Learning to Prioritize Planning Updates in Model-based Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Prioritizing the states and actions from which policy improvement is performed can improve the sample efficiency of model-based reinforcement learning systems. Although much is already known about prioritizing planning updates, more needs to be understood to operationalize these ideas in complex settings that involve non-stationary and stochastic transition dynamics, large numbers of states, and scalable function approximation architectures. Our paper presents an online meta-learning algorithm to address these needs. The algorithm finds distributions that encode priority in their probability mass. The paper evaluates the algorithm in a domain with a changing goal and with a fixed, generative transition model. Results show that prioritizing planning updates from samples of the meta-learned distribution significantly improves sample efficiency over fixed baseline distributions. Additionally, they point to a number of interesting opportunities for future research.

1 Introduction

Model-based reinforcement learning (MBRL) provides a computational framework for learning to achieve goals using both direct interactions with an environment, and from virtual interactions with an environment model. Using virtual interactions to improve a policy is a process known as *planning*. In environments where it is easier to learn a transition model than to experience transitions directly, planning can help MBRL systems rapidly learn and adapt.

Previous work shows that when planning under a limited computation budget, prioritizing virtual interactions from states with large Bellman errors can lead to improved sample efficiency (Moore and Atkeson, 1993). This idea is used in several algorithms that apply to environments with deterministic dynamics and a small number of states (Peng and Williams, 1993; Andre et al., 1997; Wingate et al., 2005). However, it remains an open question whether such planning priorities can be learned—particularly in problem settings with stochastic dynamics and where the agent plans with a generative model of its environment.

Our work takes a step in this direction by showing that planning priorities can be efficiently meta learned. We present an algorithm based on the recent Bootstrap Meta Learning (Flennerhag et al., 2022) procedure, for learning distributions of initial planning states. Experiments are performed in a stochastic environment with a periodically changing goal and interactions with a generative model. Results show that planning with a learned prioritization helps to focus policy improvement on states that matter most for planning with a limited computation budget. The key contributions of this work are as follows.

Submitted to the 6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans. Do not distribute.

- An online meta learning algorithm for prioritizing planning updates in MBRL.
- Empirical evidence that in a nonstationary domain, prioritizing updates with our learned distribution of planning states leads to improved sample efficiency.

2 Reinforcement Learning Preliminaries

Our work treats the interaction between an agent and its environment as a Markov Decision Process (Sutton and Barto, 2018), defined with a set of states $\mathcal{S}$ and actions $\mathcal{A}$, a probability distribution over starting states p_1, a conditional probability distribution over the next state and reward $p(S', R|S, A)$, and a discount factor $\gamma \in [0, 1)$. The agents considered here use episodic experiences to learn a policy $\pi \colon \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maximizes the expected sum of discounted rewards observed while following π:

$$q^\pi(s, a) \triangleq \mathbf{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]. \tag{1}$$

Reinforcement learning agents often use the ϵ-greedy policy, which selects uniform random actions with probability ϵ and otherwise selects actions that maximize $q(s, a)$. Our proposed algorithm is designed for general settings where the agent may not have access to the full environment state. In such cases, the agent constructs its own state vector $\mathbf{x}$ and forms an approximate value function $\hat{q}(\mathbf{x}, \mathbf{a}; \boldsymbol{\theta}) \approx q(s, a)$ with a function parameterized by $\boldsymbol{\theta}$. For example, $\hat{q}$ could be a neural network.

The MBRL systems studied here can be categorized under the class of Dyna algorithms (Sutton, 1991). Dyna algorithms interleave policy improvements from both direct and virtual environment interactions. Standard Dyna algorithms learn a value function q and an environment model $m \in \mathcal{P}(\mathcal{S} \times \mathbb{R})$. However, to precisely control for the effects of different planning priorities, this work considers simplified Dyna algorithms that only learn value functions, and only do so through planning. Expanding our results to the full class of Dyna algorithms remains an open challenge for future work.

Given a distribution over possible initial states $d \in \mathcal{P}(\mathcal{S})$, the agent performs a planning update by first forming a transition sample $(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}, \tilde{\mathbf{x}}', \tilde{r})$ from $\tilde{\mathbf{x}} \sim d(\cdot)$, $\tilde{\mathbf{a}} \sim \pi(\tilde{\mathbf{x}})$, and $\tilde{\mathbf{x}}', \tilde{r} \sim m(\tilde{\mathbf{x}}, \tilde{\mathbf{a}})$, then the agent uses a learning rule such as q-learning (Watkins and Dayan, 1992) to update its value parameters $\boldsymbol{\theta}$. Notice that d determines the initial state from which planning occurs and, therefore, its probability mass implicitly encodes the priority of planning updates. In what follows we propose a meta learning algorithm for learning a distribution d whose probabilities are parameterized by $\boldsymbol{\eta}$.

