Adaptive Search Control through Meta-Gradient Reinforcement Learning

by

Bradley Burega

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In model-based reinforcement learning, an agent can improve its policy by planning: learning from experience generated by a model. Search control is the problem of determining which starting state should be used to generate this experience. Given a limited planning budget, an agent should be efficient in selecting experience, as some states may provide much greater value to learning than others. Particularly in complicated environments with large state spaces, an agent could easily waste time planning in states which have little effect on policy improvement. Thus, search control should carefully select starting states to maximize planning efficiency. In this work, we study effective search control and examine how search control can be designed when an agent has access to either a perfect or imperfect model. We show that in a non-stationary environment, search control can be more effective by focusing updates on states with high value error. While using an imperfect model, it can be effective to avoid learning from states where the model produces erroneous reward predictions. However, in most cases it is not feasible to hand-design a search control strategy for a novel environment. Towards this end, we introduce a novel algorithm which uses meta-learning to adjust an agent's search control strategy while it learns a policy. We empirically evaluate the ability of this algorithm to improve the efficiency of planning in a stochastic and non-stationary environment. The results demonstrate that this algorithm outperforms several fixed search control approaches and learns behaviours similar to baselines hand-designed to perform well in our test environment.

Preface

Portions of this work appeared as *Learning to Prioritize Planning Updates in Model-based Reinforcement Learning* at the Workshop on Meta-Learning located at the Thirty-sixth Conference on Neural Information Processing Systems.

Change is the only constant.

– Heraclitus (attributed)

Acknowledgements

First, I would like to thank my supervisors Michael Bowling and John Martin. They have been essential in guiding me towards the completion of this thesis. Michael's incredible expertise has been crucial throughout my work, whether at the low-level of debugging experimental code or at the high-level of designing experiments and algorithms. John has provided a wealth of mentorship on how to become an effective researcher; I have learned much both from discussions with him but also from the opportunity to observe his approach to research.

I am thankful to Zaheen Ahmad for his mentorship and friendship over the course of my studies. Over so many discussions with Zaheen I have learned volumes about navigating the challenges of research and graduate school, and I have greatly appreciated the encouragement he has provided along the way.

I would also like to thank the friends and colleagues who have provided many insightful discussions and added so much levity to my experience at the University of Alberta. In particular, Alex Ayoub, Claire Wilson, David Tao, Garrett Engert, Liam Peet-Pare, and Rohan Nuttall have been invaluable friends during these past few years. There are many others at AMII who have made this experience special, and I greatly appreciate the community of talented people who make it such a strong institution.

Finally, I would like to thank my parents Joyce Holoboff and Rick Burega for their love and support throughout this journey and many others. I would not have been able to accomplish this without the encouragement and care they have provided me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a computational means of learning through interaction with an environment. Given the current state of the environment, an RL agent takes an action and, in return, receives the updated environment state and a scalar value called reward. By seeking to maximize future reward, RL agents are capable of learning complex behaviours needed to solve challenging tasks. RL agents have learned to play chess, Go, and shogi at superhuman levels (Schrittwieser et al., 2020), to control plasma within nuclear fusion reactors (Degrave et al., 2022), and to learn highly-performant sorting algorithms from scratch (Mankowitz et al., 2023).

However, RL agents are often very data hungry, requiring massive amounts of interaction with an environment to learn fruitful decision making. For example, training the famous AlphaZero agent (Schrittwieser et al., 2020) required 44 million games of chess, 24 million games of shogi, and 21 million games of Go. Generally, such vast amounts of interaction may not be available to the agent. Furthermore, the environment could drift over time, making any accumulated experience obsolete when the environment enters a new regime.

An important goal of RL research is understanding, characterizing, and reducing the amount of data needed for effective learning. To this end, we can search for algorithms that improve sample efficiency. Sample efficiency is the amount of environment interaction required for an agent to reach a given level of performance. For instance, an agent which learns to play chess at a given skill level with one million games of experience is more sample efficient than an

agent which requires ten million games. Greater sample efficiency reduces the need for huge amounts of data and broadens the scope of problems to which we can apply RL.

One means of increasing sample efficiency is through model-based RL (MBRL), where agents have access to a model of the environment. By interacting with this model, in addition to interacting with the environment, an agent gains an additional stream of experience from which to learn. This is known as *planning*. MBRL has resulted in agents which are capable of learning to solve complex tasks with a minimal amount of real world interaction, in some cases only less than 100 episodes of experience are needed to learn an effective model (Deisenroth & Rasmussen, 2011). Sufficiently-useful models have even produced agents which do not need to learn directly from environment experience at all (Hafner et al., 2020; Hafner et al., 2021; Hafner et al., 2023).

While planning in any capacity can often improve sample efficiency, careful consideration of how to best use model experience can result in further gains. Consider Prioritized Sweeping (PS), this algorithm selects states for planning according the TD-error of the ensuing update (Moore & Atkeson, 1993). PS achieves much greater sample efficiency compared to ordering planning updates randomly. Generally speaking, the problem of selecting the starting state from which planning occurs is called *search control*. Intelligently selecting starting states for planning can result in an agent learning more from each planning update, potentially increasing *planning efficiency*. We use planning efficiency analogously to sample efficiency; an agent which learns more from its model with less planning is more efficient than an agent which requires more model experience.

Improving an agent's use of model experience can help to reduce the overall amount of computation needed for an agent to learn how to accomplish a task. In some cases, a model's imperfections may even cause planning to be harmful to learning. Efficient planning would thus require carefully avoiding incorrect model experience. Further, while there is no guarantee that planning efficiency improvements translate directly into sample efficiency improvements, it's plau-

sible that planning efficiency can aid in the goal of increased sample efficiency. Agents which have perfect models provide an illustrative example. In these cases, model experience is equivalent to real experience from the environment, learning more quickly from a model would thus result in faster learning in the real environment. We believe that the goal of increasing planning efficiency provides compelling benefits to model-based agents. The focus of this work is on studying search control towards this end.

Beyond Prioritized Sweeping, additional prior work has attempted to address the search control problem. Andre et al. (1997) introduced Generalized Prioritized Sweeping to extend PS to large state-spaces which necessitate function approximation. Recently, more advanced search control methods have been developed. Pan et al. (2019) use the trajectory of states generated by a hill climbing algorithm on the agent's current value function estimate to create a distribution over states for search control. In further work, Pan et al. (2020) estimate which regions of the value function are most difficult to learn, focusing search control on these regions. While these recent works have brought novel solution methods to the search control problem, we believe search control remains an understudied area of research and will benefit from the incorporation of recent advances in machine learning techniques.

In this thesis, we leverage recent work in meta-learning (Flennerhag et al., 2022) to take some preliminary steps towards the goal of algorithms supplied with customized and adaptive search control. Such adaptive search control could adjust planning priorities in shifting environments, imperative for domains where stationarity cannot be guaranteed. Moreoever, complex environments are not amenable to hand-designed search control distributions, and heuristics may fail for reasons which are difficult to pin down. Learned search control distributions could automatically adjust themselves to these complex environments and discover planning priorities which may not be obvious to practitioners.

This thesis works towards the ideals of an adaptive search control distribution described above. The contributions of this work are as follows:

1. Principles for effective search control and motivating experiments to support them.

2. An algorithm which *learns to perform search control* along with empirical support of its effectiveness, showing it to be capable of following our established principles when appropriate.

Our investigations into search control are conducted in a gridworld domain and compare the performance of agents given different environment models. The gridworld domain allows for clear analysis of an agent's behaviour while also providing the stochasticity and non-stationarity expected of larger, more complex domains. We examine agents under perfect, imperfect, and learned models to evaluate that the proposed algorithm, Meta-Gradient Search Control, is capable of learning useful distributions under settings where the model's dynamics may not match that of the environment and may change during the course of policy improvement.

This thesis is organized into six chapters. Chapter 2 discusses relevant background material and related work. Chapter 3 introduces the two principles of search control: focus on high value error states and avoid incorrect states. Chapter 4 introduces the Meta-Gradient Search Control algorithm. Chapter 5 presents experimental results comparing the performance of Meta-Gradient Search Control against baseline algorithms in the TMaze environment. Results are presented for perfect, imperfect, and learned models. Finally, Chapter 6 provides a summary of the thesis and directions for future work.

Chapter 2

Background Material

2.1 Reinforcement Learning

Reinforcement learning is an algorithmic framework by which computational agents can learn to solve sequential decision making problems. Through a trial-and-error process, RL agents learn how to make decisions that result in the greatest long-term payoff. An RL agent interacts with an environment by taking actions, and receiving a numerical signal known as reward. The goal of an RL agent is to maximize the cumulative reward it receives while interacting with the environment. By learning which actions to take to receive maximum cumulative reward, an agent can learn to effectively make decisions and ultimately accomplish complex goals.

Formally, the interaction between agent and environment is modelled as a Markov Decision Process (MDP). This work follows the conventions of Sutton and Barto (2018) where an MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$. $\mathcal{S}$ is the set of all states, $\mathcal{A}$ is the set of all actions, and $\mathcal{R} \subseteq \mathbb{R}$ is the set of all rewards. Throughout this work uppercase letters denote random variables while lowercase letters denote specific outcomes. During each interaction $t \in \mathbb{N}$, the agent is in a state $S_t \in \mathcal{S}$, takes an action $A_t \in \mathcal{A}$, receives a reward $R_{t+1} \in \mathcal{R}$ and transitions to a new state $S_{t+1} \in \mathcal{S}$. A discount factor of $\gamma \in [0, 1]$ is applied to rewards received in future timesteps. We use Δ to denote a probability distribution over a set. We then define $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$. This is known as the dynamics function and represents the probability of transitioning from one state to another after taking an action and receiving a reward:

$p(s', r|s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$. An MDP is *Markov* because the next state and reward depend only on the current state and action, that is, the formulation of p means that no additional information from the past influences the future dynamics of the environment.

2.2 Policies and Value Functions

To interact with an environment defined as an MDP, an agent needs a way to choose an action to take at each timestep. An RL agent typically selects actions with a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. The agent's policy is a probability distribution which determines the likelihood of taking an action a in state s.

RL agents also need a means of evaluating states and actions to determine whether they are useful in achieving a goal. Such evaluation is accomplished using a value function. More formally, a value function is a mapping from a state or state-action pair to a scalar value representing the expected cumulative discounted reward the agent may receive by following π under the environment dynamics $p(s', r|s, a)$. The state value function is defined as

$$v_\pi(s) = \mathbf{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, \pi]. \tag{2.1}$$

Similarly, the state-action value function is defined as

$$q_\pi(s, a) = \mathbf{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, A_t = a, \pi]. \tag{2.2}$$

The optimal state value function is defined as

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.3}$$

for any state $s \in \mathcal{S}$. The optimal state-action value function is

$$q_*(s, a) = \max_\pi q_\pi(s, a). \tag{2.4}$$

The optimal policy π_* can be defined as the policy which maximizes $v_\pi(s)$ for all states $s \in \mathcal{S}$. The existence of at least one optimal policy is guaranteed for any MDP (Sutton & Barto, 2018). We can also define a *greedy* policy as a policy which selects the maximally valued action at each timestep. Multiple

actions may be maximally valued, in such cases an agent can use a tie-breaking mechanism to choose among them. A policy which is greedy with respect to the optimal state-action value function is itself an optimal policy.

2.3 Q-Learning

Q-Learning is a model-free, off-policy algorithm that learns the optimal value function independent of the policy the agent follows (Watkins & Dayan, 1992). As a model-free algorithm, Q-Learning does not use a model of the environment and learns strictly from real experience. As an off-policy method, Q-Learning allows the agent to learn the optimal value function q_* while behaving according to its current policy. Q-Learning uses an agent's estimated state-action value function along with samples from the environment in order to update the agent's state-action values towards optimality. The Q-Learning update rule is defined as

$$q_\pi(S_t, A_t) \leftarrow q_\pi(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') - q_\pi(S_t, A_t)] \tag{2.5}$$

where $\alpha \in [0, 1]$ is the step-size hyperparameter controlling how quickly an update adjusts the value function.

We use the Q-Learning algorithm along with an ϵ-greedy policy. In other words, in a given state, the agent will select a random action with probability ϵ, otherwise, it will select an action which maximizes its current value function $q_\pi(s_t, \cdot)$. The stochasticity of this policy allows the agent to experience different transitions, possibly discovering actions which may be of higher value. Throughout this work, we perform experiments using the Q-Learning algorithm to learn state-action values.

2.4 Model-Based Reinforcement Learning

Model-based reinforcement learning is a family of RL algorithms in which agents are given access to a model of the environment. The agent can learn by interacting with the environment and receiving *real* experience, or, the agent

can use the model to generate *model* experience[1]. By interacting with a model $m : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$, the agent generates transitions of the form $(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')$ (where the notation $\tilde{x}$ is used to indicate objects generated by a model). Note that often $\tilde{s}$ and $\tilde{a}$ are real states and actions from environment interaction which an agent uses to query a model. We write $(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')$ here as a broad definition which encompasses cases where $\tilde{s}$ and $\tilde{a}$ may themselves be generated by a model. The form of the model and the way in which the agent makes use of model experience differs depending on the algorithm in consideration.

We define planning as any use of a model to improve an agent's policy (Sutton & Barto, 2018). Planning in this sense can be broken down into two different means of improving a policy: background planning and decision-time planning.

In decision-time planning, immediately before taking an action, an agent uses its model to search through the space of state-action pairs that follow from its current policy and value function. By observing the effects of different actions in simulation, the agent can form estimates of the value of each action it is considering. With a good model and a sufficient number of samples, the agent can improve its ability to correctly make the decision it is faced with. This is a form of ephemeral policy improvement; improvements to the policy or value function from decision-time planning generally do not persist to the next timestep of interaction with the environment. The well-known AlphaGo family of algorithms is a prominent example of an MRBL agent which uses decision-time planning (Schrittwieser et al., 2020; Silver et al., 2016; Silver et al., 2018; Silver et al., 2017). Agents in this family typically improve their policies by performing a tree search at decision-time, allowing them to achieve greater results than using by using their policies alone to select actions.

