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
{abhinavbhati,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 can be slow to converge, require a large amount of data, and they have difficulty generalizing beyond the task they were trained on. 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. 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 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.

Preprint. Under review.

Our approach, RL^3, leverages both the asymptotic optimality of traditional RL as well as the utility of Q-value estimation as a compression and summarization mechanism within meta-RL in order to improve performance. The key idea is to perform Q-value estimation using traditional RL at object-level for each task experienced during training and augment the trajectory data with the Q-value estimate as an input to the meta-learner, represented as a transformer network mapping trajectories to actions. 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 increasingly accurate summarizations provided by the Q-values. In this work, RL^3 is implemented by injecting Q-values 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-values 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* domain used in previous work [4, 5] as well as on a more challenging custom Gridworld domain that requires long-term reasoning. We also show theoretically that Q-value injections provide an actionable input that is 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 models [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] or black box models [21, 4, 22, 23, 5, 24, 25, 26, 27, 28, 29, 30]. Parameterized policy gradient (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) [21, 4, 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 (histories) with Q-value estimates, and directly allows us to improve several weaknesses of these meta-RL approaches.

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. 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], 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). 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, 40], or random network distillation on belief states (HyperX) [41]. 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-values 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 Markov Decision Processes

A *Markov decision process* (MDP) is a model for reasoning in fully observable, stochastic environments [42], defined as a tuple $M = \langle S, A, T, R \rangle$. S is a finite set of states; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ represents the probability of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$; $R : S \times A \times S \to \mathbb{R}$ represents the immediate reward of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$. A solution to an MDP is a policy $\pi : S \to A$ indicating that an action $\pi(s) \in A$ should be performed in a state $s \in S$. A policy π induces a value function $V^\pi : S \to \mathbb{R}$ representing the expected discounted cumulative reward $V^\pi(s) \in \mathbb{R}$ for each state $s \in S$ given a discount factor $0 \leq \gamma < 1$. An optimal policy π^* maximizes the value function for every state $s \in S$ by satisfying the Bellman optimality equation $V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$. γ is the discount factor. Given a(n) (optimal) value function, a(n) (optimal) policy may be derived. When an MDP is fully defined, we may solve for the optimal value function using dynamic programming.

A *partially observable Markov decision process* (POMDP) extends MDPs to settings with partially observable states. A POMDP is described as a tuple $\langle S, A, T, R, \Omega, O \rangle$. S, A, T, R are as in an MDP. Ω is the set of possible observations. $O : S \times A \times \Omega \to [0, 1]$ is an observation function representing the probability of getting observation ω after performing action a and transitioning to state s'. To represent its uncertainty, the agent maintains a belief state $b \in \Delta^{|S|}$, a probability distribution over all states. After performing an action $a \in A$ and making an observation $\omega \in \Omega$, the agent updates its current belief state $b \in B$ to a new belief state $b' \in B$ using the belief state update equation $b'(s'|b, a, \omega) = \alpha O(a, s', \omega) \sum_{s \in S} T(s, a, s')b(s)$, where α is the normalization constant $\alpha = Pr(\omega|b, s)^{-1}$. POMDPs may also be represented as continuous-state belief-MDPs.

3.2 Reinforcement Learning

The goal of reinforcement learning (RL) is to learn an optimal policy when the transition function and/or the reward function of the MDP are unknown, using a stream of experience data comprised of actions, state transitions, and reward feedback. This is often done by incrementally estimating the (optimal) value of performing action a in state s, denoted by $Q^*(s, a)$ [43]. In large or continuous state settings, it is popular to fit deep neural networks to represent the action-value functions [1]. We denote the vector representing the Q-values of all available actions at state s, $Q(s)$, and when describing this estimate after t state-action-reward tuples have been observed, $Q^t(s)$. Q-learning, under some weak assumptions, is known to converge asymptotically [44], and thus we can write

$$\lim_{t \to \infty} \frac{\arg\max_{a \in A} Q^t(s, a)}{V^*(s)} = 1 \quad \forall s \in S. \tag{1}$$