3 Meta Learning a Priority over Initial Planning States

This work is guided by two essential questions. What is a good prioritization of states for planning? And, can such planning priorities be learned? As noted earlier, prioritized sweeping (Moore and Atkeson, 1993) provides a suggestive example of a useful prioritization. Prioritized sweeping uses Bellman error as a heuristic which determines planning priority and improves sample efficiency. Our approach attempts to answer both questions simultaneously by leveraging recent developments in meta-learning. We propose an algorithm which prioritizes states according to their contribution to reducing parameter error. Our algorithm, *Adaptive Prioritization* (AP), meta-learns the distribution d online using the reduction in parameter error to guide the distribution's shape. The procedure is described in full in the Appendix, including pseudocode (see Algorithm 1).

Intuitively, our meta loss is designed to produce a distribution which prioritizes states leading to the largest reductions in parameter error. This can be achieved by minimizing the squared Euclidean difference between some target parameters $\boldsymbol{\theta}^*$ and the expected value parameters after performing a planning update starting from $\boldsymbol{\theta}$, using step size α and learning update $\Delta(\tilde{\mathbf{x}}, \boldsymbol{\theta})$, $\bar{\boldsymbol{\theta}}(\boldsymbol{\eta}) \triangleq \boldsymbol{\theta} + \alpha \sum_{\tilde{\mathbf{x}}, \tilde{\mathbf{a}}} \pi(\tilde{\mathbf{a}}|\tilde{\mathbf{x}}) d(\tilde{\mathbf{x}}; \boldsymbol{\eta}) \Delta(\tilde{\mathbf{x}}, \boldsymbol{\theta})$. Here, our learning update is a semi-gradient q-learning step. However, it could also apply to other gradient-based learning rules. To compute the target parameters $\boldsymbol{\theta}^*$, we use the Bootstrapped Meta Learning procedure (Flennerhag et al., 2022). This procedure uses the meta-learner's own output to approximate $\boldsymbol{\theta}^* \approx \hat{\boldsymbol{\theta}}$. In the MBRL setting, the bootstrapped targets are produced by performing l updates of the form $\hat{\boldsymbol{\theta}} \triangleq \hat{\boldsymbol{\theta}} + \alpha \Delta(\tilde{\mathbf{x}}, \hat{\boldsymbol{\theta}})$ where the first such update is performed on top of $\bar{\theta}$. A final update is applied using the direct interaction from

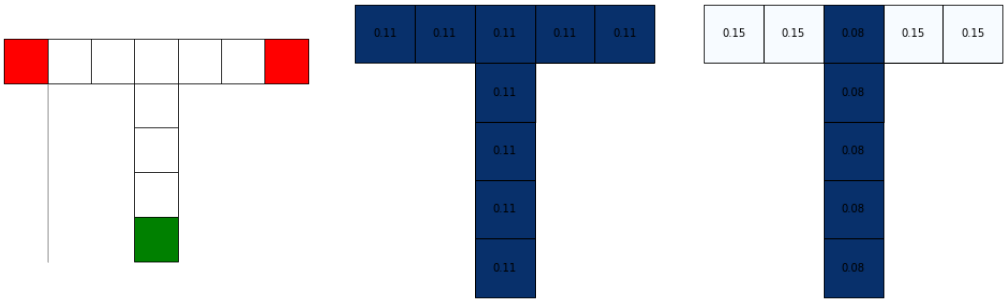


Figure 1: (a) The T-Maze environment. The green state indicates the agent's starting state while the red states indicate terminal states. (b) The uniform prioritized distribution. (c) The domain-specific prioritized distribution. Terminal states are not pictured as no probability is assigned to these states. Lighter colors indicate greater probability mass while text indicates the probability of sampling the corresponding state.