In contrast, background planning involves an agent using model experience to update its value function or policy such that these updates persist through time. Generally, this means that after a real update, an agent will perform

[1]We will use the terms real experience and model experience throughout this work to refer respectively to experience from interacting with the real environment and experience from interacting with a model.

one or more updates using model experience. While the estimates formed in decision-time planning are discarded on the next timestep, updates to the agents value function or policy will be carried forward. Dyna algorithms fall under this type of planning (Sutton, 1991). We discuss these algorithms in detail in Section 2.6. A more recent example of background planning is the Dreamer family of algorithms (Hafner et al., 2020; Hafner et al., 2021; Hafner et al., 2023). Dreamer and its descendants learn a parametric model of the world with discretized states. These agents use transitions from this discrete model to learn a policy and value function. However, despite learning purely from model experience, Dreamer and its variants have demonstrated strong performance in complex domains. This work is centered on background planning, and particularly on an important facet of background planning known as search control.

Models vary in the degree to which they match the real environment. In some cases, an agent may be provided with a perfect simulator of the environment. In other cases, the agent may receive or learn a model which imperfectly matches the environment's dynamics, or, a model which is a simplification of a far more complex environment. Such imperfect models introduce challenges which an agent must be robust to. Transitions received from parts of the model which do not match the environment can harm an agent's learning, causing it to learn incorrect values and a poor performing policy. Determining which parts of a model can be trusted and how to best make use of model experience is a challenging problem in MBRL.

Often an agent may not be provided a model a priori, but may be required to learn a model of the environment. Learned models present an additional challenge as an agent must be capable of effectively using a model which changes throughout training, and moreover, must be robust to imperfections in the model due to its learned nature. Determining how a model should be learned and where model updates should be applied is another challenging aspect in the design of MBRL algorithms.

Prior work has investigated how agents can best use model experience. Holland et al. (2019) showed that in some model-based settings, the length of the

trajectories of model experience can have an important effect on the how useful this experience is to an agent. Abbas et al. (2020) used model uncertainty as a mechanism to selectively sample the state-space of model experience, attempting to avoid planning in states in which the model is uncertain and potential harmful .

Models also differ in the information which they provide to an agent. Sample models can be queried by an agent to produce transitions of experience, however these models do not provide direct access to the full distribution of the environment's dynamics. In contrast, distribution models encode the full dynamics of the environment, in other words all the possible transitions which can occur and their associated probabilities. However, in complex and high-dimensional problems, distributions models may be prohibitively difficult to learn. We use sample models in the experiments conducted in this work and in the algorithm which we introduce.

2.5 Search Control

Search control is the selection of which starting state an agent should use to query its model when planning (Sutton & Barto, 2018). Let's perform a thought experiment to demonstrate that planning in different states may be more or less beneficial to the agent's learning process. Consider the game of chess. If one imagines that each possible configuration of the chess board is a state, there is an overwhelming multitude of possible states. Some are clearly less instructive than others. A state where all the white pawns are in one corner of the board while all the black pawns are in the opposite corner is exceedingly unlikely to occur in a real chess game. Using this state as an example during learning likely won't be as beneficial as a state which occurs during the normal course of play. Further, a very simple strategy for search control is to randomly select a state for planning. Among all of the vast number of chess states, very many will be entirely irrelevant to real play. If we designed an agent with this naive search control strategy, it would be incredibly inefficient in terms of planning, spending many cycles of computation on states which, at best,

don't significantly improve the agent's policy.

This intuition emphasizes the importance of search control, and demonstrates that the careful selection of states and actions for planning is likely to affect the efficiency of an agent's learning. We view effective search control as maximizing planning efficiency — achieving the greatest amount of learning given a limited planning budget. Let us formalize this problem as the selection of a state $\tilde{s} \in \mathcal{S}$ which the agent uses to query its model $m(\tilde{s}, \tilde{a})$ to generate a tuple of experience $(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')$. We focus our attention on search control methods which select $\tilde{s}$ by sampling from a distribution $d : \mathcal{S} \to \Delta(\mathcal{S})$. The algorithms presented in this work use the agent's current policy to select an action $\tilde{a}$ given $\tilde{s}$. While intelligently and adaptively selecting actions to maximize planning efficiency is also beneficial to an agent, we leave this as a problem for future work.

Varying approaches have been developed to address the problem of search control. One of the simplest methods is to keep track of previously observed states and sample from among them, either uniformly at random or in proportion to how often each state has been observed (Sutton, 1991).

Another approach, Prioritized sweeping, was initially developed as a method for organizing updates from prior experience, but can also be considered a form of search control when integrated into an MBRL context. We discuss this method in detail in Section 2.7. A more recent approach is introduced in (Pan et al., 2020) which uses the heuristic that regions of the value function which are more difficult to estimate should receive more samples for planning. The method uses the frequency of the Fourier representation of the value function to determine which regions of the function are more complex, and allocates more samples to these regions. The authors show that MBRL algorithms using this heuristic outperform other search control methods in a maze domain.

Prioritized Experience Replay (PER) is another closely related method (Schaul et al., 2016). PER is a method of organizing previous experience to improve sample efficiency. Transitions observed by an agent are stored in a replay buffer and are stochastically sampled in proportion to the TD-error of each transition. Empirical results demonstrate that this method improves sam-

ple efficiency over a Deep Q-Learning agent with a uniform sampling scheme. A drawback of PER is that it is a non-parametric method and cannot generalize priority over similar states. This also means that priorities may become stale as the agent's value function changes.

2.6 Dyna

Dyna is a well-known architecture for MBRL agents (Sutton, 1991). The Dyna framework interleaves three processes: value function updates, planning updates, and model updates. Interleaving these processes allows agents to efficiently make use of experience received from both the real environment and the model. In the most abstract version of Dyna, all three processes can occur simultaneously. Because Dyna's planning updates adjust the agent's value function, Dyna agents can be considered background-planning algorithms. This work considers a simplified Dyna architecture in which a real update is always performed prior to planning updates.

Dyna-Q is a particular instantiation of the Dyna architecture which uses the Q-Learning update rule both in planning updates and in updates from real experience. The basic structure of Dyna-Q is that an agent interacts with the real environment and receives a tuple of experience which it then uses to update its value function. Next, the agent uses this same transition to update its model. Finally, the agent queries its model for experience and performs k planning updates. Pseudocode of Dyna-Q is shown in Algorithm 1.

Algorithm 1 Dyna-Q

1: Receive s from the environment.
2: **for** $t = 1, 2, 3, \cdots$ **do**
3: Take a in s and receive s' and r.
4: # Update value function
5: $q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha[r + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$
6: # Update model
7: $m(s, a) \leftarrow s', r$
8: # Perform k planning updates
9: **for** $1, \cdots, k$ **do**
10: Select $\tilde{s} \sim d$
11: Take action $\tilde{a}$ in $\tilde{s}$
12: Sample $\tilde{r}, \tilde{s}' \sim m(\tilde{s}, \tilde{a})$
13: $q_\pi(\tilde{s}, \tilde{a}) \leftarrow q_\pi(\tilde{s}, \tilde{a}) + \alpha[r + \gamma \max_{\tilde{a}'} q_\pi(\tilde{s}', \tilde{a}') - q_\pi(\tilde{s}, \tilde{a})]$

The algorithm developed in this work will follow a structure similar to the version of Dyna-Q shown above. Note that, line 10 of Algorithm 1 is especially important as this is where Dyna-Q selects states for planning via search control.

2.7 Prioritized Sweeping

A classic method of performing search control is through Prioritized Sweeping (PS) (Moore & Atkeson, 1993). Prioritized sweeping is a means of ordering planning updates according to which updates are the most urgent. Urgency is taken to be the change in a state's value. When a state undergoes a significant change in value, the values of predecessor states are also very likely to have changed. By using change in value to determine which updates are the most urgent, prioritized sweeping is able to focus on areas of the state space which most benefit from planning updates, as opposed to updating states randomly or uniformly.

The algorithm works by maintaining a priority queue of states from which updates should be performed, placing items on the queue according to their priority and pulling off the highest priority element at each planning step. When a state is pulled off the queue, the priorities of the predecessors to that state are computed and each predecessor is added to the queue. Working

backwards from states which undergo large changes in value allows updated information to quickly propagate to preceding states that were also affected by this value change. Empirically, prioritized sweeping often learns much faster than other Dyna methods as its use of planning results in much more efficient updates.

The main drawback to prioritized sweeping is that it does not scale to large state spaces which require function approximation. Prioritized sweeping relies on a tabular representation of the environment and an ability to predict predecessor states. While it is possible for an agent to keep a log of which state preceded another, in very large state spaces there may be many predecessor states which have not been observed by the algorithm. These states would not have a priority assigned to them and would not be updated by prioritized sweeping, meaning that important updates would not be propagated to a significant portion of the state space. For this reason, search control methods which are amenable to function approximation are highly desirable.

2.8 Meta Reinforcement Learning

Meta reinforcement learning is the process of learning to reinforcement learn (Beck et al., 2023). In general, meta RL is concerned with learning aspects of RL algorithms which otherwise would be hand designed. This can extend to learning entire algorithms. The motivation for replacing hand crafted components with learned ones can vary, but a frequent goal of meta RL is to increase the sample efficiency of RL algorithms — ultimately, allowing these algorithms to quickly adapt to new tasks or to be capable of learning a broad range of tasks (Beck et al., 2023).

This work addresses planning efficiency rather than targeting sample efficiency directly. By learning which states are most useful for planning we can improve the effectiveness of planning updates. We believe this focus is aligned with the goals of meta RL stated above. In some cases, increased planning efficiency can result in better sample efficiency. Moreover, better use of model experience is likely beneficial to an agent's ability to adapt to new tasks. A

model-based agent adapting to a new task must also update its model; prudent selection of model experience can allow an agent to take advantage of a shifting model, or to avoid stale transitions which would be damaging to learning.

One prominent example of meta RL is the MAML algorithm (Finn et al., 2017). MAML is designed to allow agents to quickly adapt to new tasks. By backpropagating through RL updates on a variety of tasks, MAML is able to learn parameters which are a good starting point for reinforcement learning in unseen domains. RL^2 is another well-known algorithm which provides an alternative approach to meta RL (Duan et al., 2016). RL^2 encodes a "fast" RL algorithm in the parameters of a recurrent neural network, using an existing "slow" RL algorithm to optimize these parameters. The resulting learned algorithm can achieve results comparable to hand-designed algorithms and can adapt its policy after each timestep. Also relevant to this thesis is the recent work of Saleh et al. (2022) which introduces a model-based meta-RL algorithm in which the learned model is optimized to produce experience which is useful to the learner, rather than experience that accurately reflects the real environment.

Often, meta RL algorithms are referred to as having an *outer-loop* and an *inner-loop*. The inner-loop optimizes the learner's parameters, for example, this could be Q-Learning updates which adjust the Q-values of an agent interacting with an MDP. The outer-loop optimizes the learner's meta-parameters; a classic example is the optimization of the hyperparameters of the learner by taking gradients through the learner's updates (Bengio, 2000).

Especially relevant to this work is the Bootstrapped Meta Gradient (BMG) algorithm (Flennerhag et al., 2022). Often, during meta-optimization, very computationally expensive derivatives are required to be taken through learner updates. These expensive derivatives can limit the number of learner updates included in the meta-objective, ultimately, prohibiting the meta-learner from considering the learning dynamics of inner-loop updates which are far into the future.

BMG provides a method of extending the horizon of learner updates used

in the meta-objective without increasing the computation needed to propagate the derivative through these updates. In BMG, the meta-objective is to increase the similarity between the learner's parameters after k updates, and a set of target parameters. By optimizing the meta parameters to increase this similarity, the meta optimization process can make adjustments to the learner's update rule. A key element of BMG is the use of a *target bootstrap*[2]. The bootstrapping procedure constructs the target parameters by applying non-differentiable updates to the learner's parameters. That is, a differentiable parameter vector is constructed by applying k updates to the learner's parameters. Then the bootstrapped target vector is constructed by applying $(l-1)$ further, *non-differentiable* updates to the differentiable parameters. Increasing the similarity between the differentiable parameters and the bootstrapped target parameters is the objective of the meta-learner. Using bootstrapping to construct the target parameters thus increases the horizon of learner updates considered in the meta-objective without increasing the number of learner updates through which derivatives are taken.

We now explain the bootstrapped meta gradient update rule in detail. Let $\boldsymbol{\theta} \in \mathbb{R}^n$ be the parameters of the learner, while $\boldsymbol{\eta} \in \mathbb{R}^m$ are the meta parameters. In BMG, updates to the learner's parameters $\boldsymbol{\theta}$ are a function of the meta parameters $\boldsymbol{\eta}$. By applying k updates to the learner's parameter vector, we construct $\boldsymbol{\theta}^{(K)}$. Gradients of $\boldsymbol{\eta}$ can flow back through these k updates. An additional $(l-1)$ updates are applied to $\boldsymbol{\theta}^K$ before a final gradient descent step on the learner's objective is taken. This produces the target parameter vector $\hat{\boldsymbol{\theta}}$. Note that gradients of $\boldsymbol{\eta}$ do not flow through these updates.

We now arrive at the update rule

$$\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} - \beta \nabla_{\boldsymbol{\eta}} \left\| \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(K)}(\boldsymbol{\eta}) \right\|_2^2 \tag{2.6}$$

where $\beta \in \mathbb{R}^+$ is the meta learner's step size[3]. By taking the gradient of squared L^2-norm with respect to $\boldsymbol{\eta}$, BMG moves $\boldsymbol{\eta}$ in a direction which reduces

[2]BMG borrows from the concept of bootrapping in RL in which an agent updates its value function or policy using its current estimates of these quantities.

[3]Note that we use the squared L^2-norm as a relevant example. In general, any matching function $\mu : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$, can be used to measure the similarity between the target parameters $\hat{\boldsymbol{\theta}}$ and the differentiable parameters $\boldsymbol{\theta}^{(K)}$.

the difference between the target parameters and the learner's parameters after k updates. This causes $\boldsymbol{\eta}$ to produce updates which move $\boldsymbol{\theta}$ more quickly toward the target parameters. The $(l-1)$ updates of the bootstrap are not differentiated through, allowing learning dynamics from far into the future to be incorporated into the target parameters without causing the computation of derivatives to scale with l.

