RL^3: Boosting Meta Reinforcement Learning via RL inside RL^2

Abhinav Bhatia **Samer B. Nashed** **Shlomo Zilberstein**
Manning College of Information and Computer Sciences
University of Massachusetts Amherst
{abhinav,snashed,shlomo}@cs.umass.edu

Abstract

Meta reinforcement learning (meta-RL) methods such as RL^2 have emerged as promising approaches for learning data-efficient RL algorithms tailored to a given task distribution. However, these RL algorithms struggle with long-horizon tasks and out-of-distribution tasks since they rely on recurrent neural networks to process the sequence of experiences instead of summarizing them into general RL components such as value functions. Moreover, even transformers have a practical limit to the length of histories they can efficiently reason about before training and inference costs become prohibitive. In contrast, traditional RL algorithms are data-inefficient since they do not leverage domain knowledge, but they do converge to an optimal policy as more data becomes available. In this paper, we propose RL^3, a principled hybrid approach that combines traditional RL and meta-RL by incorporating task-specific action-values learned through traditional RL as an input to the meta-RL neural network. We show that RL^3 earns greater cumulative reward on long-horizon and out-of-distribution tasks compared to RL^2, while maintaining the efficiency of the latter in the short term. Experiments are conducted on both custom and benchmark discrete domains from the meta-RL literature that exhibit a range of short-term, long-term, and complex dependencies.

1 Introduction

Reinforcement learning (RL) has been shown to produce effective policies in a variety of applications including both virtual [1] and embodied [2, 3] systems. However, traditional RL algorithms have three major drawbacks: they often have difficulty generalizing beyond the exact task they were trained on, can be slow to converge, and require a large amount of data. These shortcomings are especially glaring in settings where the goal is to learn policies for a collection or distribution of problems that share some similarities, and for which traditional RL must start from scratch for each problem. For example, many robotic manipulation tasks require interacting with an array of objects with similar but not identical shapes, sizes, weights, materials, and appearances, such as mugs and cups. It is likely that effective manipulation strategies for this entire set of tasks will be similar, but they may also differ in ways that make it challenging to learn a single policy that is highly successful on all instances. Recently, meta reinforcement learning (meta-RL) has been proposed as an approach to mitigate these shortcomings by essentially trading efficiency at training time and system complexity for adaptation speed and generalizability during testing time.

While meta-RL systems represent a significant improvement over traditional RL in such settings, there are still several obstacles preventing widespread adoption of meta-RL techniques, especially in embodied systems. They still require large amounts of data during training time, can have poor performance on long-horizon tasks, and although they "learn to learn" they often generalize poorly to tasks not represented in the training distribution. Ideally, we would like meta-RL systems to

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023)

achieve high short-term data-efficiency, good asymptotic performance, and generalization to both in-distribution and out-of-distribution (OOD) tasks. Moreover, we would like to achieve these improvements without relying on privileged information, such as known task descriptions.

To that end, we propose RL^3, a principled approach that embeds the strengths of traditional RL within meta-RL. Our approach leverages the universality of traditional action-value estimates, their ability to compress trajectories into useful summaries, their direct actionability, their asymptotic optimality and their ability to inform task-identification, in order to enhance out-of-distribution (OOD) generalization and performance over extended horizons. The key idea in RL^3 is an additional 'object-level' RL procedure executed within the meta-RL architecture that computes task-specific Q-value estimates as supplementary inputs to the meta-learner, in conjunction with standard trajectory histories. Moreover, this technique can also work with an abstract, or coarse, representation of the object-level MDP. In principle, our approach allows the meta-learner to learn how to optimally fuse raw trajectory data with the summarizations provided by the Q-value estimates. In this work, RL^3 is implemented by injecting Q-value estimates into RL^2 [4] as the base meta-RL algorithm, which is modified to use transformers instead of recurrent neural networks.

The primary contribution of this paper is a confirmation of the hypothesis that injecting Q-estimates obtained via traditional object-level RL alongside the typical trajectory histories within a meta-RL agent leads to higher returns on long-horizon tasks and better OOD generalization, while maintaining short-term efficiency. Our conclusion is based on the results of experiments on the *Bandit* and *MDPs* domains used in previous work [4, 5] as well as on a more challenging custom Gridworld domain that requires long-term reasoning. We also elaborate extensively on the key insights that inform our approach and show theoretically that object-level Q-values are directly related to the optimal meta-value function.

2 Related Work

Although meta-RL is a fairly new topic of research, the general concept of meta-learning is decades old [6], which, coupled with a significant number of design decisions for meta-RL systems, has created a large number of different proposals for how systems ought to best exploit the resources available within their deployment contexts [7]. At a high level, most meta-RL algorithms can be categorized as either parameterized policy gradient (PPG) models [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] or black box models [4, 21, 22, 23, 5, 24, 25, 26, 27, 28, 29, 30]. PPG approaches assume that the underlying learning process is best represented as a policy gradient, where the set of parameters that define the underlying algorithm ultimately form a differentiable set of meta-parameters that the meta-RL system may learn to adjust. The additional structure provided by this assumption, combined with the generality of policy gradient methods, means that typically PPG methods retain greater generalization capabilities on out-of-distribution tasks. However, due to their inherent data requirements, PPG methods are often slower to adapt and initially train.

In this paper we focus on black box models, which represent the meta-learning function as a neural network, often a recurrent neural network (RNN) [4, 21, 22, 24, 25, 26, 27, 28] or a transformer [5, 31, 32]. There are also several hybrid approaches that combine PPG and black box methods, either during meta-training [33] or fine-tuning [34, 35]. Using black box models simplifies the process of augmenting meta states with Q-estimates and allows us to retain relatively better data efficiency while relying on the Q-value injections for better long-horizon performance and generalization.

Meta-RL systems may also leverage extra information available during training, such as task identification [24, 28]. Such 'privileged information' can of course lead to more performant systems, but is not universally available. As our hypothesis does not rely on the availability of such information, we expect our approach to be orthogonal to, and compatible with, such methods. Black box meta-RL systems that do not use privileged information still vary in several ways, including the choice between on-policy and off-policy learning and, in systems that use neural networks, the choice between transformers [36] and RNNs [37, 38, 39].

The most relevant methods to our work are end-to-end methods, which use a single function approximator to subsume both learner and meta-learner, such as RL^2 [4], L2L [22], SNAIL [5], and E-RL^2 [14], and methods that exploit the the formal description of the meta-RL problem as a POMDP or a Bayes-adaptive MDP (BAMDP) [40]. These methods attempt to learn policies conditioned on the BAMDP belief state while also approximating this belief state by, for example, variational

inference (VariBAD) [27, 41], or random network distillation on belief states (HyperX) [42]. Or, they simply encode enough trajectory history to approximate POMDP beliefs (RL^2) [4, 22].

Our proposed method is an end-to-end system that exploits the BAMDP structure of the meta-RL problem by spending a small amount of extra computation to provide inputs to the end-to-end learner that more closely resemble important constituents of BAMDP value functions. Thus, the primary difference between this work and previous work is the injection of Q-value estimates into the meta-RL agent state at each meta-step, in addition to the state-action-reward trajectories. In this work, our approach, RL^3, is implemented by simply injecting Q-value estimates into RL^2 alongside trajectory history, although any other meta-RL algorithm can be used.

3 Background and Notation

In this section, we briefly cover some notation and concepts upon which this paper is built.

3.1 Partially Observable Markov Decision Processes

Markov decision processes (MDPs) underpin reinforcement learning, and in this paper we use the standard notation defining an MDP as a tuple $M = \langle S, A, T, R \rangle$, where S is a set of states; A is a set of actions; T is the transition and R is the reward function. A *partially observable Markov decision process* (POMDP) extends MDPs to settings with partially observable states. A POMDP is described as a tuple $\langle S, A, T, R, \Omega, O \rangle$, where S, A, T, R are as in an MDP. Ω is the set of possible observations, and $O : S \times A \times \Omega \to [0, 1]$ is an observation function representing the probability of receiving observation ω after performing action a and transitioning to state s'. POMDPs can alternatively be represented as continuous-state belief-MDPs where a belief state $b \in \Delta^{|S|}$ is a probability distribution over all states. In this representation, a policy π is a mapping from belief states to actions, $\pi : \Delta^{|S|} \to A$. The belief state can be updated using the equation $b'(s'|b, a, \omega) = \alpha O(a, s', \omega) \sum_{s \in S} T(s, a, s') b(s)$, where α is a normalization constant.