the environment $(\mathbf{x}, \mathbf{a}, \mathbf{x}', r)$, such that $\hat{\boldsymbol{\theta}} \triangleq \hat{\boldsymbol{\theta}} + \alpha\Delta(\mathbf{x}, \hat{\boldsymbol{\theta}})$. Here we suppress the dependence on $\boldsymbol{\eta}$ because the meta loss applies a stop-gradient to this term:

$$\mathcal{L}(\boldsymbol{\eta}) \triangleq ||[\![\hat{\boldsymbol{\theta}}]\!] - \bar{\boldsymbol{\theta}}(\boldsymbol{\eta})||_2^2. \tag{2}$$

Our algorithm minimizes meta loss (2) with online samples of the direct environment interactions using the Adam optimizer (Kingma and Ba, 2014), which enables the priority distribution d to track to a changing environment. Intuitively, a distribution d would be preferred by the meta-loss if, under the same number of planning steps, it brought the parameters closer to optimality. The learned priority distributions will always place mass on states with non-zero parameter error. This suggests that under some mild realizability assumptions, convergence to an optimal value function is possible (Li and Littman, 2008).

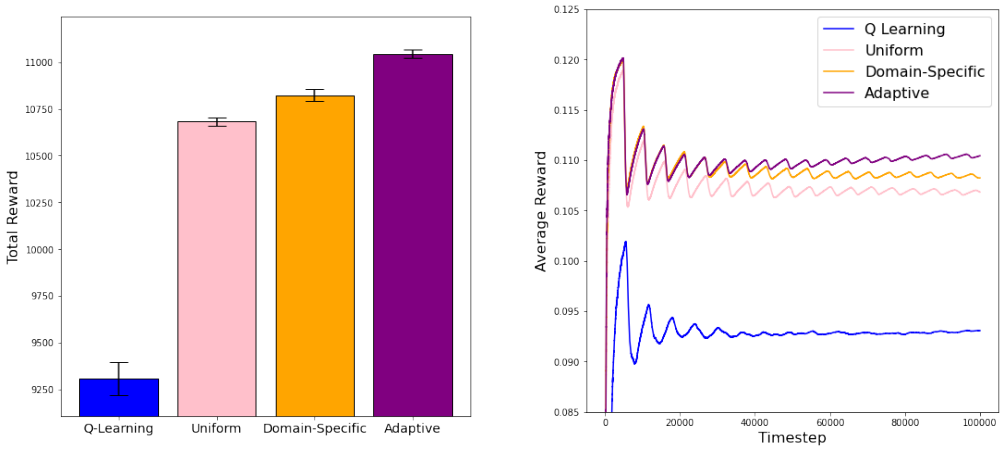
4 Empirical Results

We evaluate the Adaptive Prioritization algorithm in the TMaze; an episodic grid-world environment pictured in Figure 1a. The TMaze is a non-stationary domain in which algorithms capable of adapting to a changing reward structure stand to perform well. Our experiments demonstrate that in this domain, a learned prioritization scheme can outperform fixed distributions, even those which are hand-tailored to the domain to help the agent learn efficiently.

In the TMaze, an agent begins at a starting state and must navigate a vertical hallway, then turn left or right at a junction. Reaching a state at either the left or right of the horizontal hallway results in the termination of an episode. One of the terminal states emits a reward of $+1$ while the other emits 0. Every 600 episodes the rewards are swapped between terminal states. From the agent's perspective, the TMaze is thus non-Markov and non-stationary. At any timestep a random transition may occur with probability ϵ_{env}. A key element of the TMaze is that under the optimal policy only the values of certain states change. The values of states along the vertical hallway do not change when the reward is swapped, while the values of states in the horizontal hallway do change.

Our experiments study agents learning to act in the TMaze environment. All agents behave under an ϵ-greedy policy while attempting to learn optimal q-values. In all our experiments, we use a tabular representation of states and actions while agents learn values which are a linear function of states and actions.

We consider several non-adaptive baseline learning systems. The first baseline is intended to establish the supremacy of model-based algorithms in our domain; it is a Q-Learning agent, which learns without a model, directly from interaction with the environment. We then consider two MBRL agents with fixed priority distributions: Uniform, and Domain-Specific. Uniform places uniform probability over all non-terminal states. The domain-specific distribution is based on the observation that the



(a) Total Reward

(b) Average Reward

Figure 2: (a) Total reward accumulated over 100,000 timesteps by each algorithm considered. Bars represent averages over 30 random seeds, while error bars indicate 95% confidence intervals. (b) Average reward per timestep over the course of training. Each curve represents the average over 30 random seeds. In both figures ϵ_{env} was set to 0.05 and agents performed 3 steps of planning for every environment interaction.

value of states in the vertical portion of the TMaze will not change when the reward switches between terminal states. Thus, the agent has less need to plan from these states. Depictions of the uniform and domain-specific distributions are shown in Figures 1b and 1c.