Bootstrapped meta gradients provided significant inspiration for our work, and we discuss the influence of BMG on our algorithm in Chapter 4.

Chapter 3

Principles of Search Control

In this chapter we examine principles that result in effective search control. We believe search control distributions which implement these principles can allow agents to make more efficient use of model experience. First, let us introduce the TMaze environment, the domain in which our experiments are conducted throughout this work. We then discuss our first principle: focus on states of high value error. We conclude by considering the case of an agent with an imperfect model of the environment. In this situation, we outline the second principle: avoid incorrect states when the model is misleading.

3.1 The T-Maze Environment

Throughout this work we evaluate algorithms in the TMaze; an episodic grid-world environment pictured in Figure 3.1. We chose the TMaze domain for our

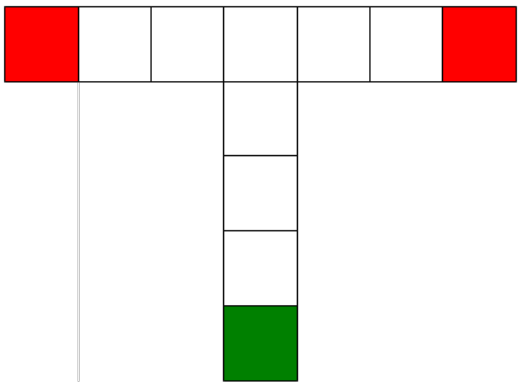


Figure 3.1: The T-Maze environment. The green state indicates the agent's starting state while the red states indicate terminal states.

experiments because its simplicity makes it amenable to analyzing the effects of different search control distributions. However, the TMaze is also a non-stationary and stochastic domain where agents must be capable of adapting in order to achieve maximum performance. This combination of simplicity with stochasticity and non-stationarity results in a domain which is both easily analyzed while incorporating characteristics of challenging RL problems.

In the TMaze, an agent begins at a starting state and must navigate a vertical hallway, then turn either left or right into a horizontal hallway. Reaching the leftmost or rightmost state of the horizontal hallway results in the termination of an episode. At any time, one of these 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, so long as the agent's state representation does not track episodes or timesteps. The TMaze is also stochastic. At any timestep, regardless of what action the agent selects, the agent may transition to a random adjacent state with probability ϵ_{env}. We set $\epsilon_{\text{env}} = 0.1$ for the remainder of this work.

A key element of the TMaze is that under the optimal policy only the values of certain states change when the reward regime is swapped between terminal states. For the optimal policy, the values of states do not change along the vertical hallway when the reward switches, while the values of states along the horizontal hallway do change. This is an important observation that will motivate our understanding of which search control distributions perform well on the TMaze.

3.2 Principle 1: Focus on High Value Error States

Let us first define *value error* as the difference between the value of a state s under the optimal policy and the value of that same state under the agent's current policy: $v^*(s) - v_\pi(s)$. We now propose that the first principle for effective search control is for the agent to *focus its planning on states where*

value error is relatively high.

Why should an agent focus its planning updates on these states? We can answer this question by considering an agent learning in the TMaze environment. As described previously, the TMaze is a non-stationary domain with two reward regimes. We refer to the reward regime where the left terminal state emits +1 and the right terminal state emits 0 as *reward-left* and the opposite reward regime as *reward-right*. Furthermore, we denote the optimal value functions for reward-left and reward-right respectively as v_L^* and v_R^*.

Picture v_L^*; the states along the horizontal portion of the maze have highest value at the left (next to the rewarding terminus) and descend in value towards the right. States along the vertical portion of the maze have highest value at the top and descend in value toward the bottom, reaching the lowest value at the starting state. In contrast v_R^* will have the highest value on states at the right of the horizontal portion of the maze, with values descending towards the left. However, the value of states in the vertical portion of the maze is the same as in v_L^*. This is because the number of transitions between any of these states and the terminal state with positive reward does not change when the reward regime changes.

Consider an agent which has been training in the TMaze under the reward-left regime and imagine that the agent has approximately learned v_L^*. Further, suppose the agent is equipped with a perfect model of the environment. When the regime changes to reward-right, the agent's value function will no longer be correct. The agent must update its value function towards v_R^* as quickly as possible, making use of its model to accelerate learning. If the agent samples states along the vertical hallway, at best, updates that it makes will not affect its value function at all. There is 0 value error for these states as v_L^* and v_R^* are identical along the vertical portion of the maze. On the other hand, the states along the horizontal portion of the maze have high value error. Were the agent to correct its value function, its policy would guide it to the positively rewarding terminal state. Hence, the agent should spend its planning updates on the high value error states of the horizontal portion of the maze.

We can validate this principle experimentally. Figure 3.2 shows the two

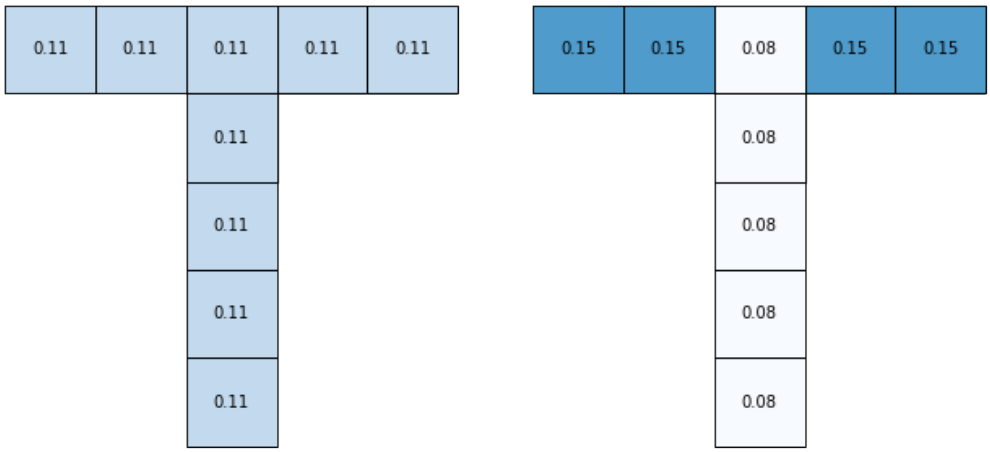


Figure 3.2: (a) The Uniform prioritized distribution. (b) The Horizontal Focus prioritized distribution. Terminal states are not pictured as no probability is assigned to these states. Darker colors indicate greater probability mass while text indicates the probability of sampling the corresponding state.

search control distributions in consideration. The Uniform distribution has equal probability of sampling any state as a starting point for a planning update. The Horizontal Focus distribution places greater probability on the states in the horizontal portion of the TMaze, with less probability allocated to the vertical states. The Horizontal Focus distribution thus places greater probability on states which will have higher value errors after a reward regime change.

We evaluate the performance of different algorithms on the TMaze by examining the total reward they accumulate in a limited number of episodes. This metric captures both the speed at which agents learn to reach terminal states and the behaviour of correctly reaching the positive rewarding state. Figure 3.3 shows the total reward achieved by two agents in the TMaze against the number of planning steps the agents perform after each real update. Uniform and Horizontal Focus are both Dyna-Q agents equipped with the respective search control distributions shown above. It is clear that Horizontal Focus outperforms Uniform across all amounts of planning in consideration. This result demonstrates that a search control distribution which implements the principle of focusing on high value error states achieves greater results when planning with a perfect model in the TMaze environment.

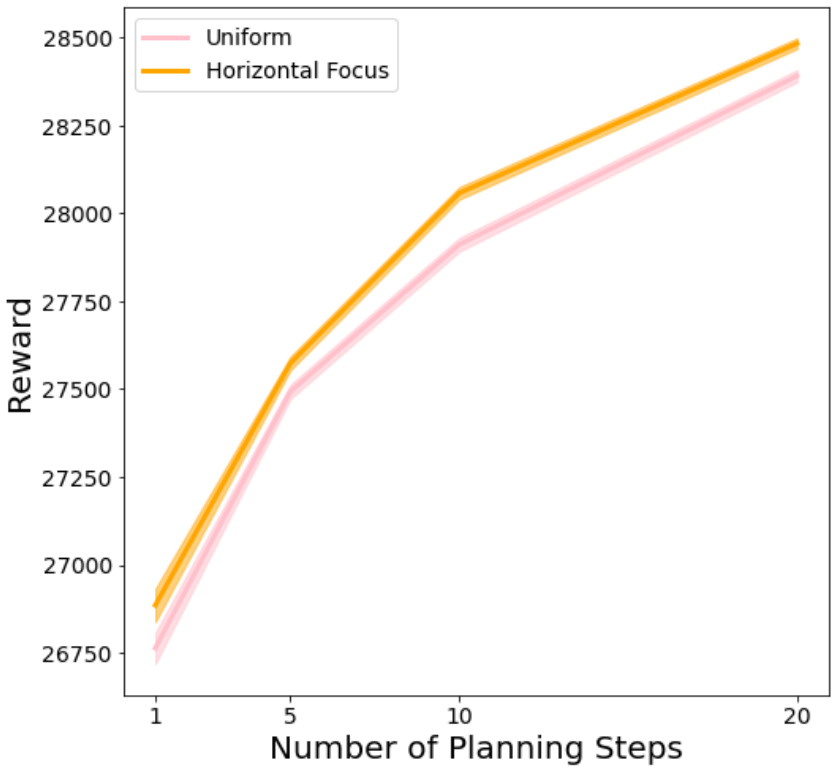


Figure 3.3: The reward accumulated during training against the number of planning steps per real update.

3.3 Principle 2: Avoid Incorrect States

We now consider a situation where the agent is a equipped with an imperfect model of the environment, whether learned or provided a priori. Similar to the introduction of the first principle, we perform a thought experiment to demonstrate the benefits to planning efficiency which stem from avoiding incorrect states.

Consider an agent equipped with a perfect model of the TMaze with the exception of the terminal transitions. Transitions into terminal states of this model emit a reward of either 0 or +1 with equal probability. The reward dynamics of this model thus match the empirical distribution of outcomes of the real environment averaged over both reward regimes. As the reward dynamics of this model do not match those of the environment what effect does this have on the agent's learning?

Imagine that the agent has learned v^*_L, at this moment the reward regime switches from reward-left to reward-right. To maximize reward, the agent should update its values towards v^*_R as quickly as possible. If the agent samples the terminal transitions of the model, in expectation it will receive a reward of 0.5. Updating its value function with these rewards will cause the value of terminal-adjancent states to approach 0.5. The value of these states will move away from the correct values of v^*_R. However, what if the agent simply ignored the terminal transitions of the model? In this case, the value of the terminal-adjacent states will only be updated by real experience. These values will thus approach the correct optimal values as more real updates are performed. The agent can still use model experience to quickly propagate changes in the value function back to other states, without learning the incorrect values. Hence, we arrive at the second principle: *an agent should avoid sampling states with incorrect information.*

We demonstrate the effects of this principle through a new experiment in the TMaze. Each agent is equipped with the imperfect model described above and performs five steps of planning after each real update. Figure 3.4 introduces a new search control distribution we refer to as Avoid Terminal.

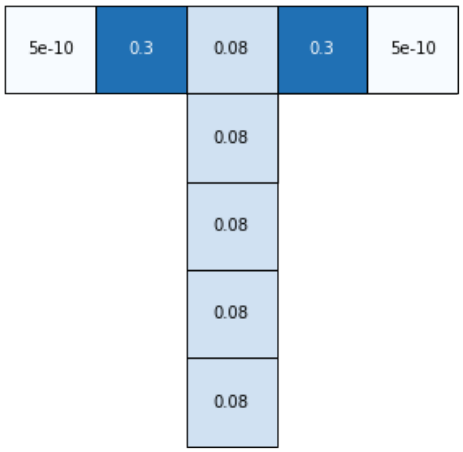


Figure 3.4: The modified Horizontal Focus distribution used in the imperfect model setting. We refer to this distribution as Avoid Terminal. This distribution places near-zero probability on states which transition into terminal states. Darker colors indicate greater probability mass while the text indicates the probability of sampling the corresponding state.

Following the first principle, this distribution places greater probability on states along the horizontal portion of the TMaze but, in accordance with the second principle, places near-zero probability on states adjacent to terminal states. In effect, these states will never be sampled during planning.

Figure 3.5 shows a comparison of total rewards for the agents considered in this experiment. The results starkly demonstrate the effects that different search control distributions may have in the presence of the imperfect model. Q-Learning is outperformed by all the model-based agents. While Horizontal Focus is outperformed by the naive Uniform distribution, Avoid Terminal achieves much greater total reward than either Uniform or Horizontal Focus. This difference in total reward illustrates that avoiding the incorrect states of the model can be important for planning efficiently — obeying the first principle is not enough in this setting.

While both principles introduced in this chapter can improve planning efficiency, they rely on privileged information of the environment. In complex environments where the application of these principles may not be obvious, or in the more general case where expert knowledge of the environment is not available, an open question remains: can agents discover search control

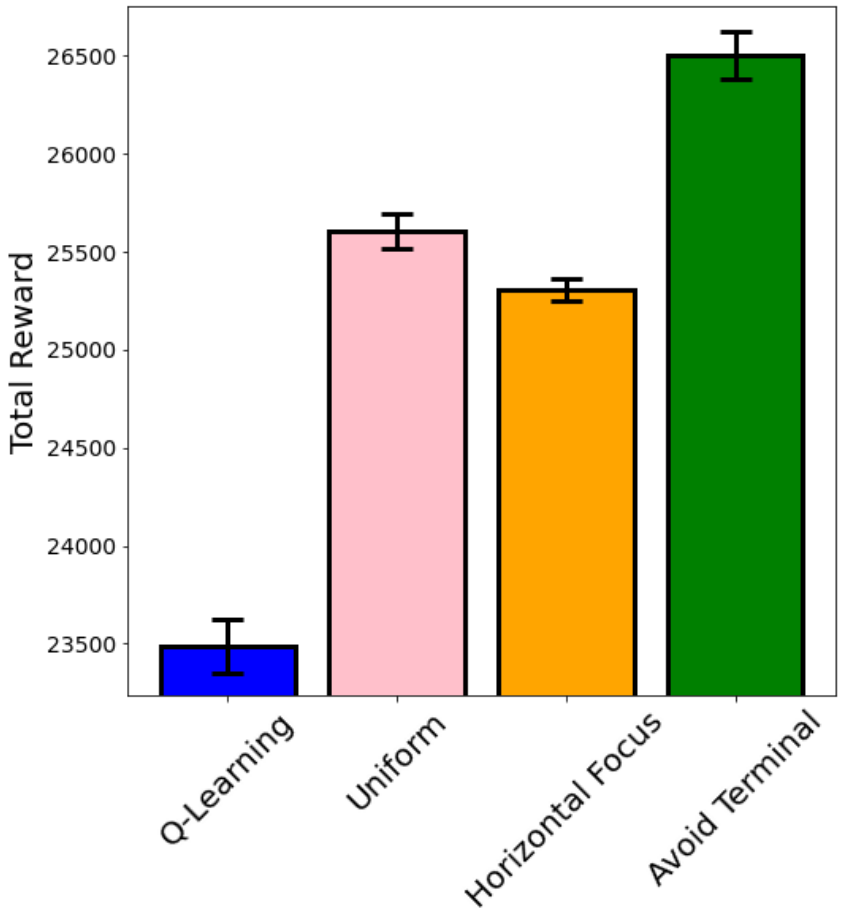


Figure 3.5: A comparison of the total reward accumulated by each agent when using the imperfect model under different search control distributions.

distributions that embody these principles and enable efficient planning?