3.2 Reinforcement Learning

The goal of reinforcement learning (RL) is to learn an optimal policy given an MDP with unknown dynamics using transition and reward feedback. This is often done by incrementally estimating the optimal action-value function $Q^*(s, a)$ [43], which satisfies the Bellman optimality equation $Q^*(s, a) = \mathbb{E}_{s'}[R(s, a) + \gamma \max_{a' \in A} Q^*(s', a')]$. In large or continuous state settings, it is popular to use deep neural networks to represent the action-value functions [1]. We denote the vector representing the Q-estimates of all actions at state s as $Q(s)$, and after t feedback steps, as $Q^t(s)$. Q-learning is known to converge asymptotically [44] provided each state-action pair is explored sufficiently. As a rough general statement, $||Q^t(s) - Q^*(s)||_\infty$ is proportional to $\approx \frac{1}{\sqrt{t}}$, with strong results on the convergence error available [45, 46, 47]. The theoretical objective in RL is to optimize the value of the final policy, i.e. the cumulative reward per episode, disregarding the data cost incurred and the cumulative reward missed (or *regret*) during learning due to suboptimal exploration.

3.3 Meta Reinforcement Learning

Meta reinforcement learning seeks action selection strategies that minimize regret in MDPs drawn from a distribution of MDPs that share the same state and action spaces. Therefore, the objective in meta-RL is to maximize the cumulative reward over the entire interaction period with an MDP, which may span multiple episodes, thereby achieving an optimal exploration-exploitation tradeoff.

$$\mathcal{J}(\theta) = \mathbb{E}_{M_i \sim \mathcal{M}} \Big[\sum_{t=0}^{H} \gamma^t \mathbb{E}_{(s_t, a_t) \sim \rho_i^{\pi_\theta}} [R_i(s_t, a_t)] \Big] \tag{1}$$

Here, meta-RL policy π_θ is interpreted as a "fast" or "inner" RL algorithm, which maps a trajectory, $\Upsilon = (s_0, a_0, r_0, ..., s_t)$, within an MDP M_i to an action a_t using either a recurrent neural network or a transformer network. $\rho_i^{\pi_\theta}$ is the state-action occupancy induced by the meta-RL policy in MDP M_i, and H is the length of the deployment, or *interaction budget*. The objective $\mathcal{J}(\theta)$ is maximized using a more conventional "slow" or "outer" deep RL algorithm due to the reformulation of the entire deployment period $t = 0...H$ with an MDP as a single (meta-)episode in the objective function,

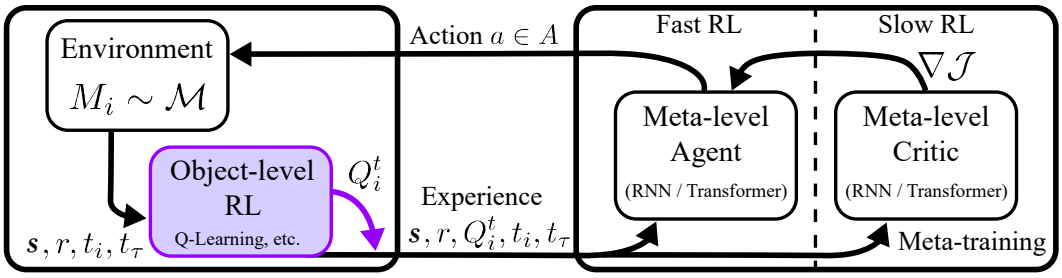

Figure 1: Overview diagram of RL^3. Black entities represent standard components from RL^2, and purple entities represent additions for RL^3. M_i is the current MDP; s is a state; r is a reward; t_i and t_τ are the amount of time spent experiencing the current MDP and current episode, respectively; Q_i^t is the Q-value estimate for MDP i after t actions; $\nabla \mathcal{J}$ is the policy gradient.

aimed at maximizing cumulative reward throughout this period. We will use the term "trajectory" (or "history"), denoted by Υ, to refer to a sequence of states, actions and rewards within a meta-episode of interaction with an MDP, M_i, which may include multiple episodes with trajectories $\{\tau_0, \tau_1, ...\tau_n\}$ within Υ. Figure 1 illustrates how these components interconnect.

One particularly clean way to conceptualize this problem is to recognize that the meta-RL problem may be written as a meta-level POMDP, where the hidden variable is the particular MDP (or task) at hand, M_i, which varies across meta-episodes. This framing, known as Bayesian RL [48], leverages the fact that augmenting the task-specific state s with belief over tasks $b(i)$ results in a Markovian meta-state $[s, b]$ for optimal action selection, a model known as the Bayes Adaptive MDP (or BAMDP) [40]. That is, this belief state captures all requisite information that is normally supplied by a trajectory for the purpose of acting. We will revisit this concept to develop intuition on the role of object-level Q-value estimates in the meta-RL value function.

4 RL^3

To address the limitations of black box meta-RL methods, we propose RL^3, a principled approach that leverages (1) the inherent generality of action-value estimates, (2) their ability to compress trajectories into useful summaries, (3) their direct actionability & asymptotic optimality, (4) their ability to inform task-identification, and (5) their relation to the optimal meta-value function, in order to enhance out-of-distribution (OOD) generalization and performance over extended horizons. The central, novel mechanism in RL^3 is an additional 'object-level' RL procedure executed within the meta-RL architecture, shown in Fig. 1, that computes task-specific Q-value estimates $Q_i^t(s_t)$ and state-action counts as supplementary inputs to the meta-RL policy in conjunction with the standard state-action-reward trajectories $(s_0, a_0, r_0, ..., s_t)$. The estimates and the counts are reset at the beginning of each meta-episode as a new task M_i is sampled from the MDP distribution $\mathcal{M}$. In all subsequent text, Q-value estimates used as input entail the inclusion of state-action counts as well. We now present a series of key insights informing our approach.

First, estimating the action-values is a key component in many **universal** RL algorithms, and asymptotically, they *fully* inform optimal behavior, *irrespective of the domain*. The strategies for optimal exploration-exploitation trade-off are domain-dependent and rely on historical data, yet many exploration approaches use estimated Q-values and some notion of counts *alone*, such as epsilon-greedy, Boltzmann exploration, upper confidence bounds (UCB/UCT) [49, 50], count-based exploration [51], curiosity based exploration [52] and maximum-entropy RL [3]. This creates a strong empirical case that relying on Q-value estimates and state-action counts for efficient exploration has an inherent generality.

Second, Q-estimates **summarize trajectories** of arbitrary length *and order* in one constant-size vector. The mapping from trajectories to Q-estimates is many-to-one, and any permutation of transitions ($\langle s, a, r, s' \rangle$ tuples) or episodes in a trajectory yield the same Q-estimates. Although this compression is lossy, it still "remembers" important aspects of the trajectory, such as high-return actions and goal positions (see Figure 2) since Q-estimates persist across object-level episodes. This simplifies the

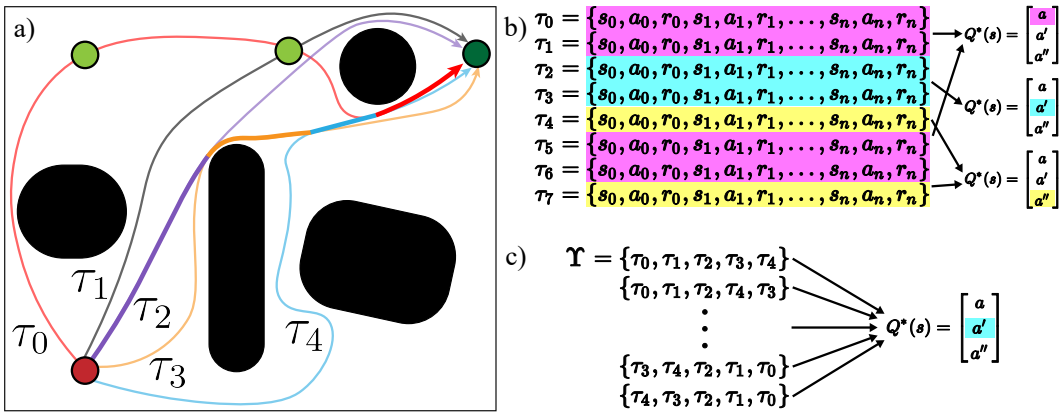

Figure 2: Sub-figure (a) shows a meta-episode in a shortest-path environment where the goal position (green circles) and the obstacles (black regions) may vary across tasks. In this meta-episode, after the meta-RL agent narrows its belief about the goal position of this task (dark-green circle) having followed a principled exploration strategy (τ_0), it explores potential shorter paths in subsequent episodes ($\tau_1, \tau_2, \tau_3, \tau_4$). Throughout this process, the estimated value-function $\hat{Q}^*$ implicitly "remembers" the goal position and previous paths traversed in a finite-size representation, and updates the shortest path calculation (highlighted in bold) using Bellman backups when paths intersect. Sub-figures (b) and (c) illustrate the many-to-one mapping of object- and meta-level data streams to Q-estimates, and thus their utility as compression and summarization mechanisms for meta-learning.

mapping the meta agent needs to learn as Q-estimates represent a smaller and more salient set of inputs in the long run compared to all possible histories that have the same implication.