Figure 2a shows the total rewards accumulated during training for each agent, while Figure 2b shows the average reward per timestep over the course of the experiment. All agents experienced 100,000 timesteps of interaction with the environment. We observe that the adaptive agent achieves the greatest total reward in this setting by a statistically significant margin. Further, we observed that the adaptive agent's average reward increases over the course of training. While that of uniform and domain-specific plateau. Even though the adaptive and baseline algorithms use the same amount of planning, and the same amount of interaction with the environment, the search control distribution learned by the adaptive algorithm allows planning steps to much more effectively update the agents towards a useful value function for control.

5 Conclusion

In this work we introduced the Adaptive Prioritization algorithm and demonstrated the potential meta-learning offers to adaptively adjust the planning procedure. Ultimately, we found that the Adaptive Prioritization algorithm's ability to shape its planning distribution continually allowed this algorithm to achieve superior performance in the non-stationary, episodic TMaze task. We believe this result signals a fruitful use for meta-learning techniques to improve planning algorithms beyond heuristic based methods to initialize planning. In future work, we intend to expand our experiments to settings where a forward transition model is learned simultaneously with the prioritization distribution, and use more powerful function approximators to learn useful distributions in even more complex environments. Eventually, we hope to incorporate adaptation through meta-learning in other aspects of planning algorithms.

References

Andre, D., Friedman, N., and Parr, R. (1997). Generalized prioritized sweeping. *Advances in neural information processing systems*, 10.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., and Singh, S. (2022). Bootstrapped meta-learning. In *International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, L. and Littman, M. (2008). Prioritized sweeping converges to the optimal value function. Technical report, Rutgers University.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive behavior*, 1(4):437–454.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

Wingate, D., Seppi, K. D., and Mahadevan, S. (2005). Prioritization methods for accelerating mdp solvers. *Journal of Machine Learning Research*, 6(5).

A Appendix

A.1 Adaptive Planning Initialization Pseudocode

We provide a detailed description of the Adaptive Prioritization algorithm during each step of interaction with the environment. Given the most recent observation $\mathbf{x}$, AP selects an action $\mathbf{a}$ according to an ϵ-greedy policy derived from its current action-value function. The agent takes the selected action in the environment and receives a reward r and observation $\mathbf{x}'$. Following this, the remainder of the algorithm can be broken down into three steps. First, the agent performs k planning updates using the distribution d. During each update, the agent samples a state using d and selects an action according to its current ϵ-greedy policy. Given the sampled state and selected action, the agent receives a next state and reward from an oracle model of the environment. This produces a tuple of virtual experience $(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}, \tilde{\mathbf{x}}', \tilde{r})$. For each such tuple the agent updates $\boldsymbol{\theta}$ by taking a semi-gradient q-learning step.

After planning, the agent's updated parameters $\boldsymbol{\theta}$ are used to generate the parameter vector $\bar{\boldsymbol{\theta}}$ according to the ExpectedUpdate subroutine (Algorithm 3). ExpectedUpdate calcuates the change in parameter values which would occur were a planning update to occur from **each** possible state-action pair. Each change in parameter values is weighted by the probability of the state being sampled from d and the action being selected by the current ϵ-greedy policy. The result is a weighted average update. This update is applied to $\boldsymbol{\theta}$ to yield $\bar{\boldsymbol{\theta}}$. This expected update results in each component of $\boldsymbol{\eta}$ having a non-zero gradient when differentiating the meta-objective.