Chapter 4

Learning Search Control Distributions

We now examine a means of learning search control distributions without prior knowledge of the environment. First, we introduce the Meta-Gradient Search Control (MGSC) algorithm. We describe the motivation behind MGSC and discuss the derivation of the algorithm's meta-objective. We then present pseudocode along with a detailed explanation for a Dyna-style agent which incorporates MGSC.

4.1 The Meta-Gradient Search Control Algorithm

We introduce Meta-Gradient Search Control (MGSC), an online meta-learning algorithm capable of learning useful search control strategies. In this section we derive the MGSC loss function and discuss how it can be integrated into the Dyna architecture.

We present a thought experiment which motivates the meta-learning objective of MGSC. Let us refer to the meta-parameters as $\boldsymbol{\eta}$ and the learner's parameters as $\boldsymbol{\theta}$. We introduce a distribution $d(\boldsymbol{\eta})$ from which an agent can sample a starting state $\tilde{s}$. To make the most efficient use of each planning step, the system should sample the state that produces the most beneficial transition for learning. But how does the agent evaluate which transitions are the most beneficial? One way of answering this question is to choose the tran-

sition which brings the learner's parameters closest to the optimal parameters $\boldsymbol{\theta}^*$. We define closeness as the Euclidean distance between the parameter vectors. The agent constructs a parameter vector $\bar{\boldsymbol{\theta}}$ by sampling states $\tilde{s} \sim d(\boldsymbol{\eta})$ and applying planning updates from these states to its parameters $\boldsymbol{\theta}$. The usefulness of these planning updates can be calculated as

$$||\boldsymbol{\theta}^* - \bar{\boldsymbol{\theta}}(\boldsymbol{\eta})||_2^2. \tag{4.1}$$

As $\bar{\boldsymbol{\theta}}$ is a function of $\boldsymbol{\eta}$, the agent can optimize $\boldsymbol{\eta}$ to reduce the parameter error and improve the effectiveness of planning updates.

In a real setting, the agent obviously does not have access to the optimal parameters. It is thus necessary to replace the optimal parameters $\boldsymbol{\theta}^*$ with an approximate target $\hat{\boldsymbol{\theta}}$. We use the most recent transition of experience from interacting with the environment (s, a, r, s') to construct the target parameters $\hat{\boldsymbol{\theta}}$. Specifically, we construct the target $\hat{\boldsymbol{\theta}} = \bar{\boldsymbol{\theta}} + \alpha\delta(s, a, r, s', \bar{\boldsymbol{\theta}})$, where $\delta(s, a, r, s', \bar{\boldsymbol{\theta}})$ is a shorthand for a semi-gradient Q-Learning update. This target was motivated by the Bootstrapped Meta-Gradients work of Flennerhag et al. (2022).

As discussed in Section 2.8, BMG creates a target for its meta-objective by bootstrapping estimates of the learner's future parameters from the learner's current parameters. Similarly, MGSC's target is bootstrapped from the current value function parameters of the learner by applying one real update. In contrast to MGSC, BMG constructs its target using a further $(l-1)$ updates sampled from the learning rule it is optimizing. We experimented with including such updates in MGSC's target (in our case these would correspond to additional planning updates) but our results indicated superior performance using only the real transition. We believe that in the MBRL setting, inaccuracies in the model can result in erroneous target parameters, making the approximate target worse and resulting in the agent learning a poor search control distribution when planning updates are used to construct the target.

The search control distribution $d(\boldsymbol{\eta})$ is modelled as a softmax distribution where each non-terminal state is represented as a logit. Planning updates cannot begin from a terminal state and so we remove them from the distribution.

The distribution is given as:

$$d(s|\boldsymbol{\eta}) \triangleq \frac{e^{\boldsymbol{\eta}_s}}{\sum_{i=1}^{|\mathcal{S}|} e^{\boldsymbol{\eta}_s}} = \mathbb{P}(s|\boldsymbol{\eta}).$$

In many domains, the number of states may be very large and representing each state by a logit can become infeasible. We discuss how future work could address this issue later in this document.

We use an expectation update over all states and actions to construct $\bar{\boldsymbol{\theta}}$. This choice allows all of the logits to be affected by each meta-update. Under a tabular state representation, a sample-based update would only adjust a single logit during each meta-update, thus our expectation update can accelerate the meta-learning procedure. The drawback to this approach is that an expectation update requires a sample from every state-action pair. This can be infeasible when there is a large number of states and actions.

A model transition is given by sampling a starting state from the learned search control distribution, an action from the current policy, and a next state and reward from the dynamics model, that is: $\tilde{s} \sim d(\cdot|\boldsymbol{\eta}), \tilde{a} \sim \pi(\cdot|\tilde{s}), \tilde{s}', \tilde{r} \sim m(\cdot|\tilde{s}, \tilde{a})$. Using this transition, we define the semi-gradient Q-Learning update as

$$\delta(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}', \boldsymbol{\theta}) \triangleq [\tilde{r} + \gamma \max_{\tilde{a}' \in \mathcal{A}} \hat{q}(\tilde{s}', \tilde{a}'; \boldsymbol{\theta}) - \hat{q}(\tilde{s}, \tilde{a}; \boldsymbol{\theta})] \nabla_{\boldsymbol{\theta}} \hat{q}(\tilde{s}, \tilde{a}; \boldsymbol{\theta}). \tag{4.2}$$

We use an analogous definition, $\delta(s, a, r, s', \boldsymbol{\theta})$ to refer to a semi-gradient Q-Learning update on real experience.

Our expectation update is taken over all state-action pairs in the environment. The update for each pair is weighted by the probability of selecting the state according to the current search control distribution $d(\boldsymbol{\eta})$ and of selecting the action from the current behaviour policy π. We then apply a fixed step size $\alpha \in \mathbb{R}_+$ to the weighted sum of the updates to obtain

$$\bar{\boldsymbol{\theta}}(\boldsymbol{\eta}) \triangleq \boldsymbol{\theta} + \alpha \sum_{\tilde{s}, \tilde{a}} \pi(\tilde{a}|\tilde{s}) d(\tilde{s}; \boldsymbol{\eta}) \delta(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}', \boldsymbol{\theta}). \tag{4.3}$$

We now arrive at our meta-loss function: the squared Euclidean error between the target parameters $\hat{\boldsymbol{\theta}}$ and the parameters from the expectation

update $\bar{\boldsymbol{\theta}}$:

$$\mathcal{L}(\boldsymbol{\eta}) \triangleq ||[\![\hat{\boldsymbol{\theta}}]\!] - \bar{\boldsymbol{\theta}}(\boldsymbol{\eta})||_2^2. \tag{4.4}$$

We must prevent the target term from being optimized by back-propogation, otherwise the algorithm could spuriously reduce the loss. A stop-gradient denoted by brackets $[\![\cdot]\!]$ is applied to the target parameters to halt this from occurring.

The experiments in this work minimize the MGSC loss (4.4) using the Adam optimizer (Kingma & Ba, 2015). Gradients are back-propagated through the parameters $\bar{\boldsymbol{\theta}}$ to optimize the meta-parameters $\boldsymbol{\eta}$ and thus influence the learned search control distribution.

Our loss function encourages the meta-learner to produce a search control distribution which results in planning updates bringing the parameters of the value function closer to optimality. Allocating greater probability mass to states which produce helpful updates will generally incur a lower loss as the difference between $\bar{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}$ will be decreased. This loss is however sensitive to the construction of the target parameters. While we found that a single real update was effective, in highly stochastic environments, or under a very poor model, the target may require more real updates to provide a good learning signal.

4.2 Meta Gradient Search Control in Dyna

Algorithm 2 presents the MGSC procedure integrated into the Dyna architecture. The algorithm is presented in a general form for continual domains, but can be applied without modification to episodic domains. We now provide a detailed explanation of the algorithm.

MGSC in Dyna-Q is an online algorithm which begins in a starting state s_1 determined by the environment. For each timestep t, the algorithm selects an action according to an ϵ-greedy policy and takes this action in the environment, receiving the real experience tuple (s, a, r, s'). Next, this experience is used to update the agent's model — MGSC is agnostic to the model update procedure. As shown on line 7, the algorithm then executes a semi-gradient Q-Learning

Algorithm 2 Meta Gradient Search Control in Dyna-Q

1: Given agent parameters $\boldsymbol{\theta}$ and meta-parameters $\boldsymbol{\eta}$.
2: Obtain initial s.
3: **for** $t = 1, 2, 3, \cdots$ **do**
4: Take ϵ-greedy action a from s then obtain s' and r.
5: $m \leftarrow \text{UpdateModel}(m, s, a, s', r)$
6: # Perform a real update
7: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta(s, a, r, s', \boldsymbol{\theta})$
8: # Perform k planning updates
9: **for** $1, \cdots, k$ **do**
10: Take ϵ-greedy $\tilde{a}$ from $\tilde{s} \sim d(\cdot|\boldsymbol{\eta})$.
11: $\tilde{s}'$, $\tilde{r} \sim m(\cdot|\tilde{s}, \tilde{a})$.
12: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}', \boldsymbol{\theta})$
13: # Construct differentiable parameters according to (4.3)
14: $\bar{\boldsymbol{\theta}}(\boldsymbol{\eta}) \leftarrow \boldsymbol{\theta} + \alpha \sum_{\tilde{s},\tilde{a}} \pi(\tilde{a}|\tilde{s}) d(\tilde{s}; \boldsymbol{\eta}) \delta(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}', \boldsymbol{\theta})$
15: # Construct target parameters
16: $\hat{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta(s, a, r, s', \bar{\boldsymbol{\theta}})$
17: Update $\boldsymbol{\eta}$ with Adam on the MGSC meta-loss using $\hat{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{\theta}}$ (4.4).

update to the learner's value function using the real experience. The direct learning portion of the algorithm is now complete.

The algorithm now enters the planning portion of its execution on lines 9 to 12. Each planning update follows the same structure. A state $\tilde{s}$ is sampled from $d(\boldsymbol{\eta})$, the algorithm then uses its current ϵ-greedy policy to select an action $\tilde{a}$. The algorithm queries its model m with $\tilde{s}$ and $\tilde{a}$, receiving a reward $\tilde{r}$ and next state $\tilde{s}'$. Using this model experience tuple $(\tilde{s}, \tilde{a}, \tilde{r}, \tilde{s}')$ the algorithm performs a semi-gradient Q-Learning update to the learner's value function parameters. The hyperparameter k controls the number of planning updates.

Following the planning updates, the parameter vectors provided to the meta-loss are constructed. Lines 14 and 16 show the construction of the differentiable parameters $\bar{\boldsymbol{\theta}}$ and the target parameters $\hat{\boldsymbol{\theta}}$ respectively. Note that the sum over all state-action pairs of line 14, these states and actions are distinct from those used in the planning procedure. However, they are still referred to as $\tilde{s}$ and $\tilde{a}$ as these are the states and actions of the model, not the real environment. The experience tuple used to construct $\hat{\boldsymbol{\theta}}$ in line 16 is the same tuple of real experience used in the real update of line 7. Finally, line 17 describes the update to the meta-learner's parameters $\boldsymbol{\eta}$ using the parameter

vectors $\bar{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}$. The algorithm updates $\boldsymbol{\eta}$ according to the Adam learning rule.

Let us discuss the scalability of Algorithm 2. We note that the semi-gradient Q-Learning updates used in direct learning and planning can extend to the case of non-linear function approximation with neural networks. Similarly, the meta-optimization with Adam also extends to non-linear function approximation. The expectation update used to construct $\bar{\boldsymbol{\theta}}$ could be approximated using a sufficient number of samples, freeing the method from requiring a sample from every state-action pair in the environment. We point out these properties as they suggest that Algorithm 2 is amenable to scaling to more complex domains which necessitate neural network function approximation.

Chapter 5

Experimental Results

In this chapter, we provide experimental results comparing the performance of baseline agents against an MGSC agent. We begin in a setting where agents have access to a perfect model of the environment. We then move on to a setting where the agents have access to an imperfect model, and finally, a learned model. Throughout we discuss the performance of MGSC in comparison to baseline agents and study the search control distributions learned by the algorithm.

There are some experimental details which are common to all of our experiments. Each experimental run lasts for 250,000 timesteps while all results are averaged over 30 random seeds. All error bars indicate 95% confidence intervals over these random seeds. We swept over hyperparameter settings to determine the best settings for each agent and use only these results for comparison. The full details of the hyperparameter sweep are included in the Appendix.

Our primary metric of evaluation is the total reward: the sum of all rewards received during training. The TMaze is an episodic environment in which each episode has no maximum number of timesteps. The total reward thus captures an agent's ability to arrive at the positively rewarding terminus quickly; agents which require fewer steps to reach the rewarding terminal state will accumulate more reward in the fixed number of timesteps allowed during training. We also examine the average reward over the course of training. At timestep t, the average reward is given by $\frac{1}{t}\sum_{i=0}^{t} r_i$. Average reward provides a useful

metric for plotting changes in the agent's performance as training proceeds.