In addition to these asymptotic results, there are also strong results on the maximum error between the estimated Q-values and the optimal value function V^* as a function of the number of tuples observed [45, 46, 47]. This difference is typically expressed as the maximum pairwise difference between these functions when evaluated at all states s, which we denote as the ∞-norm, or $|| \arg\max_{a \in A} Q^t(\cdot, a) - V^*(\cdot)||_\infty$. There are many underlying assumptions that vary between analyses, but as a rough general statement for a fixed MDP, we have the result that

$$|| \arg\max_{a \in A} Q^t(\cdot, a) - V^*(\cdot)||_\infty \propto \sqrt{\frac{\log(\log(t))}{t}} \approx \frac{1}{\sqrt{t}}. \tag{2}$$

Most results of this form technically only hold with probability $1 - \delta$, for some $\delta \in (0, 1)$, but for the purposes of this paper, the intuition provided by this relation is sufficient. The objective of RL algorithms is typically measured by the expected utility of the resultant policy, and does not include the data or computation cost of learning the value function, which may be substantial.

3.3 Meta Reinforcement Learning

While highly performant RL methods involve efficient, fixed exploration strategies to produce near-optimal Q-value estimates using as little data as possible, meta reinforcement learning (meta-RL) seeks to learn a meta-level policy for optimal exploration-exploitation tradeoff within MDP models drawn from a distribution, where the MDPs share the same state space S and action space A while having potentially different transition and reward dynamics. Thus, the objective function explicitly represents the time taken to explore in a new problem as missed reward. This results in an objective function similar to

$$\mathcal{J}(\theta) = \mathbb{E}_{M_i \sim \mathcal{M}}\Big[\mathbb{E}_{\mathcal{D}}\Big[\sum_{\tau \in \mathcal{D}_{0:H}} j_{\pi(\theta)}(\tau)\Big|\pi_\theta = f(\mathcal{D}), M_i\Big]\Big], \tag{3}$$

where $j_{\pi(\theta)}(\tau)$ is the cumulative discounted reward experienced by an agent running a reinforcement learning policy π parameterized by θ on a trajectory τ. In our case, θ are the weights of an end-to-end transformer network that represents the meta-level policy, interpreted as an object-level reinforcement learning algorithm (inner RL). $f(\mathcal{D})$ is the meta-trained function on trajectories $\mathcal{D}$ (outer RL). For this, we use Proximal Policy Optimization (PPO) [2]. H is the total length of the deployment. The inner expectation represents the distribution of possible trajectories. The outer expectation represents the distribution of tasks, where M_i is a specific MDP drawn from distribution $\mathcal{M}$. The process of optimizing for a specific task is what we call reinforcement learning, or, in the meta-RL setting, adaptation. We also refer to this process as the 'inner loop' since it is typically run many times while the meta-RL system is training. Optimizing θ is known as meta-training, or the 'outer loop'. Figure 1 shows how these pieces fit together.

One particularly clean way to conceptualize this problem is to recognize that the meta-RL problem may be written as a partially observable Markov decision process (POMDP), where the hidden variable is the particular MDP (or task) at hand. This framing, known as Bayesian RL [48], leverages the fact that augmenting the observations with belief over tasks results in a Markovian (meta-) state for optimal action selection, a model known as Bayes Adaptive MDP (or BAMDP) [49]. 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 the asymptotic optimality of traditional RL and the ability of Q-values to compress and summarize experience histories within meta-RL in order to improve out-of-distribution (OOD) generalization and performance on long-horizon tasks while remaining data efficient. The key idea behind RL^3 is to perform an additional 'object-level' RL routine within the meta-RL framework, shown in Fig. 1, that provides Q-value estimates to the black box meta-learner alongside the typical state-action-reward histories. These Q-value estimates have several useful properties.

First, they converge to the actual value of the meta-problem in the limit. Second, Q-values offer a way to compress trajectories of arbitrary length to one constant-size vector. This can reduce the difficulty meta-RL approaches often face with long-horizon tasks by simplifying the learning problem, since many unique histories may all produce similar Q-values. Third, Q-values can implicitly differentiate between tasks, at least insofar as differentiation brings greater utility, which is a consequence of Q-value convergence in the limit. Fourth, Q-values are directly actionable, requiring little intermediate reasoning to translate into action selection. Finally, no extra data is required to produce these estimates, only some extra computation. In principle, these properties allow the meta-learner to learn how to optimally fuse raw trajectory data with the summarizations provided by the Q-values.