Third, Q-estimates are **actionable**. Estimated using an in-built off-policy RL module, they explicitly represent the optimal exploitation policy for the current task given the data, insofar as the RL module is data-efficient, relieving the meta-RL agent from having to perform such computations inside the transformer/RNN. Over time, Q-estimates become increasingly reliable and directly indicate the optimal policy whereas processing raw data becomes more challenging. Fortunately, by incorporating Q-estimates the meta-RL agent can eventually ignore the history in the long run (or as time t approaches the interaction budget H) and simply exploit the Q-estimates by selecting actions greedily.

Fourth, Q-estimates are **excellent task discriminators** and serve as another line of evidence vis-à-vis maintaining belief over tasks. In a simple domain like Bernoulli multi-armed bandits [4], Q-estimates and action-counts combined are sufficient for Bayes-optimal behavior even without providing raw trajectory data – a result surprisingly unstated in literature to best of our knowledge (see Appendix A.1). However, Q-estimates and action-counts are not always sufficient for Bayes-optimal beliefs. For example, in Gaussian multi-armed bandits, the sufficient statistics include the variance in rewards for each action (see Appendix A.2). In more complex domains, it is hard to prove the sufficiency of Q-estimates regarding task discrimination. However, via empirical analysis in Appendix D, we argue that i) it is highly improbable for two tasks to have similar Q^* functions and ii) Q-estimates tend to become accurate task predictors in just a few steps. This implies that the meta-agent may use this finite summary for task inference rather than relying completely on arbitrarily long histories, potentially contributing to enhanced performance on long-horizon problems.

It can be theoretically argued that since the meta agent is a BAMDP *policy*, it is meta-trained to select greedy actions w.r.t. the BAMDP meta-value function and thus should not require constructing a task-specific plan internally. However, the optimality of the meta action-value function depends on implicitly (or explicitly in some approaches, such as [24, 27, 41, 42]) maintaining a Bayes-optimal belief over tasks in the transformer/RNN architecture. This may be challenging if the task distribution is too broad and the function approximator is not powerful enough to integrate the trajectories into Bayes-optimal beliefs, or altogether impossible if there is a distribution shift at meta-test time. This latter condition is common in practice and is a frequent target use case for

meta-RL systems. Incorporating task-specific Q-estimates gives the agent a simple alternative (even if not Bayes-optimal) line of reasoning to translate trajectories into actions. Incorporating Q-estimates thus **reduces susceptibility to distribution shifts** since the arguments presented above are domain independent.

Finally, Q-estimates often converge far more quickly than the theoretical rate of $\frac{1}{\sqrt{t}}$, allowing them to be useful in the short and medium term, since i) most real-world domains contain significant determinism, and ii) optimal meta-RL policies may represent active exploration strategies in which Q-estimates converge faster, or evolve in a manner leading to quicker task identification. This is intuitively apparent in shortest-path problems, as illustrated in Figure 2(a). In a deep neural network it is difficult to know exactly how Q-estimates will combine with state-action-reward histories when approximating the meta-value function. However, as we show below, we can rather straightforwardly write an equation for the meta-value of a given belief state in terms of these constituent streams of information, which may explain why this function is seemingly relatively easy to learn compared to predicting meta-values from histories alone.

4.1 Theoretical Justification

Here, we consider the interpretation of meta-RL as performing RL on a partially observable Markov decision process (POMDP) in which the partially observable state factor is the identity of the object-level MDP. All analysis assumes the infinite horizon setting. We will denote meta-level entities, belonging in this case to a POMDP, with an overbar. For example, we have a meta-level value function $\bar{V}$ and a meta-level belief $\bar{b}$. Thus, we can write an equation for the POMDP meta-level value function in its belief-MDP representation:

$$\bar{V}^*(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{\bar{s} \in \bar{S}} \bar{b}(\bar{s}) \bar{R}(\bar{s}, a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega} | \bar{b}, a) \bar{V}^*(\bar{b}') \Big]. \tag{2}$$

However, given that the only partially observable variable is the task, we can re-write this as

$$\bar{V}^*(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{M_i \in \mathcal{M}} \bar{b}(i) R_i(s, a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega} | \bar{b}, a) \bar{V}^*(\bar{b}') \Big], \tag{3}$$

where $\bar{b}(i)$ denotes the meta-level belief that the agent is operating in MDP M_i, and $R_i(s, a)$ is the immediate reward experienced by the agent if it executes action a in state s in MDP M_i. Here, $\bar{b}'$ may be calculated via the belief update as in §3.1. If the meta-level observation $\bar{\omega}$ includes Q-value estimates, the meta-level observation function $\bar{O}(\bar{\omega}|\bar{b}, a)$ thus gives the probability that a particular Q-estimate will be observed given an initial belief about the task identity $\bar{b}$ and an action a. Given this setup, we can rewrite the original BAMDP value function (Equation (3)) at time t in a manner that motivates our meta-RL system architecture.

$$\bar{V}^t(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{M_i \in \mathcal{M}} \bar{b}(i) R_i(s, a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega} | \bar{b}, a) \sum_{M_i \in \mathcal{M}} \bar{b}'(i) \sum_{s' \in S} T_i(s, a, s') (Q_i^t(s') + \varepsilon_i(\tau)) \Big]. \tag{4}$$

Equation (4) has two important features. First, $\bar{\omega}$ includes both trajectory τ and Q-estimate $Q^t(s, a)$. Second, the error in Q-estimate for the object-level MDPs is captured by $\varepsilon_i(\tau)$. This error is a function of the data (both amount and quality) seen by the agent so far, which can be summarized by the trajectory τ. We can see this error will diminish as $t \to \infty$, but in the short run, a function $f(\tau)$ could be learned to either estimate the error or replace the $(Q_i^t(s') + \varepsilon_i(\tau))$ term entirely. The increase in performance of RL^3 could be explained by either $f(\tau)$ being simpler to learn, or, given that Q-estimates are supplied directly as inputs, the resultant meta-agent behavior may be more robust to errors in estimates of $\varepsilon(\tau)$ than to errors in a more complicated approximation of $\bar{V}^*(\bar{b})$. Moreover, this composition benefits from the fact that observations become increasingly accurate Q-estimates in the limit, and the convergence rate for Q-estimates further suggests a natural, predictable rate of shifting reliance from $f(\tau)$ to $Q_i^t(s')$ as $t \to \infty$. However, we do not bake this structure into the network and instead let it implicitly learn how and how much to use the Q-estimates. It is important to note that as we are learning $\bar{O}$ as part of a black box model, mapping Q-estimates to tasks is a relatively straightforward function to learn compared to mapping trajectories to tasks, and as $t \to \infty$ and Q-estimates become more accurate, it is possible to reliably discriminate using ever smaller differences in Q-values. Moreover, knowing the exact task is not required to act optimally.

Algorithm 1 Value-Augmented MDP Wrapper Over a Discrete MDP

```
procedure RESETMDP(vamdp)
    vamdp.t ← 0; vamdp.τ ← 0
    vamdp.N[s, a] ← 0; vamdp.Q[s, a] ← 0   ∀s ∈ S, a ∈ A
    vamdp.rl ← INITRL()
    s = RESETMDP(vamdp.mdp)
    return ONEHOT(s) · Q[s] · N[s]
procedure STEPMDP(vamdp, a)
    s ← mdp.s
    r, s' ← STEPMDP(vamdp.mdp, a)
    d ← TERMINATED(vamdp.mdp)
    vamdp.t, vamdp.N[s, a], vamdp.τ ← += 1
    vamdp.Q ← UPDATERL(vamdp.rl, s, a, r, s', d)
    if d or vamdp.τ ≥ task_horizon then
        vamdp.τ ← 0
        s' ← RESETMDP(vamdp.mdp)
    return r, ONEHOT(s') · Q[s'] · N[s']
procedure TERMINATED(vamdp)
    return vamdp.t ≥ interaction_budget
```

4.2 Implementation

Implementing RL^3 involves simply replacing each MDP in the task distribution with a corresponding value-agumented MDP (or VAMDP) and solving the resulting VAMDP distribution using RL^2 . Each VAMDP has the same action space and reward function as the corresponding MDP. The value augmented state $\hat{s}_t \in S \times \mathbb{R}^k \times \mathbb{I}^k$ includes the object level state s_t, k real values and k integer values for the Q-estimates ($Q^t(s_t, a)$) and action counts ($N^t(s_t, a)$) respectively for each of the k actions. When the object-level state space S is discrete, s_t needs to be represented as an $|S|$-dimensional one-hot vector. Note that the value augmented state space is continuous. In the VAMDP transition function, the object-level state s has the same transition dynamics as the original MDP, while the dynamics of Q-estimates are a function of T, R, and the specific object-level RL algorithm used for Q-learning. An episode of the VAMDP spans the entire interaction period with the corresponding MDP, which may include multiple episodes of the MDP, as Q-estimates continue to evolve beyond episode boundaries. In code, a VAMDP RL environment is implemented as a wrapper over a given MDP environment. The pseudocode is provided in Algorithm 1 and additional implementation details, engineering tricks, and hyperparameters for RL^2 and RL^3 can be found in Appendix B.