5.1 The Perfect Model Setting

In this section, we compare the performance of agents which plan with a perfect model — a model which exactly matches the reward dynamics and state transitions of the environment. This is the same model we considered when introducing the first search control principle.

5.1.1 Experimental Setup

In this experiment, all agents are tasked with navigating the TMaze, but model-based agents may plan with a model which is synchronized with the reward regime of the real environment. Because the model exactly matches the environment, model experience is equivalent to real experience. As such, planning updates should always be beneficial to an agent's learning.

We compare the MGSC-Dyna agent described in Algorithm 2 against three baseline agents: Q-Learning, Uniform, and Horizontal Focus. As described in Section 2, Q-Learning is a model-free baseline which learns Q-values solely from real experience (Watkins & Dayan, 1992). Uniform and Horizontal Focus are both Dyna-style agents which implement Algorithm 1 and learn from both real and model experience. These agents were previously introduced in Chapter 3. Uniform's search control distribution places equal probability on all starting states. This distribution is an uninformed baseline available to all MBRL systems. Horizontal Focus' search control distribution embodies our principle of focusing on high value error states. Refer to Figure 3.2 for a visualization of the search control distributions of these agents. We expect the Horizontal Focus agent to outperform the Q-Learning and Uniform agents. Given the same finite planning budget, Horizontal Focus' increased emphasis on high-value error states should allow the agent to correct its Q-values faster than other baselines after a reward regime change occurs.

5.1.2 Results and Discussion

Figure 5.1 shows the total reward and average reward for each agent when planning with the perfect model. We report the results for the hyperparameter setting of ten planning updates per real update as MGSC achieved the greatest performance relative to the baselines at this level of planning. Results for the remaining hyperparameter settings can be found in the Appendix. Q-Learning, the model-free agent, achieves noticeably less total reward and average reward compared to the model-based agents. Among the model-based agents, we observe that the performance differences are much smaller, with Horizontal Focus achieving the best performance by a small margin. We performed a Welch's t-test on the total reward of MGSC and Uniform, finding that the total reward of MGSC is statistically significantly greater than that of Uniform (p-value 0.00049)(Welch, 1947). However, we note that we do not find this difference in performance to be scientifically significant — the closeness of the total reward of all three model-based algorithms does not suggest strong differences in performance between search control distributions in the perfect model setting.

Figure 5.2 shows snapshots of the distribution learned by MGSC over the course of training. While the probability mass does not shift drastically, we can observe that some probability moves away from the vertical states and towards the horizontal states. This behaviour mirrors the first principle of search control described in Chapter 3, placing greater probability mass on states with larger value error. While Horizontal Focus has a search control distribution which was hand-designed to instantiate this principle, MGSC shows signs of discovering this principle purely through the optimization of its meta-objective.

Figure 5.3 shows the total reward accumulated by the Uniform, Horizontal Focus, and MGSC agents as the number of planning updates per real update is varied. Q-Learning is not pictured as it achieves much less total reward than the model-based agents. We omit it to allow clear illustration of the performance of the model-based agents.

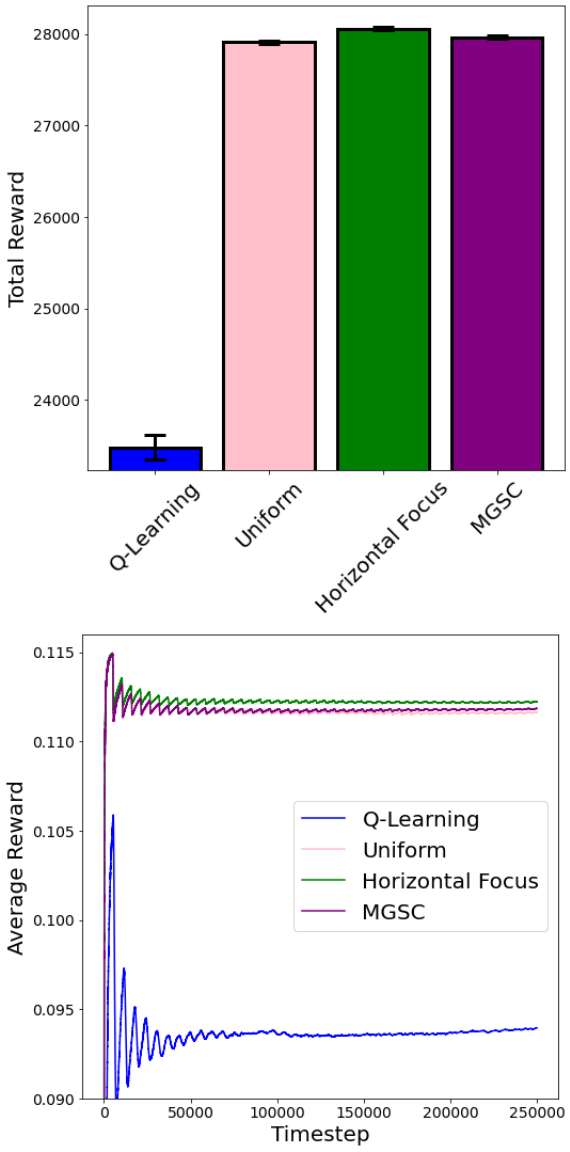
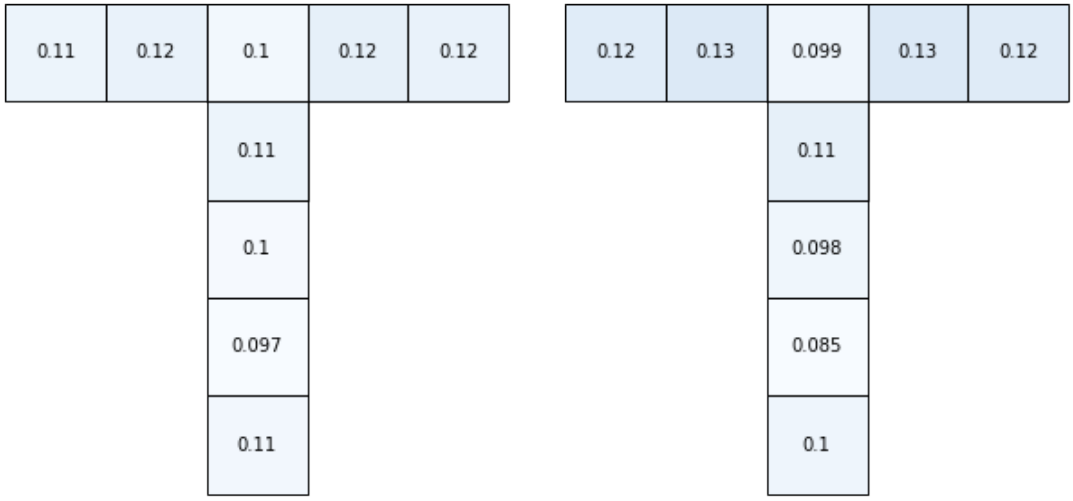


Figure 5.1: Results of agents planning with a perfect model. (a) The total reward accumulated by each agent over the course of training. Error bars denote the 95% confidence interval. (b) The average reward accumulated during training for each agent.



(a) 25% of Training

(b) 50% of Training

(c) 75% of Training

(d) 100% of Training

Figure 5.2: Plots of the average distribution learned by MGSC at different points during training when planning with a perfect model. The distribution shown is averaged over 30 random seeds.

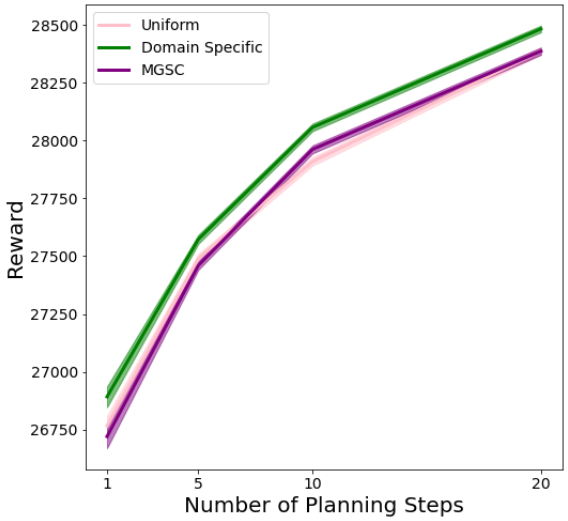


Figure 5.3: The total reward accumulated by each agent as the number of planning updates per real update is varied.

We observe that Horizontal Focus consistently achieves the greatest total reward at all levels of planning. MGSC achieves marginally greater total reward than Uniform in the regime of 10 planning steps, but performs comparably to Uniform in the other regimes considered. Overall, Figure 5.1 indicates that under a perfect model, MGSC does not show consistent improvement over baselines at all levels of planning. Here, learning the search control distribution does not appear to have much benefit, nor cause much harm, when compared to static baselines. But, even the hand-crafted Horizontal Focus distribution achieves only slightly better performance than the Uniform distribution. However, does a learned search control distribution provide greater benefit to an agent than the uniform distribution in the absence of a perfect model? We will study this question in the following section.

Finally, our analysis revealed that all the model-based agents follow a similar performance curve as the number of planning steps increases, with total reward increasing at each level of planning. This behaviour is expected, as experience from a perfect model is identical to real experience. The shape of the curves in Figure 5.3 matches our expectations. An infinite amount of planning would allow an agent to learn an optimal policy without any additional real experience, causing total reward to plateau. At the other extreme,

with no planning, the model-based agents should perform identically to the Q-Learning agent. Additionally, while we evaluate the agents in a relatively low-planning regime, we note that as the amount of planning grows we expect the performance of the model-based agents to converge. With an infinite amount of planning, the search control distribution no longer has an effect (so long as it places non-zero probability on every state).

5.2 The Imperfect Model Setting

We now present an empirical evaluation of the effects of search control distributions when agents plan with an imperfect model. Planning with such a model can inhibit an agent's ability to properly learn the optimal Q-values of the environment. Agents which can avoid querying the portions of the model where the dynamics do not match the environment are poised to achieve greater performance.

5.2.1 Experimental Setup

The *imperfect model* was previously introduced in Chapter 3. It has exactly the same transition probabilities as the TMaze, but differs in reward dynamics. In the imperfect model, transitioning into either terminal state randomly produces a reward of 0 or 1. This means the agent receives a reward of 0.5 in expectation from both terminal transitions, and, that the reward dynamics of the model do not match the reward dynamics of the environment. It follows that an agent which learns the optimal Q-values of the model will not have learned the optimal Q-values of the environment. It is unlikely that this agent's policy would be highly performant in the real environment.

In the imperfect model setting, we compare MGSC against a new baseline agent: Avoid Terminal. This agent was previously introduced in Chapter 3. It instantiates both principles of search control by using the distribution pictured in Figure 3.4. By placing greater probability mass on horizontal states where values change after reward regime switches, the distribution embodies the first principle. By placing near-zero probability on states which transition into

terminal states of the imperfect model, this distribution embodies the second principle. The combination of these attributes allows Avoid Terminal to learn from model experience which has the greatest benefit to planning efficiency while avoiding experience which would be detrimental to learning optimal Q-values. In the experimental results that follow, we report a comparison between Q-Learning, Uniform, Avoid Terminal and MGSC.

5.2.2 Results and Discussion

Experimental results for the imperfect model setting are shown in Figure 5.4. This figure reports results for agents which perform five planning updates per real update. We chose to report results with this level of planning as we observed the greatest disparity in performance between MGSC and the baseline algorithms.

We observe that the total reward accumulated by the Uniform and Avoid Terminal agents matches our intuition for what kind of distribution will be effective in this setting. The relatively low performance of Uniform suggests that sampling terminal transitions at a high rate is harmful in the imperfect model setting, as expected. In contrast, the much higher performance of Avoid Terminal further supports the usefulness of circumventing these transitions. Avoid Terminal achieves the greatest performance of all the agents in consideration. As expected, its search control distribution allows it to quickly adapt its policy after reward regime changes leading to greater total reward.

Over the course of training, MGSC achieves greater total reward than Uniform and is close to that of Avoid Terminal. Figure 5.4b shows that after an initial transient period, MGSC arrives at a much higher average reward than the Uniform baseline. Ultimately, MGSC's average reward plateaus near the level of Avoid Terminal but does not quite reach its performance.

