1\mathbb{E}_{i\sim p(\mathcal{T})}\left[D_{TV}^{\pi_{k,i}}\left(\pi_{k+1,i}, \pi_{k,i}\right)\right]
\end{aligned} \tag{7}$$

Here, we get (a) is from Assumption 3.1 from which we have $\sum_a \pi_i^{\text{dem}}(s,a)\,A_i^{\pi_{k,i}}(s,a) \geq \Delta$, $\forall s, i$, and noting that $\sum_a \pi_{k,i}(s,a)A^{\pi_{k,i}}(s,a) = 0$ by definition of advantage function. We get (b) by

denoting $C_1 = \max_i \max_{s,a} \left|A_i^{\pi_{k,i}}(s,a)\right|$. Using (7) in 6, we get

$$J_{\text{meta}}(\pi_{k+1}) - J_{\text{meta}}(\pi_k) \geq \left(\frac{1}{1-\gamma}\mathbb{E}_{i\sim p(\mathcal{T}),(s,a)\sim d_i^{\pi_{k,i}}}\left[\frac{\pi_{k+1,i}(s,a)}{\pi_{k,i}(s,a)}A_i^{\pi_{k,i}}(s,a)\right]\right.$$
$$\left.-\frac{2C_1}{1-\gamma}\mathbb{E}_{i\sim p(\mathcal{T})}\left[D_{TV}^{\pi_{k,i}}(\pi_{k+1,i},\pi_{k,i})\right]\right) + \left(\frac{\Delta}{1-\gamma} - \frac{2C_1}{1-\gamma}\mathbb{E}_{i\sim p(\mathcal{T})}\left[D_{TV}^{\pi_{k+1,i}}(\pi_{k+1,i},\pi_i^{\text{dem}})\right]\right),$$

which completes the proof. □

B Environments

In this section, we describe all the simulation and real-world environments in detail.

B.1 Simulation Environments

Point 2D Navigation: Point 2D Navigation [9] is a 2 dimensional goal reaching environment with $\mathcal{S} \subset \mathbb{R}^2$, $\mathcal{A} \subset \mathbb{R}^2$, and the following dynamics,

$$x_{t+1} = x_t + dx_t, \quad y_{t+1} = x_t + dy_t, \quad \text{such that } dx_t^2 + dy_t^2 \leq 0.1^2$$

Where x_t and y_t are the x and y location of the agent, dx_t and dy_t are the actions taken which correspond to the displacement in the x and y direction respectively, all taken at time step t. The goals are located on a semi circle of radius 2, and the episode terminates when the agent reaches the goal or spends more than 100 time steps in the environment. The sparse reward function for the agent is defined as follows,

$$R_t = \begin{cases} 1 - \sqrt{(x_{t+1}-x_g)^2 + (y_{t+1}-y_g)^2} & \text{if } \sqrt{(x_{t+1}-x_g)^2 + (y_{t+1}-y_g)^2} \leq 0.2 \\ 100 - t - 1 & \text{if } \sqrt{(x_{t+1}-x_g)^2 + (y_{t+1}-y_g)^2} \leq 0.02, \\ 0 & \text{otherwise,} \end{cases}$$

where x_g and y_g are the x, y location of the goal. The agent is given a zero reward everywhere except when it is a certain distance $D_1 = 0.2$ near the goal location. Within the distance D_1, the agent is given two kinds of rewards. If the agent is very close to the goal, say a distance $D_2 = 0.02$, then it rewarded with a positive bonus of $1\times$`Number_of_times_steps_remaining_in_episode`. This is done to create a sink near goal location to trap the agent inside it, rather than letting it wander in the D_1 region to keep collecting misleading positive reward. For distances between 0.02 and 0.2, the agent is given a positive reward of `1-dist(agent,goal)`.

TwoWheeled Locomotion: The TwoWheeled Locomotion environment [12] is designed based on the two wheeled differential drive model with $\mathcal{S} \subset \mathbb{R}^2$, $\mathcal{A} \subset \mathbb{R}^2$, and the following dynamics,

$$x_{t+1} = x_t + v_t\cos(\theta_t)dT, \;\; y_{t+1} = y_t + v_t\sin(\theta_t)dT, \;\; \theta_{t+1} = \theta_t + \omega_t dT,$$

with $v_t \in [0, 0.22], \omega_t \in [-2.84, 2.84]$, where x_t, y_t correspond to the x and y coordinate of the agent, v_t and ω_t are the actions corresponding to the linear and angular velocity of the agent all at time t, and $dT = 0.5$ is the time discretization factor. Goals are located on a semi-circle of radius 2, and the episode terminates if the agent reaches the goal, or spends more than 100 time steps in the environment, or moves out of region, which is a square box of side 2.5. The sparse reward function for the agent is defined as follows,