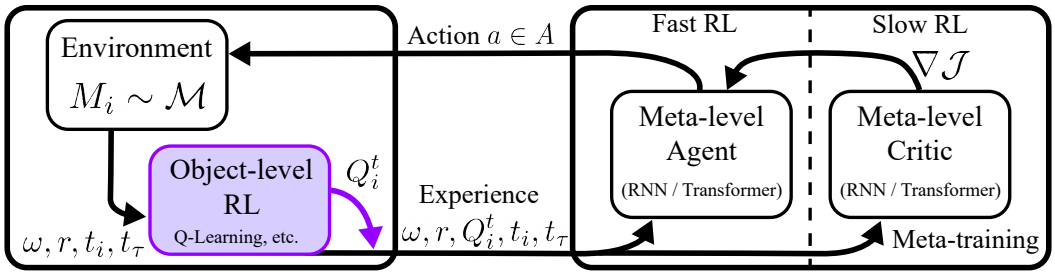

Figure 1: Overview diagram of RL^3. Black entities represent standard components from RL^2, and purple entities represent additions for RL^3. M_i is the current MDP; ω is an observation; r is a reward; t_i and t_τ are the amount of time spent experiencing the current MDP and current trajectory, respectively;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

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:

$$\bar{V}^*(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{\bar{s} \in \bar{S}} \bar{b}(\bar{s}) \bar{R}(\bar{s}, a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega} | \bar{b}, a) \bar{V}^*(\bar{b}') \Big]. \tag{4}$$

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

$$\bar{V}^*(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{M_i \in \mathcal{M}} \bar{b}(i) R_i(s, a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega} | \bar{b}, a) \bar{V}^*(\bar{b}') \Big], \tag{5}$$

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. 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,

$$\arg\max_{M_i \in \mathcal{M}} V^*_i(s) \geq \bar{V}^*(\bar{b}) \qquad \forall s \in S. \tag{6}$$

Next, we see that combining Equations (1) and (6) gives us

$$\lim_{t \to \infty} \arg\max_{a \in A, M_i \in \mathcal{M}} Q^t_i(s, a) \geq \bar{V}^*(\bar{b}) \qquad \forall s \in S. \quad \square \tag{7}$$

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 $t \geq \kappa$,

$$\Big| \arg\max_{a \in A} \Big[Q^t_i(s, a) \Big] - \bar{V}^*(\bar{b}) \Big| \leq \varepsilon \qquad \forall s \in S. \tag{8}$$

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 from (1) and (2) that Q-values 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 C) 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. □

Here, it is important to note that as we are learning $\bar{O}$ as part of a black box model, mapping Q-values to tasks is a relatively straightforward function to learn. Moreover, as $t \to \infty$, it is possible to discriminate using ever smaller differences in Q-values. Given these insights, we can rewrite the original BAMDP value function at time t (Equation (5)) in a manner that motivates our meta-RL system architecture.

$$\bar{V}^t(\bar{b}) = \arg\max_{a \in A} \Big[\sum_{M_i \in \mathcal{M}} \bar{b}(i) R_i(s,a) + \gamma \sum_{\bar{\omega} \in \bar{\Omega}} \bar{O}(\bar{\omega}|\bar{b},a) \sum_{M_i \in \mathcal{M}} \bar{b}'(i) \sum_{s' \in S} T_i(s,a,s')(Q_i^t(s') + \varepsilon_i(\tau)) \Big]. \tag{9}$$

Equation (9) has two important features. First, $\bar{\omega}$ may include both trajectory τ and Q-value $Q^t(s,a)$. Second, the error in Q-value 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. This is very similar to what current meta-RL systems are designed to do. The convergence rate for Q-values from Equation (2) further suggests a natural rate of shifting reliance from $f(\tau)$ to $Q_i^t(s')$ as $t \to \infty$.

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 A.

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]

Algorithm 1 Value-Augmented MDP Wrapper Over a Discrete MDP

procedure RESETMDP(vamdp)
  vamdp.$t \leftarrow 0$; vamdp.$\tau \leftarrow 0$
  vamdp.$N[s,a] \leftarrow 0$; vamdp.$Q[s,a] \leftarrow 0 \quad \forall s \in S, a \in A$
  vamdp.rl $\leftarrow$ INITRL()
  $s =$ RESETMDP(vamdp.mdp)
  return ONEHOT$(s) \cdot Q[s] \cdot N[s]$
procedure STEPMDP(vamdp, a)
  $s \leftarrow$ mdp.s
  $r, s' \leftarrow$ STEPMDP(vamdp.mdp, a)
  $d \leftarrow$ TERMINATED(vamdp.mdp)
  vamdp.t, vamdp.$N[s,a]$, vamdp.$\tau \leftarrow +\!= 1$
  vamdp.$Q \leftarrow$ UPDATERL(vamdp.rl, s, a, r, s', d)
  if d **or** vamdp.$\tau \geq$ `task_horizon` **then**
    vamdp.$\tau \leftarrow 0$
    $s' \leftarrow$ RESETMDP(vamdp.mdp)
  return r, ONEHOT$(s') \cdot Q[s'] \cdot N[s']$
procedure TERMINATED(vamdp)
  return vamdp.$t \geq$ `interaction_budget`