MGSC's search control distribution suggests that it has learned to shift probability away from states where the model is erroneous and would cause greater value error. Figure 5.5 shows the evolution of MGSC's learned search control distribution over the course of training. There are two notable features in this plot. Firstly, probability is concentrated away from terminal-adjacent

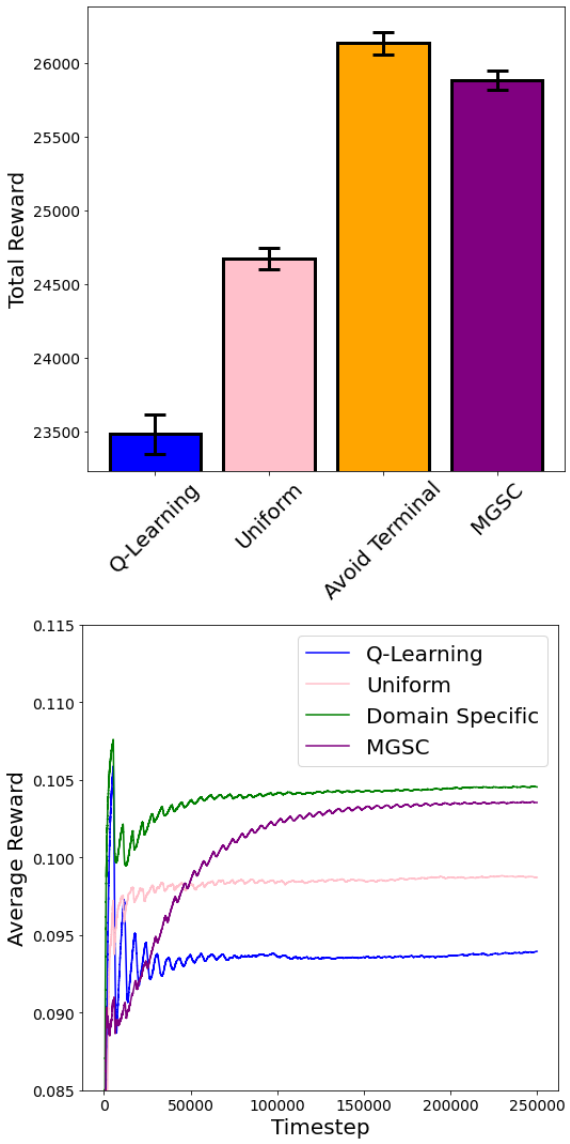
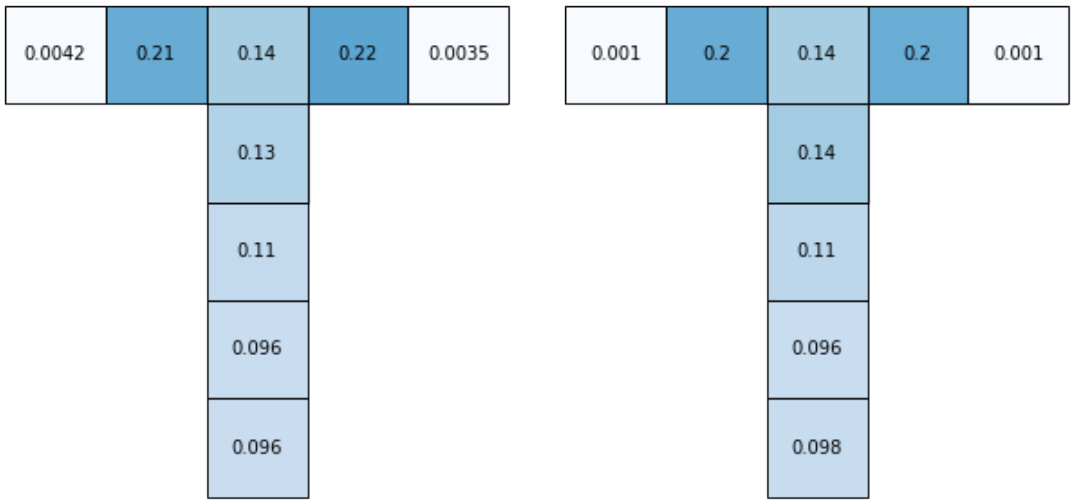


Figure 5.4: Both figures show results when planning with an imperfect model. (a) The total reward accumulated by each agent over the course of training. Error bars denote the 95% confidence interval. (b) The average reward accumulated during training for each agent.

states. The probability mass on these states is far less than on any of the other states in the distribution. Secondly, relatively greater probability is placed on the horizontal states than the vertical states. After 25% of training, this effect is already clear. As training proceeds some of this probability mass is shifted back to the vertical states — by the end of training some of the vertical states have close to the same probability mass as the horizontal states. It is unclear why this reallocation of probability towards the vertical states occurs. However, the movement of probability towards horizontal states and away from terminal-adjacent states reflects the agent's ability to discover the principles of search control. MGSC simply optimizes for a distribution which reduces the agent's parameter error, the learning mechanism allows it to generate a distribution with characteristics similar to that of Avoid Terminal, which was hand-designed using privileged information about the TMaze environment.

Figure 5.6 shows the total reward accumulated by each agent as the number of planning updates per real update is varied. We observe that MGSC achieves greater total reward compared to Uniform at all levels of planning. At most levels of planning, MGSC does not reach the performance of Avoid Terminal. One possible explanation for this behaviour is that MGSC has an initial transient period during the first few reward regime switches where large shifts in probability mass occur. This may be visible in MGSC's average reward shown in Figure 5.4b. Initially, it is much lower than the other model-based agents. Further, after 25% of training, MGSC's search control distribution already appears similar to its distribution at the end of training, suggesting that a large portion of probability mass shift occurs during this transient period. During this period, it is possible that MGSC is not able to accumulate reward as quickly as Avoid Terminal, ultimately resulting in MGSC's performance lagging behind that of Avoid Terminal in terms of total reward.

Notably, Avoid Terminal shows a large spike in variance between random seeds at the level of 20 planning steps. This behaviour is not observed in the other agents. It is possible that this behaviour is due to a few random seeds which lead to particularly poor runs for Avoid Terminal.



(a) 25% of Training

(b) 50% of Training

(c) 75% of Training

(d) 100% of Training

Figure 5.5: Plots of the distribution learned by MGSC at different points during training in the imperfect model setting.

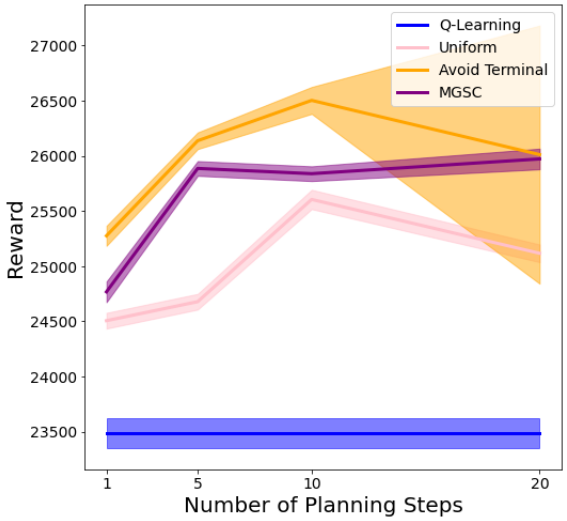


Figure 5.6: The total reward accumulated by each agent as the number of planning updates per real update is varied in the imperfect model setting. Note that Q-Learning's performance is not affected by planning. It is included here for reference.

5.3 The Learned Model Setting

We now consider a setting in which agents must learn their model simultaneously as they learn to navigate the TMaze. We refer to this as the *learned model setting*. This setting presents an additional challenge to agents beyond the imperfect model, as the learning process will result in the model shifting over time.

5.3.1 Experimental Setup

In this experiment, the model keeps a count for each value of reward observed during each transition in the real environment. When the model transitions from one state to another, a reward is sampled according to the empirical probability of observing each value. In the limit, this model will behave identically to the imperfect model. However, planning with the learned model presents multiple challenges. While it is biased in the limit, the model's reward distribution will also often be out of sync with the real environment. Particularly, in the first few reward regime switches, the model may lag behind the environment, producing rewards which more closely match the previous reward

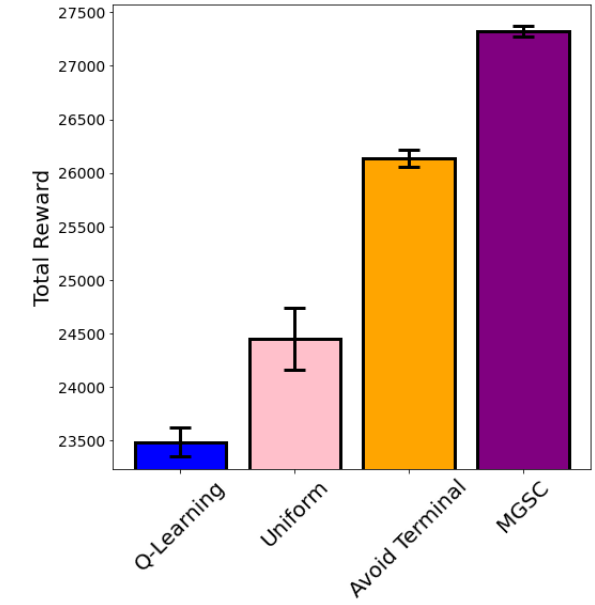
regime. Agents must compensate for both an erroneous and changing model to achieve strong performance in this setting. We use the same agents for comparison in this setting as in the imperfect model setting, examining Q-Learning, Uniform, Avoid Terminal, and MGSC.

5.3.2 Results and Discussion

We present the total reward and average reward observed in the learned model setting in Figure 5.7. We note that in this setting, MGSC appears to achieve the greatest total reward, followed by Avoid Terminal, Uniform, and finally Q-Learning. Examining the average reward, all the agents show a sharp decrease in performance after a few thousand timesteps: this corresponds to the first time the reward regime is switched. In particular, Uniform's average reward plummets at this moment. Uniform's large drop in average reward, and overall lower total reward, appear to reflect the importance of avoiding the terminal transitions in the learned model. In contrast, Avoid Terminal achieves more total reward and suffers less after the first reward regime change as its distribution skips these unhelpful transitions.

Figure 5.8 shows snapshots of the search control distribution learned by MGSC which help to explain its relatively strong performance. We observe that after 25% of training, probability has already moved away from the terminal transitions, and away from the state at the intersection of the vertical and horizontal hallways. Similar to observations in our prior experiments, this behaviour suggests that MGSC is able to learn a distribution which follows both principles of search control without explicit programming or privileged domain knowledge.

The snapshots also reveal some unexpected behaviours. MGSC's learned distribution places a large amount of probability on the starting state of the TMaze. This is surprising as the value of this state does not change after reward regime swaps. Seemingly, once the agent has learned this value, it could focus planning updates on other states where values change. This unexpected behaviour could be an artifact of MGSC's meta-loss; concentrating probability on this state may result in a lower meta-loss even if the distribution is sub-



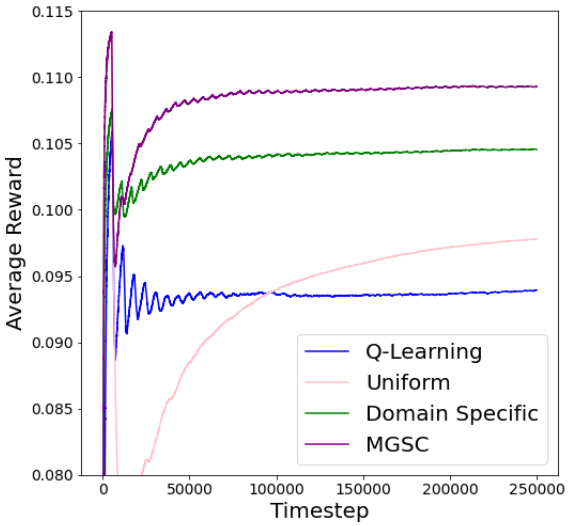
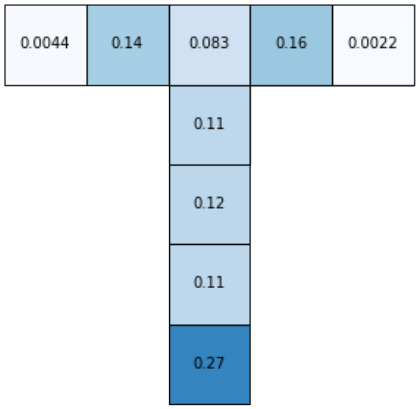


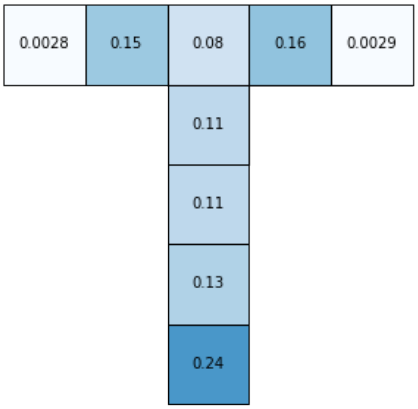
Figure 5.7: Both figures show results when planning with the learned model. (a) The total reward accumulated by each agent over the course of training. Error bars denote the 95% confidence interval. (b) The average reward accumulated during training for each agent.

optimal for planning. Alternatively, there may be some benefit to allocating probability to states with static values. Sampling states with high value-error too frequently could slow down the agent's ability to learn the correct values by propagating back existing values from the previous reward regime.

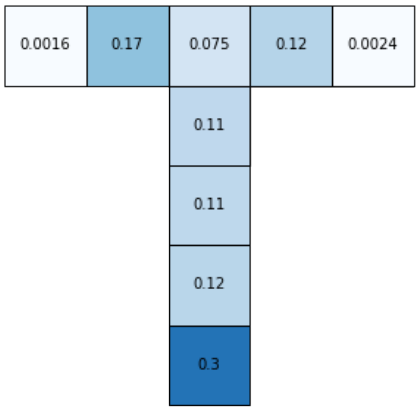
The agents' sensitivity to the number of planning steps is shown in Figure 5.9. The results shown in this figure further emphasize the detrimental effect of search control distributions which use terminal transitions. We observe that Uniform's performance decreases as the amount of planning increases. As more planning occurs, terminal-adjacent states will be sampled more often under the Uniform distribution, leading to updates which may include erroneous rewards. Ultimately, this drastically reduces Uniform's total reward as planning increases. We also note the surprising performance of Uniform when one planning step is used. It's unclear why Uniform performs comparatively well at this low level of planning and we are unsure why this result occurs. Comparatively, Avoid Terminal benefits from an increase in planning up to the level of 10 planning steps, after which it suffers from increased variance. MGSC remains robust to all levels of planning. Its total reward appears to increase slightly with the number of planning steps, but not drastically. Generally, we expect model-based agents to benefit from an increased planning budget so long as they may avoid erroneous portions of the model. While MGSC is able to achieve relatively high total reward at all levels of planning, it does not seem to fully take advantage of an incrased planning budget. It is possible there is a superior distribution — with greater returns to more planning — that MGSC has not been able to learn. Future work could examine if MGSC's meta-loss is reaching a local optimum and whether adjustments, such as an entropy regularizer or additional expectation updates, could help the agent learn a better distribution as the planning budget increases.



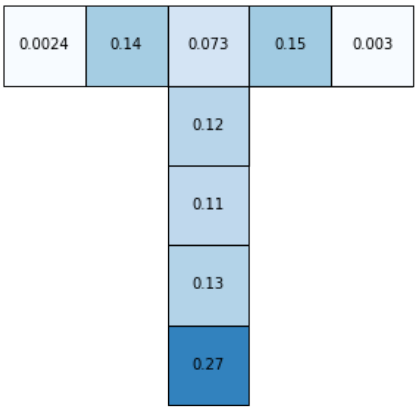
(a) 25% of Training



(b) 50% of Training



(c) 75% of Training



(d) 100% of Training

Figure 5.8: Plots of the distribution learned by MGSC at different points during training in the learned model setting.