$$R_t = \begin{cases} 1 - \sqrt{(x_{t+1}-x_g)^2 + (y_{t+1}-y_g)^2} & \text{if } \sqrt{(x_{t+1}-x_g)^2 + (y_{t+1}-y_g)^2} \leq 0.5 \\ 100 - t - 1 & \text{if } |x_{t+1}-x_g| \leq 0.2 \quad \text{and} \quad |y_{t+1}-y_g| \leq 0.2, \\ 0 & \text{otherwise,} \end{cases}$$

where x_g and y_g are the x, y location of the goal.

Half Cheetah Forward-Backward: The Half Cheetah Forward-Backward environment [9], is a modified version of the standard MuJoCo[35] HalfCheetah environment with $\mathcal{S} \subset \mathbb{R}^{20}$ and $\mathcal{A} \subset \mathbb{R}^6$, where the agent is tasked with moving forward or backward, with the episode terminating if the agent spends more than 100 time steps in the environment. The sparse reward function is as follows,

$$R_t = \begin{cases} d_g \cdot \dfrac{(x_{t+1}-x_t)}{dT} - c_t & \text{if } |x_{t+1}-x_0| > 2. \\ 0 & \text{otherwise,} \end{cases}$$

where x_t corresponds to the x position of the agent, c_t is the control cost, all at time step t, dT is the time discretization factor, and d_g is the goal direction, which is $+1$ for the forward task and -1 for the backward task.

TwoWheeled Locomotion - Changing Dynamics: We modify the TwoWheeled locomotion environment by fixing the goal to $(2, 1)$, and adding a residual angular velocity,

$$x_{t+1} = x_t + v_t \cos(\theta_t) dT, \quad y_{t+1} = y_t + v_t \sin(\theta_t) dT, \quad \theta_{t+1} = \theta_t + \omega_g + \omega_t dT$$

with $v_t \in [0, 0.15], \omega_t \in [-1.5, 1.5]$, where ω_g is the residual angular velocity, which corresponds to different task, and mimics drift in the environment. The sparse reward function is similar to the one described in section B.1.

$$R_t = \begin{cases} 1 - \sqrt{(x_{t+1} - 2)^2 + (y_{t+1} - 1)^2} & \text{if } \sqrt{(x_{t+1} - x_g)^2 + (y_{t+1} - y_g)^2} \leq 0.5 \\ 100 - t - 1 & \text{if } |x_{t+1} - 2| \leq 0.1 \quad and \quad |y_{t+1} - 1| \leq 0.1, \\ 0 & \text{otherwise,} \end{cases}$$

B.2 Real-World TurtleBot Platform and Experiments

We deploy the policy trained on the environment described in section B.1 on a TurtleBot 3 [2], a real world open source differential drive robot. We use ROS as a middleware to set up communication between the bot and a custom built OpenAI Gym environment. The OpenAI Gym environment acts as an interface between the policy being deployed and the bot. The custom built environment, subscribes to ROS topics (`/odom` for x_t, y_t, θ_t), which are used to communicate the state of the bot, and publish (`/cmd_vel` for v_t, ω_t) actions. This is done asynchronously through a callback driven mechanism. The bot transmits its state information over a wireless network to an Intel NUC, which transmits back the corresponding action according to the policy being deployed.

The trajectories executed by the adapted policies are plotted in figure 7 (note that figure 7 is the same as figure 6, re-plotted here for clarity). During policy execution on the TurtleBot, we set the residual angular velocity that mimics drift to $\omega_g = -0.65$, we note that our algorithms (EMRLD and EMRLD-WS) are able to adapt to the drift in the environment and reach the goal. We further note that MAML, takes a longer sub-optimal route to reach the reward region, but misses the goal.

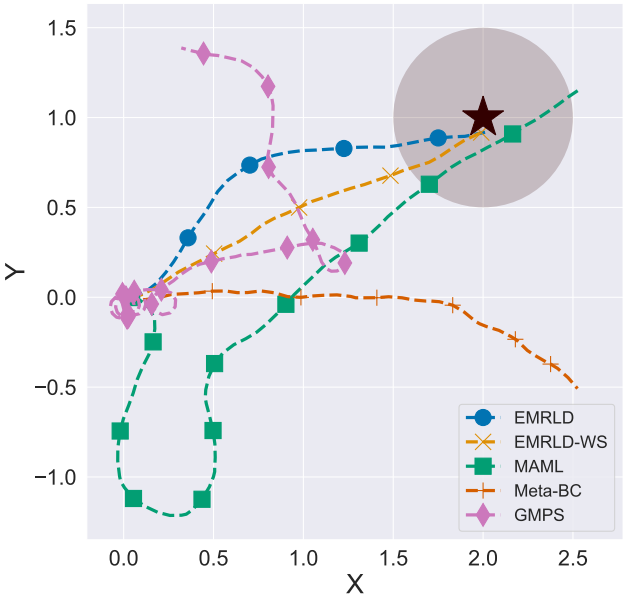


Figure 7: Trajectories in the real world for all algorithms with residual angular velocity $\omega_g = -0.65$