5 Experiments

We compare RL^3 to a modified version of RL^2. First, we replace LSTMs with transformers in both the meta-actor and meta-critic for the purpose of mapping trajectories to actions and meta-values, respectively. This is done to maximize RL^2's ability to handle long-term dependencies instead of suffering from vanishing gradients. Moreover, RL^2-transformer trains significantly faster than RL^2-LSTM. Second, we include in the state space the total number of interaction steps and the total number of steps within each episode during a meta-episode (see Fig. 1). Third, we use PPO [2] for training the meta actor-critic, instead of TRPO [53]. These modifications and other minor-implementation details incorporate the recommendations made by Ni et al. [54], who show that model-free recurrent RL is competitive with other state-of-the-art meta RL approaches such as VeriBAD [27], if implemented properly. RL^3 simply applies the modified version of RL^2 to the distribution of value-augmented MDPs explained in section 4.2. Within each VAMDP, our choice of object-level RL is a model-based algorithm in the interest of sample efficiency – we estimate a tabular model of the environment and run finite horizon value iteration on the model to get Q-estimates.

In our test domains, each meta-episode involves procedurally generating an MDP according to a parameterized distribution, which the meta-actor then interacts with for a fixed budget H. This interaction might consist of multiple object-level episodes of variable length, each of which are no longer than a separate maximum task horizon. For a given experiment, each approach is trained

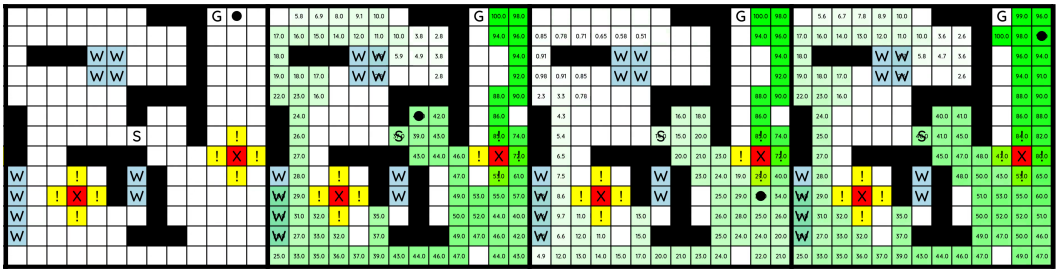

Figure 3: An RL^3 policy on a selected meta-episode visualized using a sequence of snapshots. 'S' is the starting tile, 'G' is the goal tile and the black circle shows the current position of the agent. Blue tiles marked 'W' are wet tiles. Wet tiles always lead to the agent slipping to one of the directions orthogonal to the intended direction of movement. Entering wet tiles yield an immediate reward of -2. Yellow tiles marked '!' are warning tiles and entering them causes -10 reward. Red tiles marked 'X' are fatally dangerous. Entering them ends the episode and leads to a reward of -100. Black tiles are obstacles. White tiles yield a reward of -1 to incentive the agent to reach the goal quickly. On all tiles other than wet tiles, there is a chance of slipping sideways with a probability of 0.2. The object-level state-values $v^t(s) = \max_a Q^t(s, a)$, as approximated by object-level RL, is represented using shades of green (and the accompanying text), where darker shades represent higher values.

on the same series of MDPs. For testing, each approach is evaluated on an identical set of 1000 MDPs distinct from the training MDPs. For testing OOD generalization, MDPs are generated from distributions with different parameters than in training. We select three task domains for our experiments, which cover a range of short-term, long-term, and complex dependencies.

Bernoulli Bandits: We use the same setup described in [4] with $k = 5$ arms. We experiment with $H = 100$ and $H = 500$ interaction budgets. To test OOD generalization, we generate bandit tasks by sampling success probabilities from $\mathcal{N}(0.5, 0.5)$.

Random MDPs: We use the same setup described in [4]. The MDPs have 10 states and 5 actions. The mean rewards and transition probabilities are drawn from a normal and a flat Dirichlet distribution, respectively. The task horizon is 10. To test OOD generalization, the rewards are generated deterministically and initialized from $\mathcal{U}(0, 2)$.

GridWorld Navigation: A set of navigation tasks in a 2D grid environment. We experiment with 11x11 (121 states) and 13x13 (169 states) grids. The agent always starts in the center of the grid and needs to navigate through obstacles to a single goal location. The grid also contains slippery tiles, fatally dangerous tiles and warning tiles surrounding the latter. See Fig. 3(a) for an example of a 13x13 grid. Instead of one-hot vectors, we use the 2D (x, y) grid location to represent agent state. To test OOD generalization, we vary parameters including the stochasticity of actions, density of obstacles and the number of dangerous tiles. For this domain, we consider an additional variation of RL^3, called RL^3-*coarse* where a given grid is partitioned into clusters of tiles (or abstract states), each of size 2. Abstract states are comprised only of adjacent, traversable cells, and are used *solely* for the purpose of estimating the object-level Q-values. Our goal is to test whether coarse-level Q-value estimates are still useful to the meta-RL policy. The domains and the abstraction strategy used for the RL^3-coarse approach are described in greater detail in Appendices E and B.3, respectively.

6 Results

In this section we present several experiments that demonstrate the (sometimes surprising) effectiveness of RL^3. Beyond matching or exceeding the performance of RL^2 in all test domains, RL^3 also shows better OOD generalization, which we attribute to the increased generality of the Q-value representation. More striking, the advantages of RL^3 increase significantly as horizons increase or domains become less stochastic. We hypothesize this is due to the increased accuracy of the object-level Q-value estimates in these cases. Last, we find that RL^3 performs well even with coarse-grained object-level RL over abstract states, showing minimal drop in performance in most cases and occasionally even increases, while the computational savings from abstraction are substantial.

Table 1: Test scores (mean $\pm$ standard error) for Bandits domain and the †OOD variation.

Budget H	**Random**	**RL**^{2}	**RL**^{3}	**RL**^{3} **(Markov)**
100	50.0	76.9 ± 0.6	77.5 ± 0.5	75.2 ± 0.5
500	250.2	392.1 ± 2.5	393.2 ± 2.7	391.75 ± 2.6
500†	249.0	430.2 ± 2.8	$\mathbf{434.9} \pm 2.8$	433.7 ± 2.8

Table 2: Test scores (mean $\pm$ standard error) for MDPs domain and the †OOD variation.

Budget H	**Random**	**RL**^{2}	**RL**^{3}
100	99.9	159.5 ± 0.8	158.9 ± 0.8
500	502.1	927.8 ± 3.7	926.9 ± 3.7
500†	501.7	772.8 ± 1.7	$\mathbf{775.9} \pm 1.7$
1000 (extrapolated)	1000.6	1871.8 ± 7.4	$\mathbf{1916.8} \pm 7.4$

We emphasize that the core of our approach, which is augmenting MDP states with action-value estimates, is not inherently tied to RL^2 and is orthogonal to most other meta-RL research. VAMDPs can be plugged into any base meta-RL algorithm with a reasonable expectation of improving it.

Bandits: Table 1 shows the results for the Bandits domains. For $H = 100$ and $H = 500$, both approaches perform comparably. However, the OOD generalization for RL^3 is slightly better. We also experiment with a Markovian version of RL^3, where we use a feed-forward neural network that observes only the Q-estimates and action-counts, since Q-estimates are sufficient for Bayes-optimal behavior in this domain. As expected, the results are similar to those with regular RL^3.

MDPs: Table 2 shows the results for the MDPs domains. Once again, for $H = 100$ and $H = 500$, both approaches perform comparably, and once again, OOD generalization is slightly better for RL^3. We suspect that for such short budgets on this highly stochastic domain, Q-estimates do not converge enough to be very useful. To test this hypothesis, we test both approaches by applying the models trained for $H = 500$ to $H = 1000$ (row 4 in table 3) using a moving window context for the transformer models (see Appendix B for details). Here, RL^3 generalizes significantly better, demonstrating the utility of Q-esimates when they are allowed to converge for more iterations. In fact, the score achieved by RL^3 when trained for $H = 500$ and tested on $H = 1000$ is similar to that of the original RL^2 implementation [4] when trained specifically for $H = 1000$.