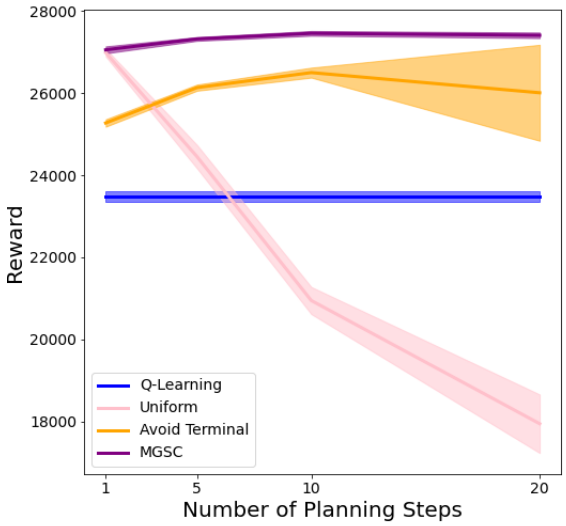


Figure 5.9: The total reward accumulated by each agent as the number of planning updates per real update is varied in the learned model setting. Note that Q-Learning's performance is not affected by planning. It is included here for reference.

Chapter 6

Future Work and Conclusion

We now discuss some avenues for future work and provide a summary of the work presented in this document.

6.1 Future Work

While we have presented two principles for useful search control in this thesis, future work can investigate the shape of the distributions learned by MGSC. Studying learned distributions might teach us about additional principles for useful search control.

Further, the selection of an action during planning steps is arguably part of the search control problem. Just as some states may be more beneficial to learning than others, some actions may also result in faster learning. While MGSC uses the current policy to select an action during planning, incorporating action selection into the learned search control distribution may provide further gains to planning-efficiency.

We now focus on two particularly important directions for future work in search control and in building upon MGSC: scaling the algorithm and the results of applying MGSC with a perfect model.

While we believe MGSC is amenable to scaling, there are details of the algorithm which must be adjusted to deploy it in settings with more states and actions, or with high-dimensional observations. In Chapter 4, we discussed how the use of an expectation update requires samples from the entire state-action space. As the number of states and actions grows this update becomes

prohibitively expensive. One possibility is to replace it with a sample-based approximation. This method would need to be tested empirically to validate that relatively a small number of samples can produce an approximation which provides a useful signal to the meta-objective.

Similarly, when the state-action space is continuous, or very large, it becomes impossible to represent each state or action with a logit. Future work along this line should consider new methods of encoding the search control distribution which are compatible with such settings. A variety of approaches are available which provide sample access to a learned distribution including AutoEncoders, Generative Adversarial Networks, and Normalizing Flows (Goodfellow et al., 2016; Goodfellow et al., 2014; Papamakarios et al., 2021). These methods can learn a compact, latent representation of a distribution over the state space. MGSC's meta parameters could then be used to train this latent representation – making it tractable to sample states and assign probabilities in much more complex environments.

While this thesis has concentrated on integrating MGSC into the Dyna framework, MGSC could be incorporated into other model-based architectures which have demonstrated strong abilities in complex environments. The Dreamer family of algorithms, which learn a world model of the domain, could be adapted to include search control using MGSC (Hafner et al., 2020; Hafner et al., 2021; Hafner et al., 2023). Future work could compare the performance of Dreamer algorithms as a baseline against versions augmented with learned search control.

Future work can also address the lacklustre results we observed in the perfect model setting. When the agent was given a perfect model of the TMaze, MGSC showed similar performance to the Uniform baseline in most settings of the environment. Understanding why MGSC struggles to learn a superior search control distribution, and how to improve its performance is an important component of ensuring that MGSC is robust to a variety of different models.

One possible explanation for this behaviour is that the current meta-loss does not provide a strong enough signal to push the logits away from the

uniform initialization. If this were the case, a different construction for the target parameter vector could be considered. Applying planning updates to the target parameters (similar to the method originally used by Flennerhag et al. (2022)) could result in a greater difference between the parameter vectors in the meta-loss.

Another possibility is applying a regularization term to the meta-loss. If the search control distribution is remaining largely static during training, we could keep a checkpoint of the logits from some numbers of timesteps in the past. By applying a penalty to the loss based on the KL-divergence between the checkpoint logits and the current logits, MGSC could be incentivized to adjust its search control distribution more rapidly.

6.2 Conclusion

In this thesis, we investigated the effects of search control on model-based RL agents. We formulated the problem of search control as finding a probability distribution over states and considered how this distribution can improve planning-efficiency with an eye towards ameliorating the use of model experience in MBRL. By comparing model-free agents and model-based agents equipped with different search control distributions, we identified two principles that can result in useful search control. First, the agent should focus on high value-error states. Second, the agent should avoid incorrect states.

We then introduced an algorithm, Meta-Gradient Search Control, capable of learning search control distributions. The derivation of MGSC's objective function was discussed in detail, providing clarity on why we believe it can learn useful distributions. We detailed a pseudo-code implementation of MGSC within the Dyna framework of MBRL agents.

We finally demonstrated the performance of MGSC in the TMaze domain. Under a fixed imperfect model, and a generative reward model, MGSC was shown to learn distributions which exceed the performance of a naive baseline and compare favourably with hand-coded distributions that use privileged, domain-specific knowledge. These experiments demonstrate the utility

of MGSC for search control in the TMaze and suggest it is a promising step towards learned search control in domains where a custom distribution, created a priori, is not feasible.

RL and model-based RL have demonstrated remarkable achievements in recent years. As these algorithms continue to be iterated on and explored, sample-efficiency remains a key metric for developing agents which can learn quickly in exacting tasks. This work is animated by our belief that improvements to search control can improve planning-efficiency, and that hopefully, such improvements can result in greater sample-efficiency. We hope that providing principles for useful search control and the MGSC algorithm has made a small measure of progress towards this goal.

References

Abbas, Z., Sokota, S., Talvitie, E., & White, M. (2020). Selective dyna-style planning under limited model capacity. *International Conference on Machine Learning*.

Andre, D., Friedman, N., & Parr, R. (1997). Generalized prioritized sweeping. *Neural Information Processing Systems*.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., & Whiteson, S. (2023). A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural Computation*, *12*(8), 1889–1900.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, *602*(7897), 414–419.

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. *International Conference on Machine Learning*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., & Abbeel, P. (2016). RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*.

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

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Neural Information Processing Systems*.

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations*.

Hafner, D., Lillicrap, T., Norouzi, M., & Ba, J. (2021). Mastering Atari with discrete world models. *International Conference on Learning Representations*.

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Holland, G. Z., Talvitie, E. J., & Bowling, M. (2019). The effect of planning shape on dyna-style planning in high-dimensional state spaces. *arXiv preprint arXiv:1806.01825*.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., et al. (2023). Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, *618*(7964), 257–263.

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

Pan, Y., Mei, J., & Farahmand, A. (2020). Frequency-based search-control in dyna. *International Conference on Learning Representations*.

Pan, Y., Yao, H., Farahmand, A.-m., & White, M. (2019). Hill climbing on value estimates for search-control in dyna. *International Joint Conference on Artificial Intelligence*.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, *22*(1), 2617–2680.

Saleh, E., Martin, J. D., Koop, A., Pourzarabi, A., & Bowling, M. (2022). Should models be accurate? *arXiv preprint arXiv:2205.10736*.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *International Conference on Learning Representations*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, *588*(7839).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–503.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359.

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

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

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

Welch, B. L. (1947). The generalization of 'Student's' problem when several different population variances are involved. *Biometrika*, *34*(1-2), 28–35.

Appendix A

Hyperparameter Settings and Selection

To select hyperparameters, we perform a grid search over all possible hyperparameter configurations from Table A.1. Each configuration is run with 30 random seeds during the selection process. We average results from all 30 random seeds and report the results of the best hyperparameters for each algorithm in consideration. For the baselines agents Q-Learning, Uniform, Horizontal Focus, and Avoid Terminal, only the step size parameter was swept over. For the MGSC agent both step size and meta-step size were swept over.

Note that the number of planning steps was varied during the experiments. However, we did treat not the number of planning steps as a hyperparameter during our sweep and we did not compare agents across different numbers of planning steps. For example, we report results comparing Q-Learning, Uniform, Horizontal Focus, and MGSC when each algorithm is set to use five planning steps. We do not make any comparisons where one algorithm is allocated five planning steps, and another algorithm is allocated 10.

Hyperparameter	Values
Step Size	1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1, 1e0
Meta-Step Size	5e-5, 5e-4, 5e-3, 5e-2, 5e-1
ϵ_{policy}	1e-1

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

Appendix B

Additional Results

We include additional experimental results not presented in the main text of this document. In particular, the main body of this work presents results for only some settings of the number of planning steps given to the model-based agents. We present figures which show results for all of these settings, including reproducing the figures which appeared in the main body.

B.1 The Perfect Model Setting

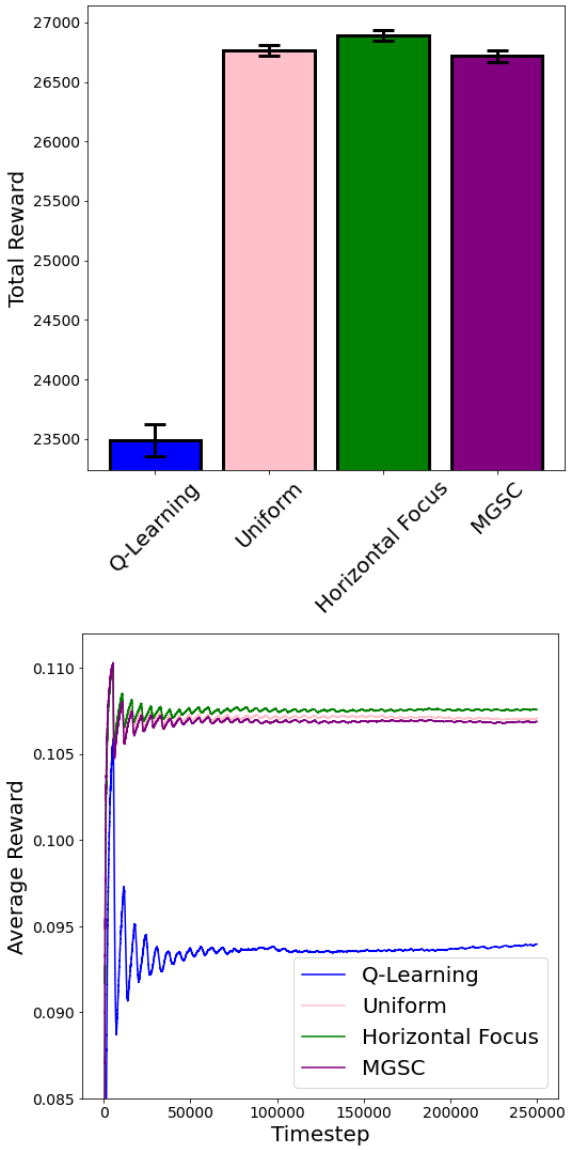


Figure B.1: Total reward and average reward for agents with one step of planning per real update in the perfect model setting.

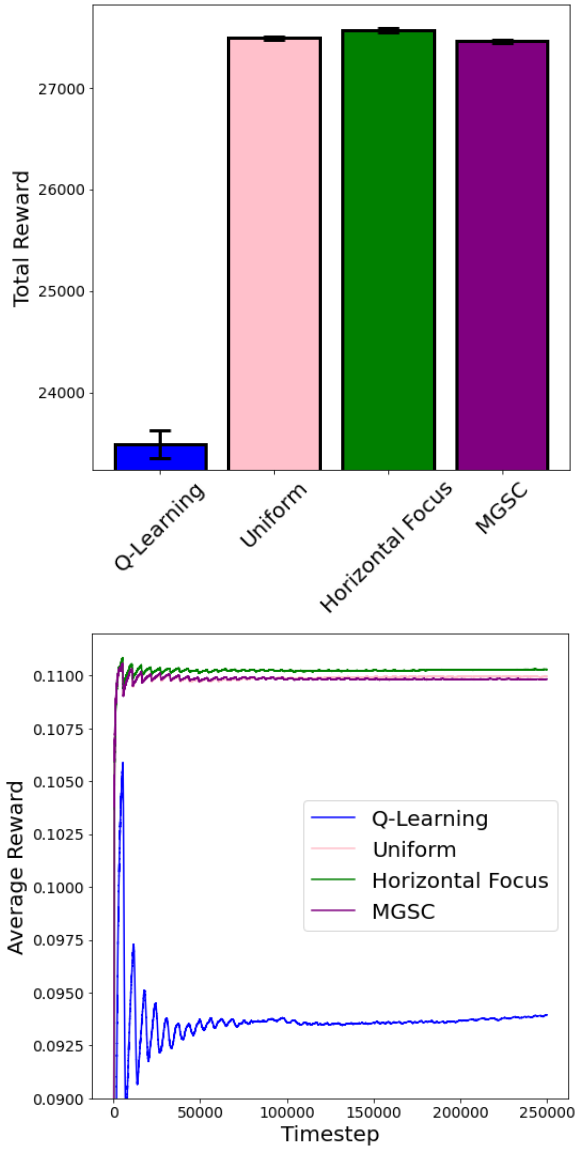


Figure B.2: Total reward and average reward for agents with five steps of planning per real update in the perfect model setting.