Next, the BootstrapTarget subroutine (Algorithm 2) receives $\bar{\boldsymbol{\theta}}$ as input and produces a set of target parameters $\hat{\boldsymbol{\theta}}$. Ideally, the meta-loss would compute the difference between $\bar{\boldsymbol{\theta}}$ and the optimal parameters $\boldsymbol{\theta}^*$. Of course, optimal parameters are not available to our agent. Instead, we use the idea of bootstrapping (Flennerhag et al., 2022). BootstrapTarget performs ℓ additional planning steps on top of $\bar{\boldsymbol{\theta}}$ by sampling starting states from d. Finally, an update is made using the tuple veridical experience $(\mathbf{x}, \mathbf{a}, \mathbf{x}', r)$ collected through direct interaction with the environment. The resulting parameter vector $\hat{\boldsymbol{\theta}}$ approximates $\boldsymbol{\theta}^*$. Adaptive Prioritization optimizes to reduce the parameter difference between $\hat{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{\theta}}$. A stop gradient prevents any adjustment to $\boldsymbol{\eta}$ through the target parameters. Only $\bar{\boldsymbol{\theta}}$ is differentiated through, and so the distribution is shifted in favour of states which have the greatest effect on the parameter difference during the ExpectedUpdate step.

Algorithm 1 Adaptive Planning Initialization

Receive $\mathbf{x}_1$ from environment.
for $t = 1, 2, 3, \cdots$ **do**
  Take ϵ-greedy $\mathbf{a}_t$ from $\mathbf{x}_t$ and receive $\mathbf{x}_{t+1}$ and r_{t+1}.
  # Perform planning update.
  for $1, \cdots, k$ **do**
    Take ϵ-greedy $\tilde{\mathbf{a}}$ from $\tilde{\mathbf{x}} \sim d(\boldsymbol{\eta})$.
    $\tilde{\mathbf{x}}', \tilde{r} \sim m(\tilde{\mathbf{x}}, \tilde{\mathbf{a}})$.
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[\tilde{r} + \gamma \max_{\tilde{\mathbf{a}}'} \hat{q}(\tilde{\mathbf{x}}', \tilde{\mathbf{a}}'; \boldsymbol{\theta}) - \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})$
  # Get expected and target parameters, then compute meta-loss.
  $\bar{\boldsymbol{\theta}} \leftarrow ExpectedUpdate(\boldsymbol{\theta}, d, m)$
  $\hat{\boldsymbol{\theta}} \leftarrow BootstrapTarget(\bar{\boldsymbol{\theta}}, \boldsymbol{\eta}, \mathbf{x}, \mathbf{a}, r, \mathbf{x}')$
  $\mathcal{L}(\boldsymbol{\eta}) \leftarrow ||[[\hat{\boldsymbol{\theta}}]] - \bar{\boldsymbol{\theta}}(\boldsymbol{\eta})||_2^2$
  $\boldsymbol{\eta} \leftarrow$Adam$(\boldsymbol{\eta}, \mathcal{L})$

Algorithm 2 BootstrapTarget

input: $\boldsymbol{\theta}$, $\boldsymbol{\eta}$, $(\mathbf{x}, \mathbf{a}, r, \mathbf{x}')$
for $1, \cdots, \ell$ **do**
  $\tilde{\mathbf{x}}, \tilde{\mathbf{a}} \sim d(\boldsymbol{\eta})$.
  $\tilde{\mathbf{x}}', \tilde{r} \sim m(\tilde{\mathbf{x}}, \tilde{\mathbf{a}})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[\tilde{r} + \gamma \max_{\mathbf{a}' \in \mathcal{A}} \hat{q}(\tilde{\mathbf{x}'}, \mathbf{a}'; \boldsymbol{\theta}) - \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})$
$\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha[r + \gamma \max_{\mathbf{a}' \in \mathcal{A}} \hat{q}(\mathbf{x}', \mathbf{a}'; \boldsymbol{\theta}) - \hat{q}(\mathbf{x}, \mathbf{a}; \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} \hat{q}(\mathbf{x}, \mathbf{a}; \boldsymbol{\theta})$
return: $\hat{\boldsymbol{\theta}}$

Algorithm 3 ExpectedUpdate

input: $\boldsymbol{\theta}$, $\boldsymbol{\eta}$, m
for $\tilde{\mathbf{x}} \in \mathcal{S}$ **do**
  for $\tilde{\mathbf{a}} \in \mathcal{A}$ **do**
    $\tilde{\mathbf{x}}', \tilde{r} \sim m(\tilde{\mathbf{x}}, \tilde{\mathbf{a}})$.
    $\Delta_{\tilde{\mathbf{x}}, \tilde{\mathbf{a}}} \leftarrow [\tilde{r} + \gamma \max_{\tilde{\mathbf{a}}' \in \mathcal{A}} \hat{q}(\tilde{\mathbf{x}'}, \tilde{\mathbf{a}}'; \boldsymbol{\theta}) - \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} \hat{q}(\tilde{\mathbf{x}}, \tilde{\mathbf{a}}; \boldsymbol{\theta})$
$\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \sum_{\tilde{\mathbf{x}} \in \mathcal{S}, \tilde{\mathbf{a}} \in \mathcal{A}} \pi(\tilde{\mathbf{a}}|\tilde{\mathbf{x}}) d(\mathbf{x}; \boldsymbol{\eta}) \Delta_{\tilde{\mathbf{x}}, \tilde{\mathbf{a}}}$
return: $\bar{\boldsymbol{\theta}}$