Gridworlds: Table 3 shows the results for the Gridworld Navigation domain. On 11x11 grids, RL^3 significantly outperforms RL^2 . On 13x13 grids, the performance margin is even greater, showing that while RL^2 struggles with a greater number of states and a longer horizon, RL^3 can take advantage of the Q-estimates to overcome the challenge. We also test the OOD generalization of both approaches in different ways by increasing the obstacle density (DENSE), making actions on non-water tiles deterministic (DETERMINISTIC), increasing the number of wet 'W' tiles (WATERY), increasing the number of danger 'X' tiles (DANGEROUS) and having the goal only in the corners (CORNER). For the dense variation, RL^3 continues to outperform RL^2 . On the DETERMINISTIC variation, RL^2 gains 380 points, while RL^3 gains 654 points, which is likely because Q-estimates converge faster on this less stochastic MDP and therefore provide greater help to RL^3 . Conversely, in the WATERY variation, which is more stochastic, both RL^2 and RL^3 lose roughly equal number of points. It is worth noting that RL^3 still outperforms RL^2 on this variation. On the DANGEROUS variation, RL^3 loses relatively fewer points compared to RL^2 , and continues to outperform it. In each case, RL^3-coarse significantly outperforms RL^2 . In fact, it performs on par with RL^3, even outperforming it on CORNER variation, except on the canonical 13x13 case and its DETERMINISTIC version, where it scores about 90% of the scores for RL^3.

Fig. 3 shows a sequence of snapshots, from left to right, of a meta-episode where the trained RL^3 agent is interacting with an instance of a 13x13 grid. The first snapshot shows the agent just before reaching the goal for the first time. Prior to the first snapshot, the agent had explored many locations in the grid. The second snapshot shows the next episode just after the agent finds the goal, resulting in value estimates being updated using object-level RL for all visited states. Snapshot 3 shows the agent consequently using the Q-estimates to navigate to the goal presumably by choosing

Table 3: Test scores (mean $\pm$ standard error) for Gridworld domain and the †OOD variations.

Size, Budget, Variation	Random	RL^2	RL^3	RL^3-coarse
11x11, 250, None	-568.2	524.3 ± 21.7	$\mathbf{630.8} \pm 21.6$	$\mathbf{612.6} \pm 21.3$
13x13, 350, None	-621.6	583.6 ± 28.1	$\mathbf{901.9} \pm 27.2$	831.3 ± 27.7
13x13, 350, DENSE†	-663.8	383.8 ± 25.8	$\mathbf{690.7} \pm 27.9$	$\mathbf{673.6} \pm 27.5$
13x13, 350, DETERMINISTIC†	-637.8	959.9 ± 37.8	$\mathbf{1574.3} \pm 34.8$	1463.6 ± 36.0
13x13, 350, WATERY†	-632.4	513.2 ± 25.6	$\mathbf{826.0} \pm 27.0$	$\mathbf{822.1} \pm 27.5$
13x13, 350, DANGEROUS†	-1016.7	282.7 ± 28.4	$\mathbf{646.7} \pm 29.5$	$\mathbf{657.2} \pm 29.7$
13x13, 350, CORNER†	-638.8	318.7 ± 22.6	507.8 ± 23.1	$\mathbf{645.4} \pm 23.2$

high-value actions. The agent also explores several new nearby states for which it does not have Q-estimates. Snapshot 4 shows the final Q-value estimates. A short video of the Gridworld environment, showing both RL^2 and RL^3 agents solving a set of problem instances side-by-side, can be found at `https://youtu.be/eLA_S1BQUYM`.

Computation Overhead Considerations: As mentioned earlier, for implementing object-level RL, we use model estimation followed by finite-horizon value-iteration to obtain Q-estimates. The computation overhead is negligible for Bandits (5 actions, task horizon = 1) and very little for the MDPs domain (10 states, 5 actions, task horizon 10). For 13x13 Gridworlds (up to 169 states, 5 actions, task horizon = 350), RL^3 takes approximately twice the time of RL^2 per meta-episode. However, RL^3-coarse requires only 10% overhead while still outperforming RL^2 and retaining more than 90% of the performance of RL^3. This demonstrates the utility of state abstractions in RL^3 for scaling. Our implementation is available at `https://github.com/bhatiaabhinav/RL3`.

7 Conclusion

In this paper, we introduced RL^3, a principled hybrid approach that combines the strengths of normal RL and meta-RL. We demonstrated that RL^3 is capable of enhancing performance of meta-RL in long-horizon tasks and OOD tasks while maintaining the efficiency in the short term. Our results show that RL^3 outperforms RL^2 across a wide range of tasks, highlighting its potential as a powerful and versatile reinforcement learning framework. By providing a unified approach to tackle short-term and long-term dependencies, RL^3 paves the way for more robust and adaptable reinforcement learning agents in complex and diverse environments. In future work, we plan to explore extending RL^3 to handle continuous state spaces, possibly using state abstractions for discretization.

Acknowledgments

This work was supported in part by the National Science Foundation grant numbers 1954782, 2205153, and 2321786.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[4] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[5] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[6] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18:77–95, 2002.

[7] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

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

[9] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[10] Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[11] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*, 2018.

[12] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in Neural Information Processing Systems*, 31, 2018.

[13] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[14] 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.

[15] Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702, 2019.

[17] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? Towards understanding the effectiveness of MAML. *arXiv preprint arXiv:1909.09157*, 2019.

[18] Rituraj Kaushik, Timothée Anne, and Jean-Baptiste Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5269–5276, 2020.

[19] Ali Ghadirzadeh, Xi Chen, Petra Poklukar, Chelsea Finn, Mårten Björkman, and Danica Kragic. Bayesian meta-learning for few-shot policy adaptation across robotic platforms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1274–1280, 2021.

[20] Zhao Mandi, Pieter Abbeel, and Stephen James. On the effectiveness of fine-tuning versus meta-RL for robot manipulation. In *CoRL Workshop on Pre-training Robot Learning*, 2022.

[21] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[22] 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.

[23] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *International Conference on Autonomous Agents and MultiAgent Systems*, page 122–130, 2018.

[24] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[25] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-Q-learning. In *International Conference on Learning Representations*, 2020.

[26] Liqi Yan, Dongfang Liu, Yaoxian Song, and Changbin Yu. Multimodal aggregation approach for memory vision-voice indoor navigation with meta-learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5847–5854, 2020.

[27] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning. In *International Conference on Learning Representations*, 2020.

[28] Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International Conference on Machine Learning*, pages 6925–6935, 2021.

[29] David Emukpere, Xavier Alameda-Pineda, and Chris Reinke. Successor feature neural episodic control. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

[30] Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pages 1478–1487, 2022.

[31] Jane X Wang, Michael King, Nicolas Pierre Mickael Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, H. Francis Song, Gavin Buttimore, David P Reichert, Neil Charles Rabinowitz, Loic Matthey, Demis Hassabis, Alexander Lerchner, and Matthew Botvinick. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[32] Luckeciano C Melo. Transformers are meta-reinforcement learners. In *International Conference on Machine Learning*, pages 15340–15359, 2022.

[33] Allen Z Ren, Bharat Govil, Tsung-Yen Yang, Karthik R Narasimhan, and Anirudha Majumdar. Leveraging language for accelerated learning of tool manipulation. In *Conference on Robot Learning*, pages 1531–1541, 2023.

[34] Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. In *International Joint Conference on Artificial Intelligence*, pages 2794–2800, 2019.

[35] Zheng Xiong, Luisa M Zintgraf, Jacob Austin Beck, Risto Vuorio, and Shimon Whiteson. On the practical consistency of meta-reinforcement learning algorithms. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[37] Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[39] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[40] Michael O'Gordon Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002.

[41] Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta learning of exploration. *arXiv preprint arXiv:2008.02598*, 2020.

[42] Luisa M Zintgraf, Leo Feng, Cong Lu, Maximilian Igl, Kristian Hartikainen, Katja Hofmann, and Shimon Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In *International Conference on Machine Learning*, pages 12991–13001, 2021.

[43] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[44] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[45] Csaba Szepesvári. The asymptotic convergence-rate of Q-learning. In *Advances in Neural Information Processing Systems*, volume 10, 1997.

[46] Michael Kearns and Satinder Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Advances in Neural Information Processing Systems*, volume 11, 1998.

[47] Eyal Even-Dar, Yishay Mansour, and Peter Bartlett. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5(1), 2003.

[48] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 8(5-6):359–483, 2015.

[49] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

[50] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[51] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, pages 2753–2762, 2017.

[52] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*, 2019.

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

[54] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *International Conference on Machine Learning*, pages 16691–16723, 2022.

[55] Kevin Esslinger, Robert Platt, and Christopher Amato. Deep transformer Q-networks for partially observable reinforcement learning. In *NeurIPS Workshop on Foundation Models for Decision Making*, 2022.

A Proofs

A.1 Bayes Optimality of Q-value Estimates in Bernoulli Multi-armed Bandits

Given an instance of a Bernoulli multi-armed bandit MDP, $M_i \sim \mathcal{M}$, and trajectory data $\Upsilon_{1:T}$ up to time T, we would like to show that the probability $P(i|\Upsilon_{1:T})$ can be determined entirely from Q-estimates Q_i^T and action-counts N_i^T, as long as the initial belief is uniform or known.

In the following proof, we represent an instance i of K-armed Bandits as a K-dimensional vector of success probabilities $[p_{i1}, ..., p_{iK}]$, such that pulling arm k is associated with reward distribution $P(r = 1|i, k) = p_{ik}$ and $P(r = 0|i, k) = (1 - p_{ik})$.