We have provided a link to real-world demonstration with our code[2]. For EMRLD, we show the execution of the meta policy used to collect data, and the adapted policy. It can be clearly seen that the the meta policy collects rewards in the vicinity of the goal region, which is then used for adaptation. The adapted policy then reaches the goal. We further show the execution of the adapted policies for the baseline algorithms on the TurtleBot, and we can observe that EMRLD and EMRLD-WS outperform all the baseline algorithms and reach the goal.

[2]https://github.com/DesikRengarajan/EMRLD

C Experimental Setup

Computing infrastructure and run time: The experiments are run on computers with AMD Ryzen Threadripper 3960X 24-Core Processor with max CPU speed of 3800MHz. Our implementation does not make use of GPUs. Insead, the implementation is CPU thread intensive. On an average, EMRLD and EMRLD-WS take $\sim$3h to run on smaller environments, and take $\sim$5h on HalfCheetah. We train goal conditioned expert policies using TRPO. Expert policy training takes $\sim$0.5h to run. Our code is based on *learn2learn*[3] [3], a software library built using PyTorch [24] for Meta-RL research.

Neural Network and Hyperparameters: In our work, the meta policy π^{dem} and the adapted policies $\pi_{k,i}$ are stochastic Gaussian policies parameterized by neural networks. The input for each policy network is the state vector s and the output is a Gaussian mean vector μ. The standard deviation σ is kept fixed, and is not learnable. During training, an action is sampled from $\mathcal{N}(\mu, \sigma)$.

For value baseline (used for advantage computation) of meta-learning algorithms, we use a linear baseline function of the form $B(s) = \zeta_{s,t}^{\top} G(s)$, where $\zeta_{s,t} = \text{concat}\left(s, s \odot s, 0.01t, (0.01t)^2, (0.01t)^3, 1\right)$, and $G(s)$ is discounted sum of rewards starting from state s till the end of an episode. This was first proposed in [7] and is used in MAML [9]. This is preferred as a learnable baseline can add additional gradient computation and backpropagation overheads in meta-learning.

We use TRPO on goal conditioned policies to obtain optimal and sub-optimal experts for all the tasks in an environment at once. For each environment, the task context variable, *i.e.*, a vector that contains differentiating information on a task, is appended to the input state vector of the policy network. The rest of the policy mechanism is same as described above for meta-policies. A learnable value network is used to cut variance in advantage estimation. Once the expert policy is trained to the desired amount, just one trajectory per task is sampled to construct demonstration data.

All the models used in this work are multi-layer perceptrons (MLPs). The policy models for all the meta-learning algorithms have two layers of 100 neurons each with Rectified Linear Unit (ReLU) non-linearities. The data generating policy and value models use two layers of 128 neurons each.

Table 1 lists the set of hyperparameters used for EMRLD, EMRLD-WS and the baseline algorithms. In addition to the ones listed in Table 1, meta batch size is dependant on the training environment: it is 12 for Point2D Navigation, 24 for TwoWheeled Locomotion and 10 for HalfCheetah Forward-Backward. In Table 1, Meta LR specified as 'TRPO' means that the learning rate is determined by step-size rule coming from TRPO. The meta optimization steps in Meta-BC and GMPS use ADAM [18] optimizer with a learning rate of 0.01. We use 20 CPU cores to parallelize policy rollouts for adaptation. The hyperparameters $\mathrm{w_{rl}}$ and $\mathrm{w_{bc}}$ are kept fixed across environments for EMRLD and EMRLD-WS. The parameter $\mathrm{w_{bc}}$ is kept at 1 for both optimal and sub-optimal data, and across environments. The parameter $\mathrm{w_{rl}}$ takes a lower value of 0.2 across environments for optimal data as in practise optimal data is expected to be highly informative. Hence, we desire the gradient component arising from optimal data to hold more value while adaptation. For sub-optimal data, the agent is required to explore to obtain performance beyond data, and hence, $\mathrm{w_{rl}}$ is kept at 1. We further show in section D that our algorithm is robust to choice of $\mathrm{w_{bc}}$ and $\mathrm{w_{rl}}$.