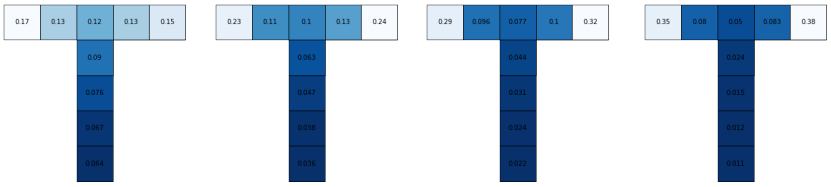
A.2 Hyperparameter Selection

Hyperparameter	Values
Step Size	1e-3, 1e-3, 1e-1, 1e0
Meta-Step Size	5e-4, 1e-3, 5e-3, 1e-3, 5e-2, 1e-1
ϵ_{policy}	0.1
Bootstrap Target Samples	10

Table 1: Hyperparameters and values considered during grid search. Note that Meta-Step Size and Boot Target Samples are only used by the Adaptive Search Control Algorithm

To select hyperparameters, we perform a grid search over all possible hyperparameter configurations from Table 1. Each configuration is run with 30 random seeds during the selection process. We then run the best hyperparameter configurations for an additional 30 seeds and report these as the final results.

A.3 Learned Distributions



(a) 25% of Training (b) 50% of Training (c) 75% of Training (d) 100% of Training

Figure 3: Plots of the distribution learned by API at different points during training. It is clear that probability is concentrated in states adjacent to the terminal states. These states have the greatest change in value when the reward regime switches.

Figure 3 shows snapshots of the probability distribution learned by the API algorithm at various points throughout training. API begins with a uniform distribution and quickly learns to concentrate probability in the states nearest to the terminal states. As well, it clearly learns the intuition underlying the domain-specific distribution described earlier. Very little probability mass is placed in the vertical hallways, where the value of states does not change when the reward regimes changes. API is clearly learning that samples from these states is relatively less important than sampling from states whose value function does change.

A.4 Results for Additional Settings of the Environment

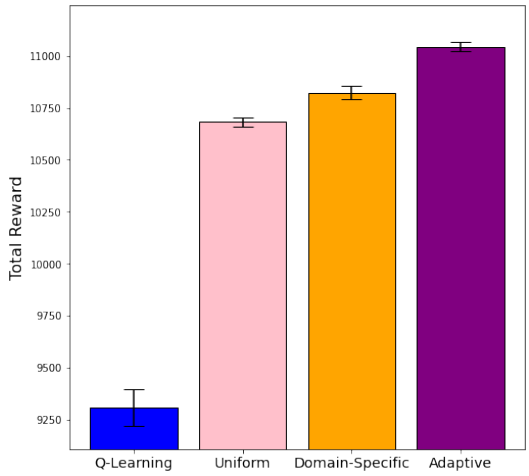
Figure 4 shows additional TMaze results for all combinations of ϵ_{env} . The results demonstrate that in all settings, the adaptive algorithm matches the performance of the hand-constructed biased distribution while outperforming biased in several settings. Table TODO INCLUDE REF also reports the results of the statistical significance test in each ϵ_{env} and planning steps setting.

A.5 Statistical Significance Tests

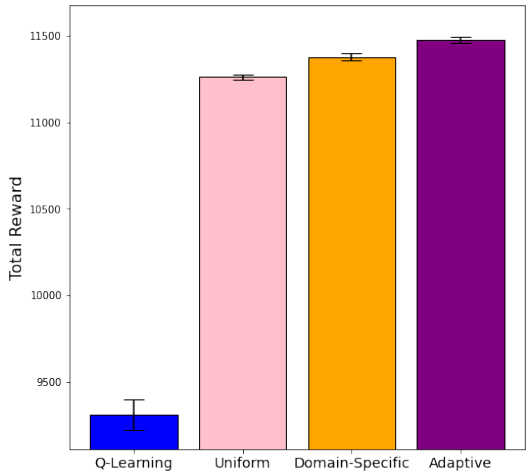
As shown in Figure 2, our metric for evaluating each algorithm was the total reward accumulated over the course of 100,000 interactions with the TMaze environment. To determine whether statistically significant differences exists between the mean results of each algorithm we performed Tukey's method. We report the results in the table below. The p-values indicate that in all settings of ϵ_{env} and for both 3 and 5 steps of planning, API at least matches the performance of the Domain-Specific distribution. In several settings, API exceed the performance of Domain-Specific.