Let the number of times arm k is pulled up to time T be N_{ik}^T and the number of successes associated with pulling arm k up to time T be q_{ik}^T. Given that this is an MDP with just a single state and task horizon of 1, the Q-estimate associated with arm k is just the average reward for that action, which is the ratio of successes to counts associated with that action i.e., $Q_{ik}^T = \frac{q_{ik}^T}{N_{ik}^T}$. To reduce the clutter in the notation, we will drop the superscript T for the rest of the subsection.

Now,

$$P(i|\Upsilon_{1:T}) = \alpha P(i) \cdot P(\Upsilon_{1:T}|i) \tag{5}$$

where α is the normalization constant, $P(i)$ is the prior probability of task i (which is assumed to be known beforehand), and $\Upsilon_{1:T}$ is the sequence of actions and the corresponding rewards up to time T. Assuming, without loss of generality, that the sequence of actions used to disambiguate tasks is a given, $P(\Upsilon_{1:T}|i)$ becomes simply the product of probabilities of reward outcomes up to time T, noting that the events are independent. Therefore,

$$\begin{aligned} P(\Upsilon_{1:T}|i) &= \prod_{k=1:K} \prod_{t=1:T} ([r_{tk} = 1]p_{ik} + [r_{tk} = 0](1 - p_{ik})) && (6) \\ &= \prod_{k=1:K} p_{ik}^{q_{ik}} \cdot (1 - p_{ik})^{N_{ik} - q_{ik}} && (7) \\ &= \prod_{k=1:K} p_{ik}^{Q_{ik} N_{ik}} \cdot (1 - p_{ik})^{N_{ik} - Q_{ik} N_{ik}} && (8) \end{aligned}$$

Putting everything together,

$$P(i|\Upsilon_{1:T}) = \alpha P(i) \cdot \prod_{k=1:K} p_{ik}^{Q_{ik} N_{ik}} \cdot (1 - p_{ik})^{N_{ik} - Q_{ik} N_{ik}} \tag{9}$$

Hence proven that N_i^T and Q_i^T are sufficient statistics to determine $P(i|\Upsilon_{1:T})$ in this domain, assuming that the prior over task distribution is known.

A.2 Non-Bayes Optimality of Q-value Estimates in Gaussian Multi-armed Bandits

Given an instance of a Gaussian multi-armed bandit MDP, $M_i \sim \mathcal{M}$, and trajectory data $\Upsilon_{1:T}$ up to time t, here we derive the closed-form expression of the probability $P(i|\Upsilon_{1:T})$ and show that it contains terms other than Q-estimates Q_i^t and action-counts N_i^t.

In the following proof, we represent an instance i of K-armed Bandits as a $2K$-dimensional vector of means and standard deviations $[\mu_{i1}, ..., \mu_{iK}, \sigma_{i1}, ..., \sigma_{iK}]$, such that pulling arm k is associated with reward distribution $P(r|i, k) = \frac{1}{\sqrt{2\pi}\sigma_{ik}} \exp(\frac{r - \mu_{ik}}{\sigma_{ik}})^2$.

Let the number of times arm k is pulled up to time T be N_{ik}^T. Given that this is an MDP with just a single state and task horizon of 1, the Q-estimate associated with arm k is just the average reward for that action Avg$[r_k]$ up to time T. To reduce the clutter in the notation, we will drop the superscript T for the rest of the subsection.

As in the previous subsection, we now compute the likelihood $P(\Upsilon_{1:T}|i)$.

$$P(\Upsilon_{1:T}|i) = \prod_{k=1:K} \prod_{t=1:T} \frac{1}{\sqrt{2\pi\sigma_{ik}}} \exp(\frac{r_{tk} - \mu_{ik}}{\sigma_{ik}})^2 \tag{10}$$

$$\log P(\Upsilon_{1:T}|i) = \sum_{k=1:K} \sum_{t=1:T} \frac{(r_{tk} - \mu_{ik})^2}{\sigma_{ik}^2} - \log(2\pi\sigma_{ik})/2 \tag{11}$$

$$= \sum_{k=1:K} N_{ik} \frac{\text{Avg}[(r_{tk} - \mu_{ik})^2]}{\sigma_{ik}^2} - N_{ik} \log(2\pi\sigma_{ik})/2 \tag{12}$$

$$= \sum_{k=1:K} N_{ik} \frac{\text{Avg}[r_k^2] - 2\mu_{ik}\text{Avg}[r_k] + \mu_{ik}^2}{\sigma_{ik}^2} - N_{ik} \log(2\pi\sigma_{ik})/2 \tag{13}$$

$$= \sum_{k=1:K} N_{ik} \frac{(\text{Var}[r_k] + \text{Avg}[r_k]^2) - 2\mu_{ik}\text{Avg}[r_k] + \mu_{ik}^2}{\sigma_{ik}^2} - N_{ik} \log(2\pi\sigma_{ik})/2 \tag{14}$$

$$= \sum_{k=1:K} N_{ik} \frac{\text{Var}[r_k] + (Q_{ik})^2 - 2\mu_{ik}Q_{ik} + \mu_{ik}^2}{\sigma_{ik}^2} - N_{ik} \log(2\pi\sigma_{ik})/2 \tag{15}$$

Therefore, computing this expression requires computing the variance in rewards, $\text{Var}[r_k]$ associated with each arm up to time T, apart from the the Q-estimates and action-counts.

Hence proven that Q-estimates and action-counts are insufficient to completely determine $P(i|\Upsilon_{1:T})$ in Gaussian multi-armed bandits domain.

A.3 Object-level Q-estimates and Meta-level Values

We now show a basic result, that the optimal meta-level value function is upper bounded by the object-level Q-value estimates in the limit.

Proof: Given a distribution of tasks $\mathcal{M}$, then for a given state s there exists a maximum object-level optimal value function $V^*_{max}(s)$, corresponding to some particular MDP $M_{max} \in \mathcal{M}$, such that for all MDPs $M_i \in \mathcal{M}$, $V^*_{max}(s) \geq V^*_i(s)$. Observe that the expected cumulative discounted reward experienced by the agent cannot be greater than the most optimistic value function over all tasks, since $\bar{V}^*(\bar{b})$ is a weighted average of individual value functions $V^{\pi_\theta}(s)$ which are themselves upper bounded by $V^*_{max}(s)$. Thus,

$$\arg\max_{M_i \in \mathcal{M}} V^*_i(s) \geq \bar{V}^*(\bar{b}) \quad \forall s \in S. \tag{16}$$

Next, we see that combining the asymptotic accuracy of Q-estimates and Equation (6) gives us

$$\lim_{t \to \infty} \arg\max_{a \in A, M_i \in \mathcal{M}} Q^t_i(s,a) \geq \bar{V}^*(\bar{b}) \quad \forall s \in S. \quad \square \tag{17}$$

Furthermore, if the meta-level observation $\bar{\omega}$ includes Q-value estimates, we have the following result. Given that the current task is represented by MDP M_i, then for any $\varepsilon > 0$, there exists $\kappa \in \mathbb{N}$ such that for $t \geq \kappa$,

$$\left| \arg\max_{a \in A} \left[Q^t_i(s,a) \right] - \bar{V}^*(\bar{b}) \right| \leq \varepsilon \quad \forall s \in S. \tag{18}$$

Proof: Let the set of observations $\bar{\Omega}$ be the set of possible object-level Q-value estimates. That is, $\bar{\Omega}$ contains $Q^t_i(s,a)$ for all s, a, t, and, most importantly, i. Thus, $\bar{\omega} = Q^t_i(s,a)$ for *known* s, a, and t, and an *unknown* i. The meta-level observation function $\bar{O}(\bar{\omega}|\bar{b}, a)$ thus gives the probability that a particular Q-value will be observed given an initial belief about the task identity $\bar{b}$ and an action a. We know that Q-estimates converge at a known rate in the limit. Thus, after t timesteps, their error is bounded (with high probability) by some ε.

Thus, observations become stable Q-value estimates in the limit. In practice, such Q-value estimates are excellent discriminators (see Appendix D) for the underlying task. Generally, there are two cases.

Case 1: $\bar{\omega}$ is unique to MDP M_i. In this case belief will collapse rapidly driving $b(j)$ terms to zero where $j \neq i$ and thus $\arg\max_{a \in A} Q_i^t(s,a) \approx \bar{V}^*(\bar{b})$.

Case 2: $\bar{\omega}$ is not unique. In this case, belief will not collapse to a single MDP. However, belief will still go to zero for tasks not commensurate with the observed Q-value, and the remaining n tasks will share belief equally since they cannot be disambiguated. Thus, the expression for the meta-level value will resemble $\sum_{i=1}^{n} \frac{1}{n} Q_i(s,a)$. Since all $Q_i(s,a)$ are identical, this will simplify to the $Q_i(s,a)$, where i may represent any of the (identical Q-valued) tasks with non-zero belief.