D Sensitivity Analysis

We perform sensitivity analysis for parameters $\mathrm{w_{rl}}$ and $\mathrm{w_{bc}}$ on our algorithms EMRLD and EMRLD-WS for optimal data on Point2D Navigation. The results for the same are included in Fig. 8. All the plots are averaged over three random seed runs. To assess the sensitivity of our algorithms to $\mathrm{w_{rl}}$, we fix $\mathrm{w_{bc}} = 1$ and vary $\mathrm{w_{rl}}$ to take values from $0.2, 0.4, 0.6, 0.8$ and 1. Similarly, to assess how sensitive our algorithm's performance to $\mathrm{w_{bc}}$ is, we fix $\mathrm{w_{rl}} = 1$ and vary $\mathrm{w_{bc}}$ to take values from $0.2, 0.4, 0.6, 0.8$ and 1. All the hyperparameters are kept fixed to the values listed in Table 1. We observe that our algorithms are fairly robust to variations in $\mathrm{w_{rl}}$ and $\mathrm{w_{bc}}$ for three random seeds. Since demonstration data is leveraged to extract useful information regarding the environment and the reward structure, our algorithms are slightly more sensitive to $\mathrm{w_{bc}}$ variation than $\mathrm{w_{rl}}$ variation.

[3] `https://github.com/learnables/learn2learn`

HYPERPARAMETER	EMRLD	EMRLD-WS	MAML	META-BC	GMPS
ADAPTATION LR	0.01	0.01	0.01	0.01	0.01
META LR	TRPO	TRPO	TRPO	0.01(ADAM)	0.01(ADAM)
ADAPT STEPS	1	1	1	1	1
ADAPT BATCH SIZE	20	20	20	20	20
GAE τ	1	1	1	1	1
γ	0.95	0.95	0.95	0.95	0.95
CPU THREAD NO.	20	20	20	20	20
$\mathrm{w_{rl}}$	0.2/1	0.2/1	N/A	N/A	N/A
$\mathrm{w_{bc}}$	1	1	N/A	N/A	N/A

Table 1: Hyperparameter values for EMRLD, EMRLD-WS, MAML, Meta-BC and GMPS. The hyperparameters are kept fixed across algorithms, across environments, and across demonstration data type.

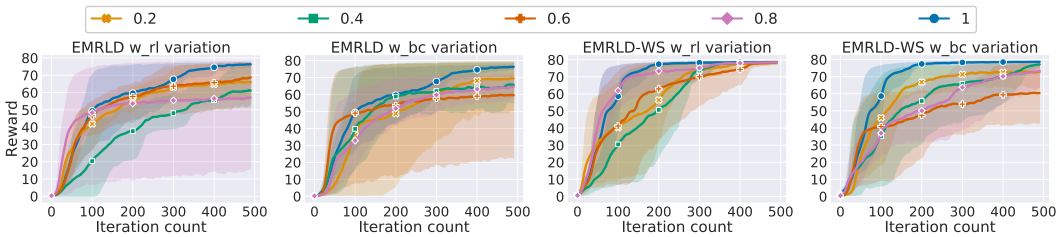


Figure 8: Sensitivity analysis for EMRLD and EMRLD-WS when the demonstration data is optimal. (a) Keeping $\mathrm{w_{bc}} = 1$, $\mathrm{w_{rl}}$ is varied from 0.2 to 1 in steps of 0.2 for EMRLD. (b) Keeping $\mathrm{w_{rl}} = 1$, $\mathrm{w_{bc}}$ is varied from 0.2 to 1 in steps of 0.2 for EMRLD. (c) Keeping $\mathrm{w_{bc}} = 1$, $\mathrm{w_{rl}}$ is varied from 0.2 to 1 in steps of 0.2 for EMRLD-WS. (d) Keeping $\mathrm{w_{rl}} = 1$, $\mathrm{w_{bc}}$ is varied from 0.2 to 1 in steps of 0.2 for EMRLD-WS.

E Ablation experiments

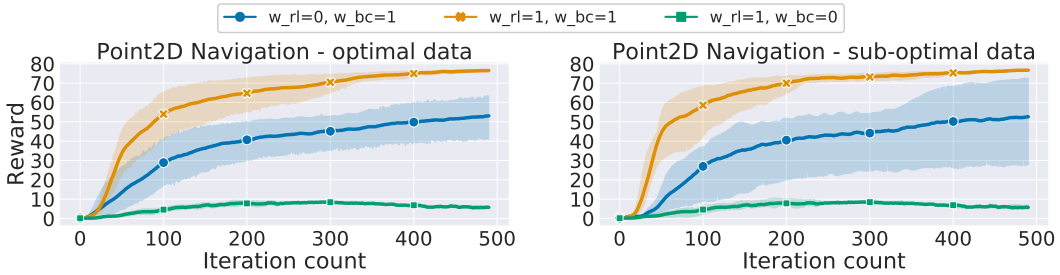


Figure 9: Ablation experiments on EMRLD for Point2D Navigation environment by changing w_{bc} and w_{rl} in the adaptation step with the optimal and the sub-optimal demonstration data.