for training the meta actor-critic, instead of TRPO [50]. To implement RL^3, we simply apply 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-values.

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 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. 2(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 D and A.3, respectively.

6 Results

In this section we show results of 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

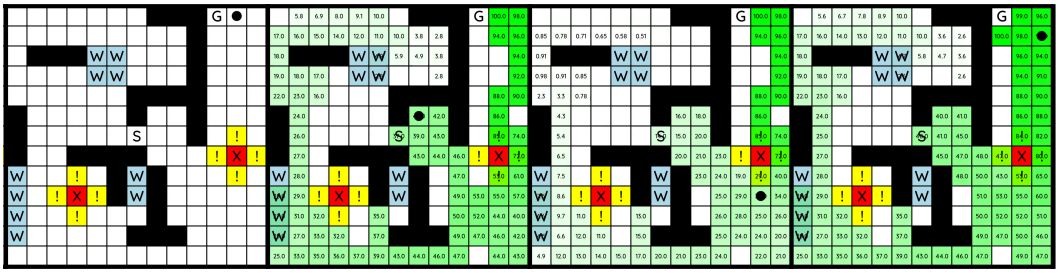

Figure 2: An RL^3 policy on a selected meta-episode visualized using a sequence of snapshots. 'S' is the starting tile, 'G' is the goal tile and the black circle shows the current position of the agent. Blue tiles marked 'W' are wet tiles. Wet tiles always lead to the agent slipping to one of the directions 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.

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.

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.

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-values 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 A for details). Here, RL^3 generalizes significantly better, demonstrating the utility of Q-values 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.

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

Table 1: Test scores (mean $\pm$ standard error) for Bandits domain.

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 (OOD)	501.7	772.8 ± 1.7	$\mathbf{775.9} \pm 1.7$
1000 (extrapolated)	1000.6	1871.8 ± 7.4	$\mathbf{1916.8} \pm 7.4$

Table 2: Test scores (mean $\pm$ standard error) for MDPs domain.

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	$\mathbf{645.4} \pm 23.2$

Table 3: Test scores (mean $\pm$ standard error) for Gridworld Navigation domain.

Fig. 2 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 states visited so far. Snapshot 3 shows the agent consequently utilizing the value estimates to navigate to the goal presumably by choosing high-value actions. The agent also explores several new states for which it does not have Q-value estimates for near states with existing, high Q-values. Snapshot 4 shows the final Q-value estimates.

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 Julia 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 addressing the limitations of RL^2 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, perhaps using state abstractions for discretization.

8 Acknowledgements

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

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. PMLR, 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. URL `https://openreview.net/forum?id=B1DmUzWAW`.

[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. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 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. URL `https://openreview.net/forum?id=Sk2u1g-0-`.

[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. PMLR, 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 *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5269–5276. IEEE, 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 *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1274–1280. IEEE, 2021.

[20] Zhao Mandi, Pieter Abbeel, and Stephen James. On the effectiveness of fine-tuning versus meta-RL for robot manipulation. In *CoRL 2022 Workshop on Pre-training Robot Learning*, 2022. URL `https://openreview.net/forum?id=21TVvjhOkV`.

[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 *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 122–130, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[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. URL `https://openreview.net/forum?id=SJeD3CEFPH`.

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

[27] L Zintgraf, K Shiarlis, M Igl, S Schulze, Y Gal, K Hofmann, and S Whiteson. Varibad: a very good method for bayes-adaptive deep rl via meta-learning. *Proceedings of ICLR 2020*, 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. PMLR, 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. URL `https://openreview.net/forum?id=e1Q2_jaE08J`.

[30] Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pages 1478–1487. PMLR, 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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL `https://openreview.net/forum?id=eZu4BZxlRnX`.