Knowing the exact task is not required to act optimally and achieve the optimal meta-level value function, so long as the Q-value estimates contain enough information to select actions. □

B Architecture

B.1 RL^2

Our modified implementation of RL^2 uses transformer decoders [36] instead of RNNs to map trajectories to action probabilities and meta-values, in the actor and the critic, respectively, and uses PPO instead of TRPO for outer RL. The decoder architecture is taken from [36] as it is, with 2 layers of masked multi-head attention. However, we use learned position embeddings instead of sinusoidal, which are followed by layer normalization. Our overall setup is similar to [55].

The transformer always looks at the entire trajectory up to the timestep t and outputs the corresponding meta-values $\bar{V}_1 ... \bar{V}_t$ (critic) or action probabilities $\pi_1 ... \pi_t$ (actor). An experience tuple in the trajectory at timestep t consists of the previous action a_{t-1}, latest reward r_{t-1}, state s_t, episode timestep τ and meta-episode timestep t, all of which are normalized to be in the range $[0, 1]$. In order to reduce inference complexity, say at timestep t, we append t new attention scores (corresponding to experience t w.r.t. the previous $t-1$ experience inputs) to a previously cached $(t-1) \times (t-1)$ attention matrix, instead of recomputing the entire $t \times t$ attention matrix. This caching mechanism is implemented for each attention head and reduces the inference complexity at time t from $\mathcal{O}(t^2)$ to $\mathcal{O}(t)$. Note that this caching mechanism is possible only when the input to the transformer is the entire trajectory instead of a moving window.

[A note on using a moving context to apply a model trained $H = 500$ to $H = 1000$: A naive implementation would use a moving window context of 500 timesteps. However, with this choice, the model, which is likely biased towards mostly exploiting near the end of the window, would only exploit from timesteps 501 through 1000 even though there is room for exploring more. To ameliorate this problem, we use a context length of 250 timesteps until timestep 750, and increase it gradually therafter to 500 timesteps towards the end. In our experiments, this heuristic rule led to a significant improvement over the naive method.]

Note that our results are not significantly better than those in the original RL^2 paper. However, these changes drastically reduce real-world training time.

B.2 RL^3

In RL^3, the only difference from RL^2 is the inclusion of a vector of Q-estimates (and a vector of action counts) for the corresponding state in the input tuples. As mentioned in section 4.2 in the main text, this was implemented in our code by simply transforming MDPs to VAMDPs using a wrapper, and running RL^2 thereafter. For RL^3-Markov, we use a dense neural network, with two hidden layers of 64 nodes each, with ReLU activation function.

For object-level RL, we use model estimation followed by value iteration (with discount factor $\gamma = 1$) to obtain Q-estimates. Transition probabilities and mean rewards are estimated using maximum likelihood estimation (MLE). Rewards for unseen actions in a given state are assumed to be 0. States are added to the model incrementally when they are visited, so that value iteration does not compute values for unvisited states. Moreover, value iteration is carried out only for task horizon iterations (`task_horizon` = 1, 10, 250, 350 for Bandits, MDPs, 11x11 Gridworlds and 13x13 Gridworlds, respectively), unless the maximum Bellman error drops below 0.01.

Table 4: RL^2/RL^3 Hyperparameters

Hyperparameter	**Value**
Learning rate (actor and critic)	0.0003 (Bandits, MDPs), 0.0002 (GW)
Adam $\beta 1, \beta 2, \epsilon$	$0.9, 0.999, 10^{-7}$
Adam weight decay (critic only)	10^{-2}
Batch size	32000
Nsteps	interaction_budget (H)
Number of parallel envs	batch_size / nsteps
Minibatch size	3200
Entropy regularization coefficient	0.001 (Bandits $H = 100$), 0.04 (GW), 0.01 otherwise
PPO Iterations	See training curves
Epochs per iteration	8
KL target	0.01
PPO clip ϵ	0.2
GAE λ	0.3
Discount factor γ	0.99
Decoder layers	2
Attention heads per decoder layer	4
Activation function	gelu
Decoder size (d_model)	64

B.3 RL^3-coarse

During model estimation in RL^3-coarse, concrete states in the underlying MDP are incrementally clustered into abstract states as they are visited. When a new concrete state is encountered, its abstract state ID is set to that of a previously visited state within a 'clustering radius', unless the previous state is already part of a full cluster (determined by a maximum 'cluster size' parameter). If multiple old states satisfy the criteria, the ID of the closet old state is chosen. If there are no old states that satisfy the criteria, then the new state is assigned to a new abstract state ID, increasing the number of abstract states in the model.

The mechanism for learning the transition function and the reward function in the abstract MDP is the same as before. For estimating Q-values for a given concrete state, value iteration is carried out on the abstract MDP and the Q-values of the corresponding abstract state are returned.

For our Gridworld domain, we chose a cluster size of 2 and a clustering radius such that only non-diagonal adjacent states are clustered (Manhattan radius of 1).

It is worth noting that this method of deriving abstractions does not take advantage of any structure in the underlying domain. However, this simplicity makes it general purpose, efficient, and impartial, while still leading to excellent performance.

C Training Curves

Figs. 4, 5, and 6 show the training curves for Bandits, MDPs, and Gridworld environments, respectively, across 4 random seeds. The results in the main text may differ slightly since the actor models were evaluated greedily, whereas the training curves reflect the actors' stochastic policies.

We ran each experiment on an NVIDIA GeForce RTX 2080 Ti GPU, which took approximately 12-24 hours for Bandits and MDPs (H=100); and took approximately 3-4 days for other domains.

D Additional Analysis

RL^3 relies on the discriminatory power of data-sparse Q-estimates that, though imperfect, produce reasonable signals for the meta-learner. Here, we test this claim more thoroughly with 3 analyses.

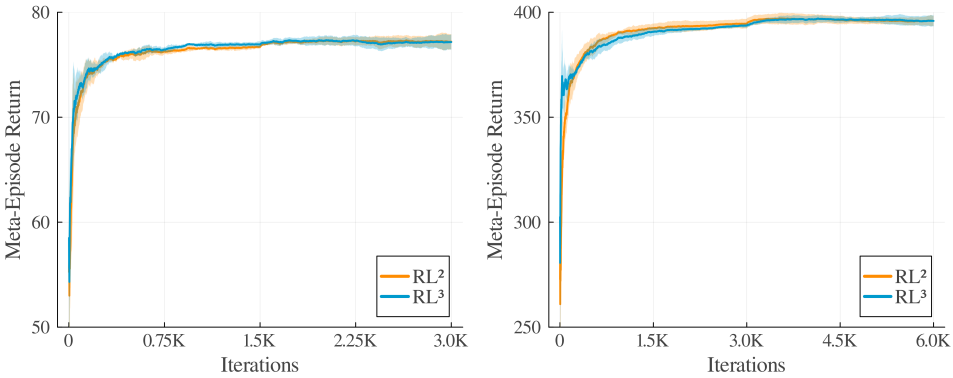

Figure 4: Average meta-episode return vs iterations for Bandits $H = 100$ and $H = 500$

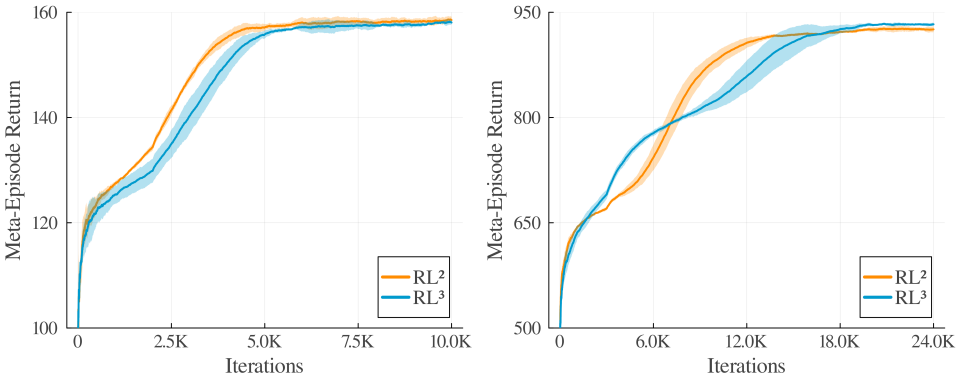

Figure 5: Average meta-episode return vs iterations for MDPs $H = 100$ (*left*) and $H = 500$ (*right*)
.