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-1375.467	0.000
Q-Learning v. Domain-Specific	-1515.500	0.000
Q-Learning v. API	-1736.133	0.000
Uniform v. Q-Learning	1375.467	0.000
Uniform v. Domain-Specific	-140.033	0.000
Uniform v. API	-360.667	0.000
Domain-Specific v. Q-Learning	1515.500	0.000
Domain-Specific v. Uniform	140.033	0.000
Domain-Specific v. API	-220.633	0.000
API v. Q-Learning	1736.133	0.000
API v. Uniform	360.667	0.000
API v. Domain-Specific	220.633	0.000

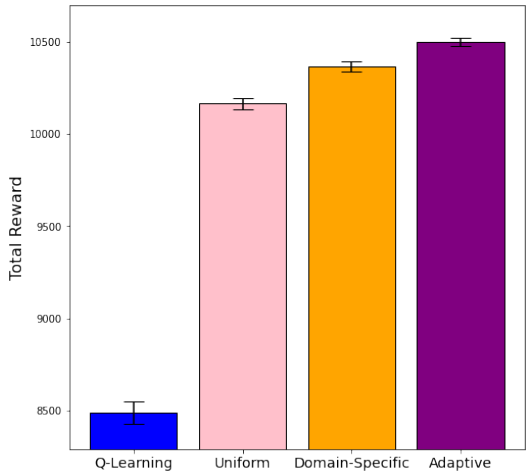
Table 2: Statistical significance test results for ϵ_{env} of 0.05 and 3 planning steps