[32] Luckeciano C Melo. Transformers are meta-reinforcement learners. In *International Conference on Machine Learning*, pages 15340–15359. PMLR, 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. PMLR, 2023.

[34] Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 2794–2800. AAAI Press, 2019. ISBN 9780999241141.

[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. URL `https://openreview.net/forum?id=xwQgKphwhFA`.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 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 *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL `https://aclanthology.org/D14-1179`.

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

[41] 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. PMLR, 2021.

[42] Richard Bellman. On the theory of dynamic programming. *National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[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. *Advances in neural information processing systems*, 10, 1997.

[46] Michael Kearns and Satinder Singh. Finite-sample convergence rates for q-learning and indirect algorithms. *Advances in neural information processing systems*, 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, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[49] Michael O'Gordon Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

[50] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[51] Kevin Esslinger, Robert Platt, and Christopher Amato. Deep transformer q-networks for partially observable reinforcement learning. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL `https://openreview.net/forum?id=DrzwyQZNJz`.

A Architecture

A.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 [51].

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.

A.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 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.

A.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

Hyperparameter H	**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 Bonus	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

Table 4: RL^2 Hyperparameters

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.

B Training Curves

Figs. 3, 4, and 5 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.

C 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.

C.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.

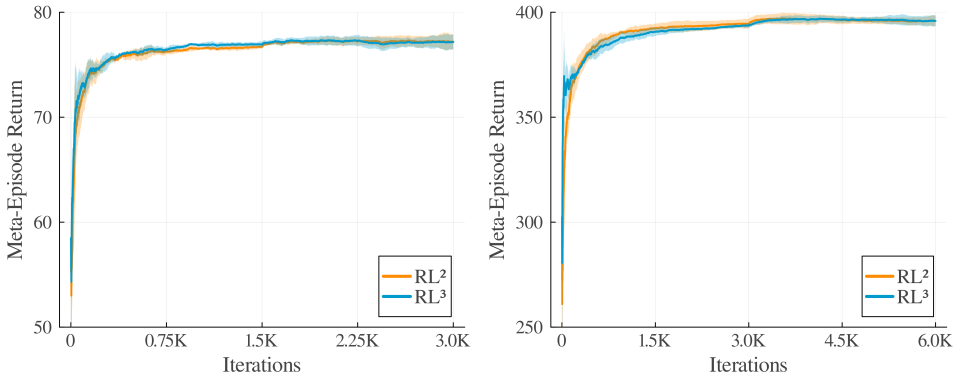

Figure 3: Average Meta-Episode Return vs Iterations For Bandits $H = 100$ and $H = 500$

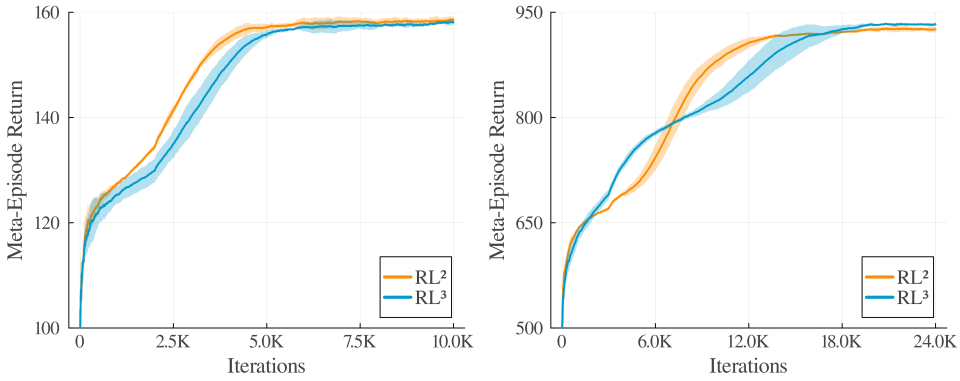

Figure 4: Average Meta-Episode Return vs Iterations For MDPs $H = 100$ (*left*) and $H = 500$ (*right*)

.