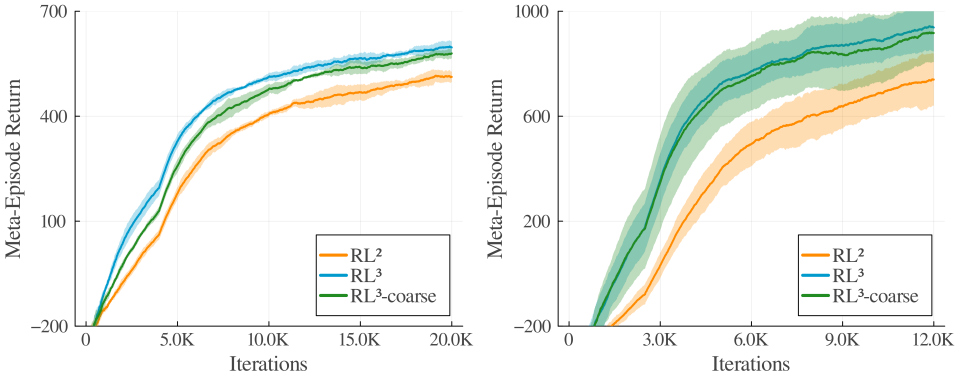

Figure 6: Average meta-episode return vs iterations for Gridworld 11x11 (*left*) and 13x13 (*right*).

D.1 Requirements for a Unique Q-value

Throughout, we assume states and actions are fixed. Below, we simply show that if the transition(reward) function is fixed, then 2 Q-tables will be identical if and only if both reward(transition) functions are also equal. First we have same Q $\implies$ same reward.

$$Q_1^*(s,a) = R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') \quad (19)$$

$$Q_2^*(s,a) = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_2^*(s',a') \quad (20)$$

$$R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_2^*(s',a') \quad (21)$$

$$R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') \quad (22)$$

$$R_1(s,a) = R_2(s,a) \quad (23)$$

Now, if two MDPs have the same reward function and same transition function, they are the same MDP and will have the same optimal solution. So, same reward $\implies$ same Q.

Since encountering similar Q-tables is thus dependent on both transitions and rewards "balancing" each other, the question is then for practitioners: How likely are we to get many MDPs which all appear to have very similar Q-tables?

D.2 Empirical Test using Max Norm

Given an MDP with 3 states and 2 actions, we want to find the probability that $||Q_1 - Q_2||_\infty < \delta$, where Q_1 and Q_2 are 6-entry (3 states $\times$ 2 actions) Q-tables. The transition and reward functions are drawn from distributions parameterized by α and β, respectively. Transition probabilities are drawn from a Dirichlet distribution, Dir(α), and rewards are sampled from a normal distribution, $\mathcal{N}(1, \beta)$. In total, we ran 3 combinations of α and β, each with 50,000 MDPs, a task horizon of 10, and $\delta = 0.1$. To get the final probability, we test all $((50,000-1)^2)/2$ non-duplicate pairs and count the number of max norms less than δ.

Results: For $\alpha = 1.0$, $\beta = 1.0$, we found the probability of a given pair of MDPs having duplicate Q-table to be $\epsilon = 2.6 \times 10^{-9}$. For $\alpha = 0.1$, $\beta = 1.0$, which is a more deterministic setting, we found $\epsilon = 4.6 \times 10^{-9}$. Further, with $\alpha = 0.1$, $\beta = 0.5$, where rewards are more closely distributed, we found $\epsilon = 1.1 \times 10^{-7}$. Overall, we can see that even for a very small MDP, the probability of numerically mistaking one Q-table for another is vanishingly small.

D.3 Predicting Task Families

The natural uniqueness of Q-values is encouraging, but max norm is not a very sophisticated metric. Here, we test whether a very simple multi-class classifier (1 hidden layer of 64 nodes), can accurately identify individual tasks based on their Q-table estimates. Moreover, we track how the accuracy improves as a function of the number of steps taken within the MDP. In this experiment, the same random policy is executed in each MDP for 50 time steps. As before, our MDPs have 3 states and 2 actions.

We instantiate 10,000 MDPs whose transition and reward functions are drawn from the same distribution as before: transitions from a Dirichlet distribution with $\alpha = 0.1$ and rewards sampled from a normal distribution $N(1, 0.5)$. Thus, the task is a classification problem with 10,000 classes. *A priori*, this task seems relatively difficult given the number of tasks and the parameters chosen for the distributions. Fig. 7 shows a compelling result given the simplicity of the model and the relative difficulty of the task. Clearly, Q-tables, even those built from only 20 state-action pairs, provide a high level of information w.r.t. task identification. And this is for a *random* policy. In principle, the meta-learner could follow a much more deliberate policy that actively disambiguates trajectories such that the Q-estimates evolve in a way that leads to faster or more reliable discrimination.

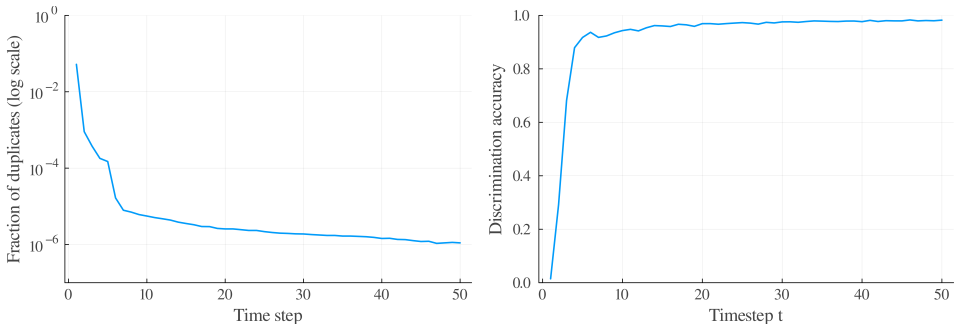

Figure 7: The discriminatory power of Q-estimates. *Left*: Fraction of δ-duplicates, with $\delta = 0.1$, as a function of time steps in a sample of 5,000 of random MDPs. *Right*: Accuracy of a simple multi-class classifier in predicting task ID given Q-table estimates as function of time step. Both figures are generated using the same policy

E Domain Descriptions

E.1 Bernoulli Multi-Armed Bandits

We use the same setup described in [4]. At the beginning of each meta-episode, the success probability corresponding to each arm is sampled from a uniform distribution $U(0, 1)$. To test OOD generalization, we sample success probabilities from $\mathcal{N}(0.5, 0.5)$

E.2 Random MDPs

We use the same setup described in [4]. The MDPs have 10 states and 5 actions. For each meta-episode, the mean rewards $R(s, a)$ and transition probabilities $T(s, a, s')$ are initialized from a normal distribution ($\mathcal{N}(1, 1)$) and a flat Dirichlet distribution ($\alpha = 1$), respectively. Moreover, when an action a is performed in state s, a reward is sampled from $\mathcal{N}(R(s, a), 1)$. To test OOD generalization, the reward function is made deterministic and initialized from $\mathcal{U}(0, 2)$

Each episode begins at state $s = 1$ and ends after `task_horizon` = 10 time steps.

E.3 Gridworlds

A set of navigation tasks in a 2D grid environment. We experiment with 11x11 (121 states) and 13x13 (169 states) grids. The agent always starts in the center of the grid and needs to navigate through obstacles to a single goal location. The goal location is always at a minimum of `min_goal_manhat` Manhattan distance from the starting tile. The grid also contains slippery wet tiles, fatally dangerous tiles and warning tiles surrounding the latter. There are `num_obstacle_sets` set of obstacles, and each obstacle set spans `obstacle_set_len` tiles, in either horizontal or vertical configuration. There are `num_water_sets` set of wet regions and each wet region always spans `water_set_length`, in either a horizontal or vertical configuration. Entering wet tiles yields an immediate reward of -2. There are `num_dangers` danger tiles and entering them ends the episode and leads to a reward of -100. Warning tiles always occur as a set of 4 tiles non-diagonally surrounding the corresponding danger tiles. Entering warning tiles causes -10 reward. Normal tiles yield a reward of -1 to incentivize the agent to reach the goal quickly. On all tiles, there is a chance of slipping sideways with a probability of 0.2, except for wet tiles, where the probability of slipping sideways is 1.

The parameters for our canoncial 11x11 and 13x13 gridworlds are: `num_obstacle_sets` = 11, `obstacle_set_len` = 3, `num_water_sets` = 5, `water_set_length` = 2, `num_dangers` = 2, and `min_goal_manhat` = 8. The parameters for the OOD variations are largely the same and the differences are as follows. For DETERMINISTIC variation, the slip probability on non-wet tiles is 0. For DENSE variation, `obstacle_set_len` is increased to 4. For WATERY variation, `num_water_sets` is increased to 8. For DANGEROUS variation, `num_dangers` is increased to 4. For CORNER variation, `min_goal_manhat` is set to 12, so that the goal is placed on one of the corners of the grid.

There is no fixed task horizon for this domain. An episode ends when the agent reaches the goal or encounters a danger tile. Therefore, an episode can last up to 250 steps in 11x11 gridworlds, and up to 350 steps in 13x13 gridworlds.

When a new grid is initialized at the beginning of each meta-episode, it is ensured that the optimal un-discounted return within 100 time steps of the meta-episode is between 50 and 100. This is to ensure that the grid both has a solution and is not trivial.