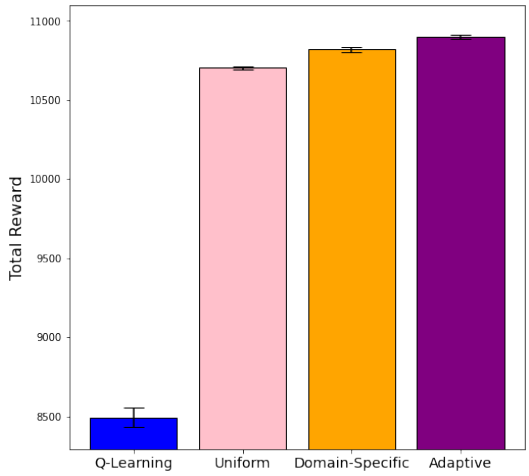
(a) ϵ_{env}: 0.05, planning steps: 3



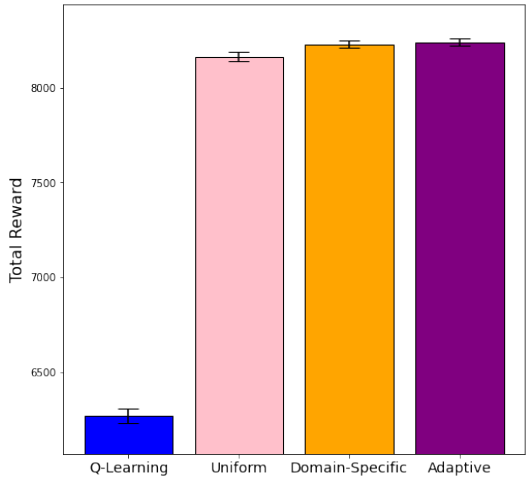
(b) ϵ_{env}: 0.05, planning steps: 5



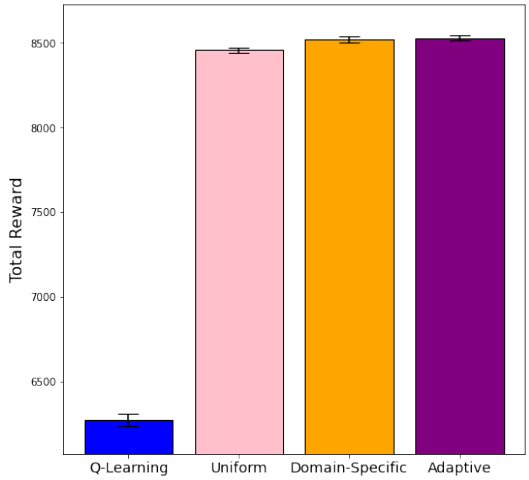
(c) ϵ_{env}: 0.1, planning steps: 3



(d) ϵ_{env}: 0.1, planning steps: 5



(e) ϵ_{env}: 0.3, planning steps: 3



(f) ϵ_{env}: 0.05, planning steps: 5

Figure 4: Total reward results for all value of ϵ_{env} and all numbers of planning steps considered.

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-1955.000	0.000
Q-Learning v. Domain-Specific	-2070.700	0.000
Q-Learning v. API	-2170.033	0.000
Uniform v. Q-Learning	1955.000	0.000
Uniform v. Domain-Specific	-115.700	0.002
Uniform v. API	-215.033	0.000
Domain-Specific v. Q-Learning	2070.700	0.000
Domain-Specific v. Uniform	115.700	0.002
Domain-Specific v. API	-99.333	0.012
API v. Q-Learning	2170.033	0.000
API v. Uniform	215.033	0.000
API v. Domain-Specific	99.333	0.012

Table 3: Statistical significance test results for ϵ_{env} of 0.05 and 5 planning steps

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-1671.433	0.000
Q-Learning v. Domain-Specific	-1872.967	0.000
Q-Learning v. API	-2004.933	0.000
Uniform v. Q-Learning	1671.433	0.000
Uniform v. Domain-Specific	-201.533	0.000
Uniform v. API	-333.500	0.000
Domain-Specific v. Q-Learning	1872.967	0.000
Domain-Specific v. Uniform	201.533	0.000
Domain-Specific v. API	-131.967	0.000
API v. Q-Learning	2004.933	0.000
API v. Uniform	333.500	0.000
API v. Domain-Specific	131.967	0.000

Table 4: Statistical significance test results for ϵ_{env} of 0.1 and 3 planning steps

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-2209.900	0.000
Q-Learning v. Domain-Specific	-2325.833	0.000
Q-Learning v. API	-2406.900	0.000
Uniform v. Q-Learning	2209.900	0.000
Uniform v. Domain-Specific	-115.933	0.000
Uniform v. API	-197.000	0.000
Domain-Specific v. Q-Learning	2325.833	0.000
Domain-Specific v. Uniform	115.933	0.000
Domain-Specific v. API	-81.067	0.003
API v. Q-Learning	2406.900	0.000
API v. Uniform	197.000	0.000
API v. Domain-Specific	81.067	0.003

Table 5: Statistical significance test results for ϵ_{env} of 0.1 and 5 planning steps

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-1892.933	0.000
Q-Learning v. Domain-Specific	-1957.300	0.000
Q-Learning v. API	-1968.467	0.000
Uniform v. Q-Learning	1892.933	0.000
Uniform v. Domain-Specific	-64.367	0.002
Uniform v. API	-75.533	0.000
Domain-Specific v. Q-Learning	1957.300	0.000
Domain-Specific v. Uniform	64.367	0.002
Domain-Specific v. API	-11.167	0.915
API v. Q-Learning	1968.467	0.000
API v. Uniform	75.533	0.000
API v. Domain-Specific	11.167	0.915

Table 6: Statistical significance test results for ϵ_{env} of 0.3 and 3 planning steps

Algorithms Compared	Difference in Mean Total Reward	P-Value
Q-Learning v. Uniform	-2186.833	0.000
Q-Learning v. Domain-Specific	-2249.933	0.000
Q-Learning v. API	-2259.333	0.000
Uniform v. Q-Learning	2186.833	0.000
Uniform v. Domain-Specific	-63.100	0.001
Uniform v. API	-72.500	0.000
Domain-Specific v. Q-Learning	2249.933	0.000
Domain-Specific v. Uniform	63.100	0.001
Domain-Specific v. API	-9.400	0.934
API v. Q-Learning	2259.333	0.000
API v. Uniform	72.500	0.000
API v. Domain-Specific	9.400	0.934

Table 7: Statistical significance test results for ϵ_{env} of 0.3 and 5 planning steps