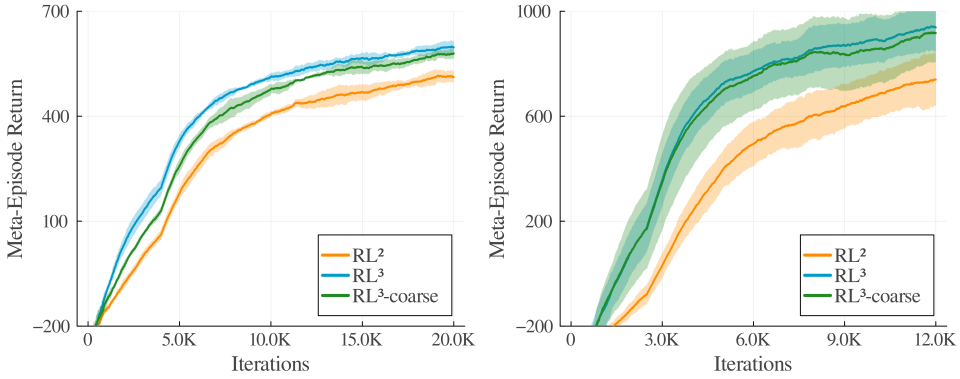

Figure 5: Average Meta-Episode Return vs Iterations For Gridworld 11x11 (*left*) and 13x13 (*right*).

$$Q_1^*(s,a) = R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') \quad (10)$$

$$Q_2^*(s,a) = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_2^*(s',a') \quad (11)$$

$$R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_2^*(s',a') \quad (12)$$

$$R_1(s,a) + \gamma \sum_{s'} T(s,a,s') \text{max}_{a'} Q_1^*(s',a') = R_2(s,a) + \gamma \sum_{s'} T(s,a,s') max_{a'} Q_1^*(s',a') \quad (13)$$

$$R_1(s,a) = R_2(s,a) \quad (14)$$

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?

C.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.

C.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. 6 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-values evolve in a way that leads to faster or more reliable discrimination.

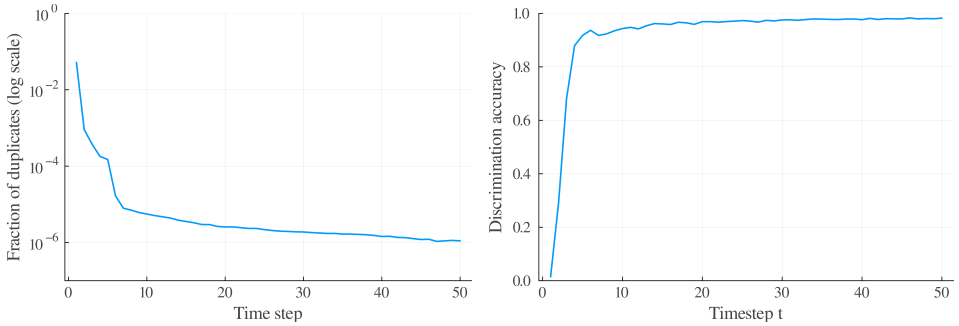

Figure 6: The discriminatory power of Q-values. *Left*: Fraction of δ-duplicates, with $\delta = 0.1$, as a function of time steps in a sample of 5,000 of random MDPs. *Right*: Accuracy of a simple multi-class classifier in predicting task ID given Q-table estimates as function of time step. Both figures are generated using the same policy

D Domain Descriptions

D.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 $\mathcal{N}(0.5, 0.5)$

D.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.

D.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 8 Manhattan distance from the starting tile. The grid also contains slippery wet tiles, fatally dangerous tiles and warning tiles surrounding the latter. Each obstacle spans 3 tiles, in either 3x1 or 1x3 configuration. Wet tiles always occur in adjacent pairs, in either a 2x1 or 1x2 configuration. Warning tiles always occur as a set of 4 tiles non-diagonally surrounding a danger tile. Wet tiles always lead to the agent slipping to one of the directions orthogonal to the intended direction of movement. Entering wet tiles yields an immediate reward of -2. Entering warning tiles causes -10 reward. Entering danger tiles ends the episode and leads to a reward of -100. Normal tiles yield a reward of -1 to incentivize 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. A set of 4 short videos of the Gridworld environment, showing both RL^2 and RL^3 agents solving the same set of instances, can be found at `https://youtu.be/sTQ9vFC53-k`.

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`

- `min_goal_manhat = 8`

The parameters for the OOD variations are largely the same. Below we note only the difference.

DETERMINISTIC: Same as canonical except slip probability on non-wet tiles is 0.

DENSE: `obstacle_set_len=4`

WATERY: `num_water_sets=8`

DANGEROUS: `num_dangers=4`

CORNER: `min_goal_manhat=12`

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.