We perform ablation experiments for EMRLD by setting $\mathrm{w_{bc}} = 0$ and $\mathrm{w_{rl}} = 0$ on the Point2D Navigation environment with the optimal and the sub-optimal demonstration data. We observe from figure 9, that setting $\mathrm{w_{bc}} = 0$ hampers the performance to a greater extant as the agent is unable to extract useful information from the environment due to the sparse reward structure. We also observe that setting $\mathrm{w_{rl}} = 0$ hampers the performance, as the agent is unable to exploit the RL structure of the problem to achieve high rewards.

F Related Work

Meta-Learning: Reinforcement learning (RL) has become popular as a tool to perform *learning from interaction* in complex problem domains like autonomous navigation of stratospheric balloons [5] and autonomously solving a game of Go [32]. In large scale complex environments, one requires

a large amount of data to learn any meaningful RL policy [6]. This is in stark contrast to how we as humans behave and learn - by translating our *prior knowledge* of past exposure to same/similar tasks into behavioural policies for a new task at hand. The initial work [30] took to addressing the above mentioned gap and proposed the paradigm of meta-learning. The idea has been extended to obtain gradient based algorithms in supervised learning, unsupervised learning, control, and reinforcement learning [31, 15, 34, 37, 8]. More recently, model-agnostic meta-learning (MAML) [9] introduced a gradient based two-step approach to meta-learning: an inner adaptation step to learn specific task policies, and an outer meta-optimization loop that implicitly makes use of the inner policies. MAML can be used both in the supervised learning and RL contexts. Reptile [22] introduced efficient first order meta-learning algorithms. PEARL [26] takes a different approach to meta-RL, wherein task specific contexts are learned during training, and interpreted from trajectories during testing to solve the task. In its native form, the RL variant of MAML can suffer from issues of inefficient gradient estimation, exploration, and dependence on a rich reward function. Among others, algorithms like ProMP [28] and DiCE [11] address the issue of inefficient gradient estimation. Similarly, E-MAML [1, 33] and MAESN [12] deal with the issue of exploration in meta-RL. Inadequate reward information or sparse rewards is a particularly challenging problem setting for RL , and hence, for meta-RL. Very recently, HTR [23] proposed to relabel the experience replay data of any off-policy algorithm to overcome exploration difficulties in sparse reward goal reaching environments. Different from this approach, we leverage the popular *learning from demonstration* idea to aid learning of meta-policies on tasks including and beyond goal reaching ones.

RL with demonstration: 'Learning from demonstrations' (LfD) [29] first proposed the use of demonstrations in RL to speed up learning. Since then, leveraging demonstrations has become an attractive approach to aid learning [13, 36, 21]. Earlier work has incorporated data from both expert and inexpert policies to assist with policy learning in sparse reward environments [21, 14, 36, 17, 27]. In particular, DQfD [14] utilizes demonstration data by adding it to the replay buffer for Q-learning. DDPGfD[36] extend use of demonstration data to continuous action spaces, and is built upon DDPG [19]. DAPG [25] proposes an online fine-tuning algorithm by combining policy gradient and behavior cloning. POfD [17] propose an approach to use demonstration data through an appropriate loss function into the RL policy optimization step to implicitly reshape sparse reward function. LOGO [27] proposes a two-step guidance approach where demonstration data is used to guide the RL policy in the initial phase of learning.

Meta-RL with demonstration: Use of demonstration data in meta-RL is new, and the works in this area are rather few. Meta Imitation Learning [10] extends MAML [9] to imitation learning from expert video demonstrations. WTL [40] uses demonstrations to generate an exploration algorithm, and uses the exploration data along with demonstration data to solve the task. ODA [39] use demonstration data to perform offline meta-RL for industrial insertion, and [4] propose generalized 'upside down RL' algorithms that use demonstration data to perform offline-meta-RL. GMPS [20] extends MAML [9] to leverage expert demonstration data by performing meta-policy optimization via supervised learning. Closest to our approach are GMPS [20] and Meta Imitation Learning [10], and we will focus on comparisons with versions of these algorithms, along with the original MAML [9].