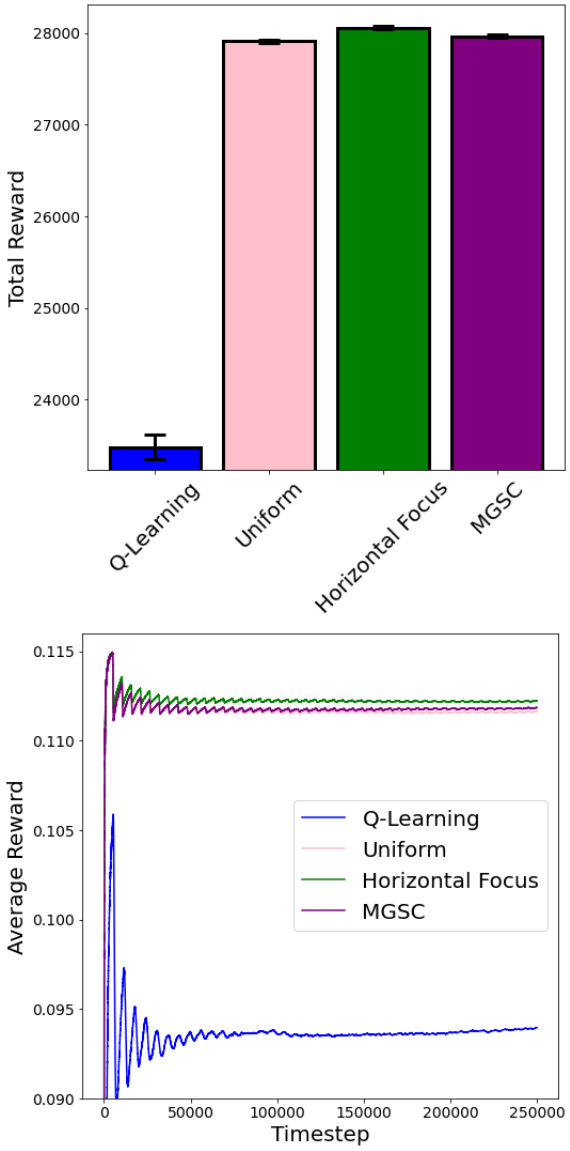


Figure B.3: Total reward and average reward for agents with ten steps of planning per real update in the perfect model setting.

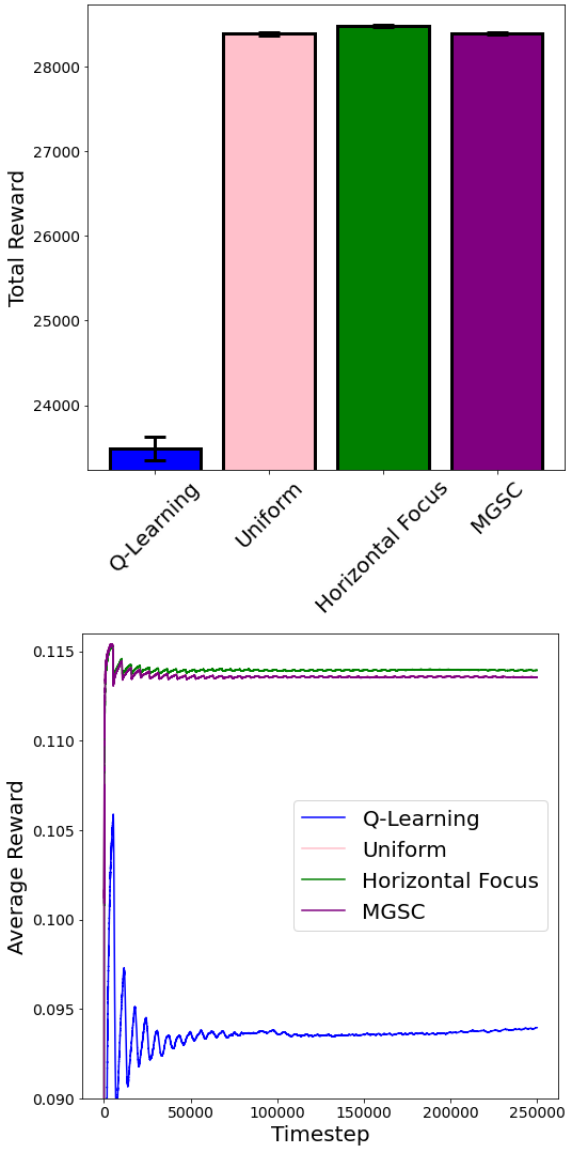
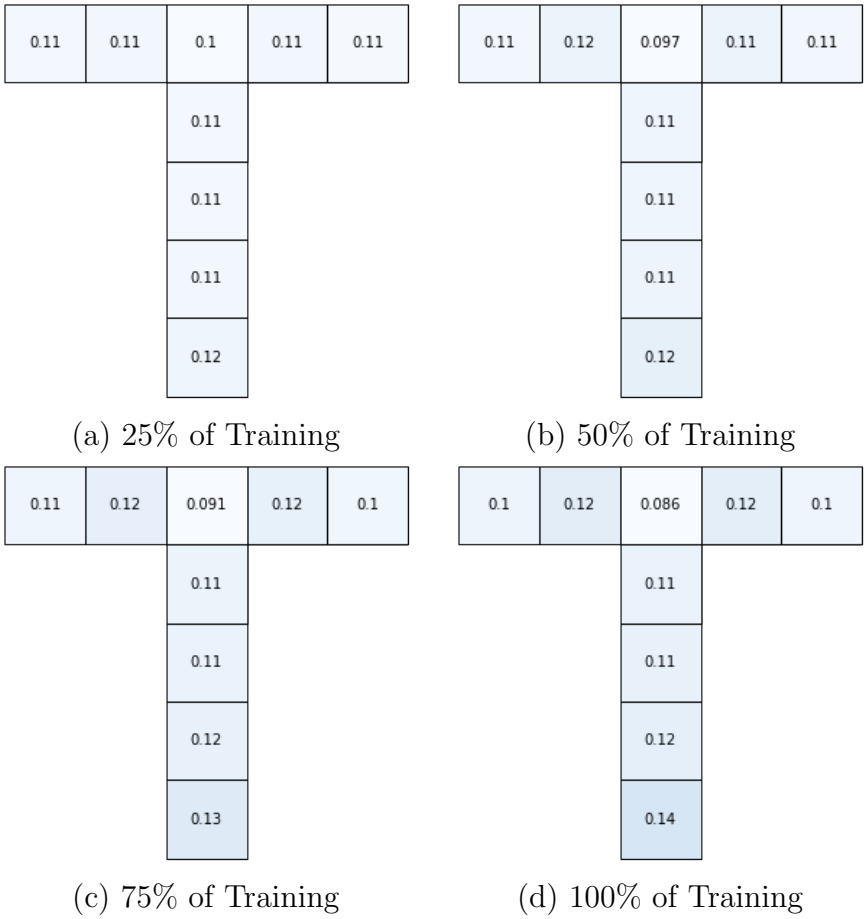


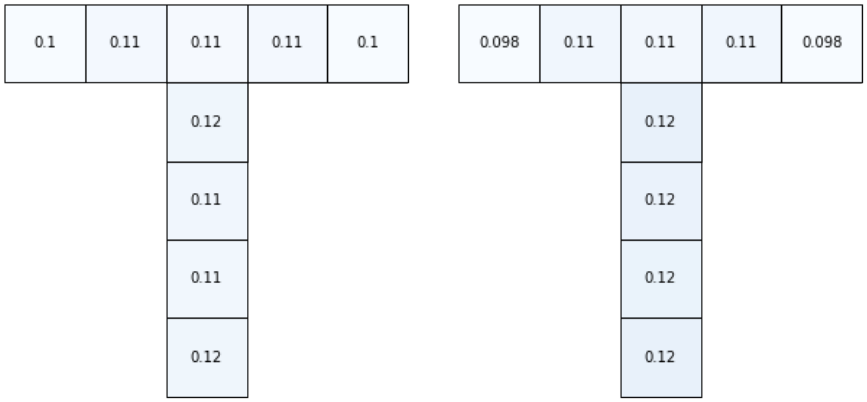
Figure B.4: Total reward and average reward for agents with twenty steps of planning per real update in the perfect model setting.



(a) 25% of Training (b) 50% of Training

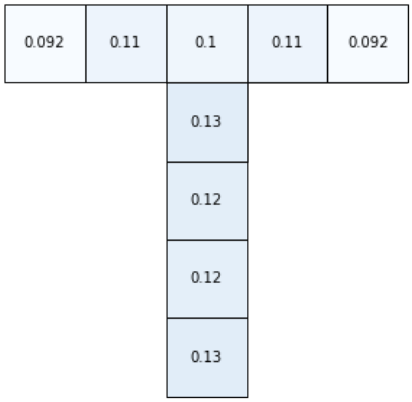
(c) 75% of Training (d) 100% of Training

Figure B.5: Plots of the distribution learned by MGSC at different points during training in the perfect model setting with one step of planning.

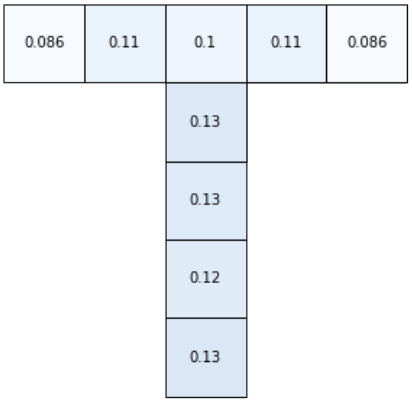


(a) 25% of Training

(b) 50% of Training

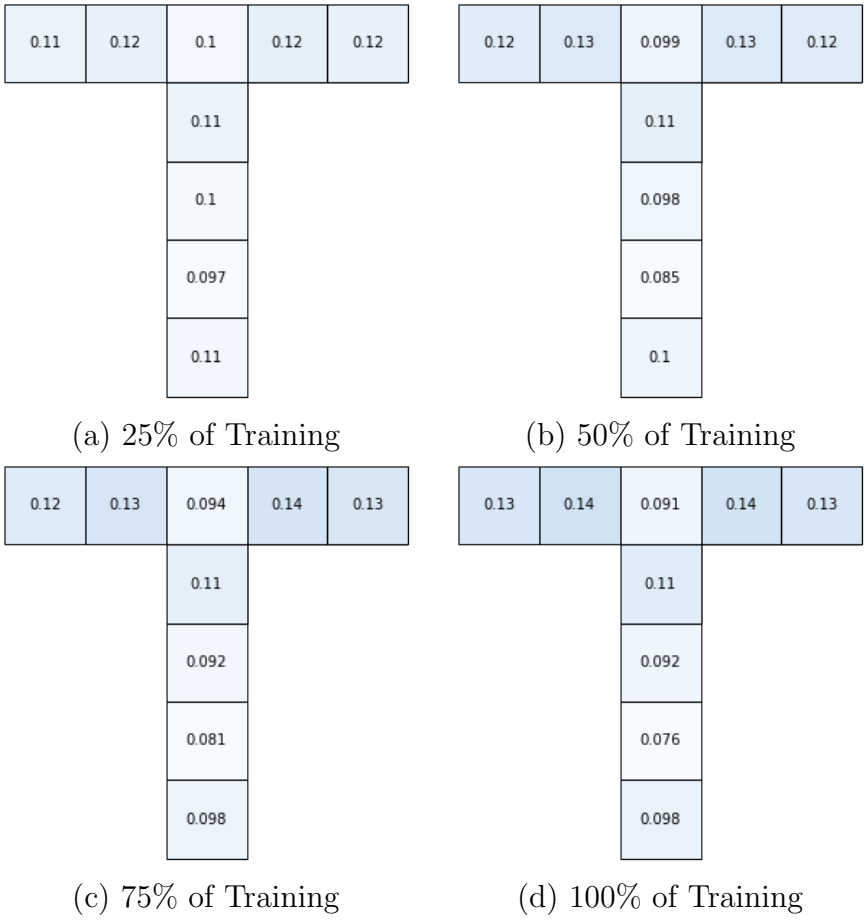


(c) 75% of Training



(d) 100% of Training

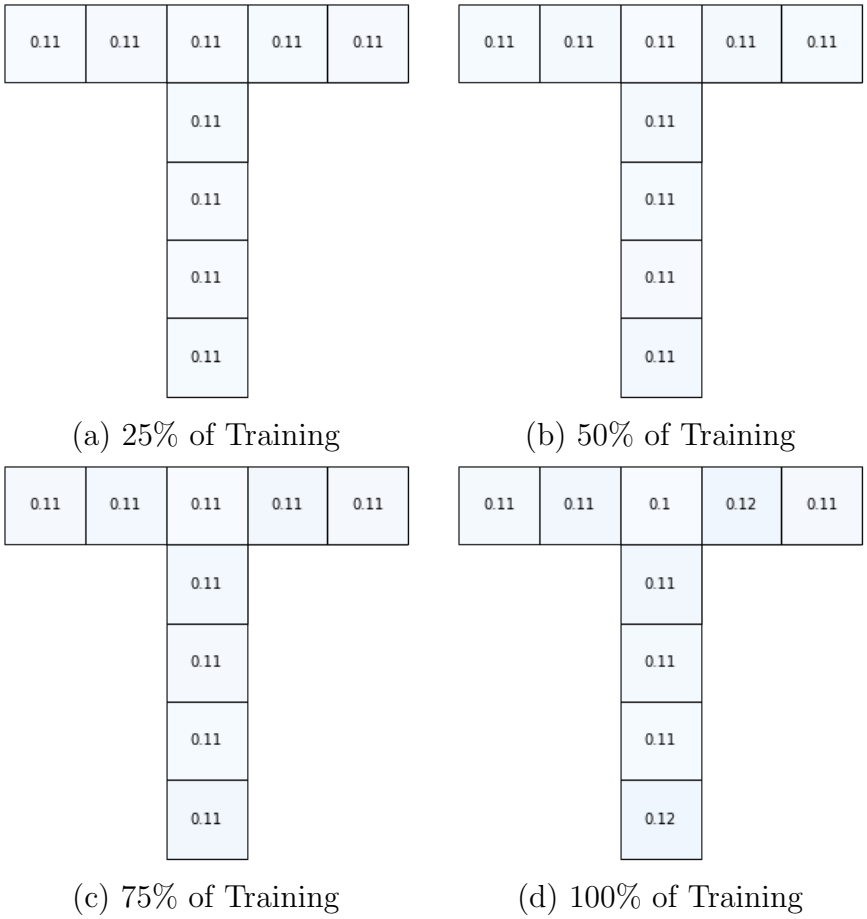
Figure B.6: Plots of the distribution learned by MGSC at different points during training in the perfect model setting with five steps of planning.



(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.7: Plots of the distribution learned by MGSC at different points during training in the perfect model setting with ten steps of planning.



(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.8: Plots of the distribution learned by MGSC at different points during training in the perfect model setting with twenty steps of planning.

B.2 The Imperfect Model Setting

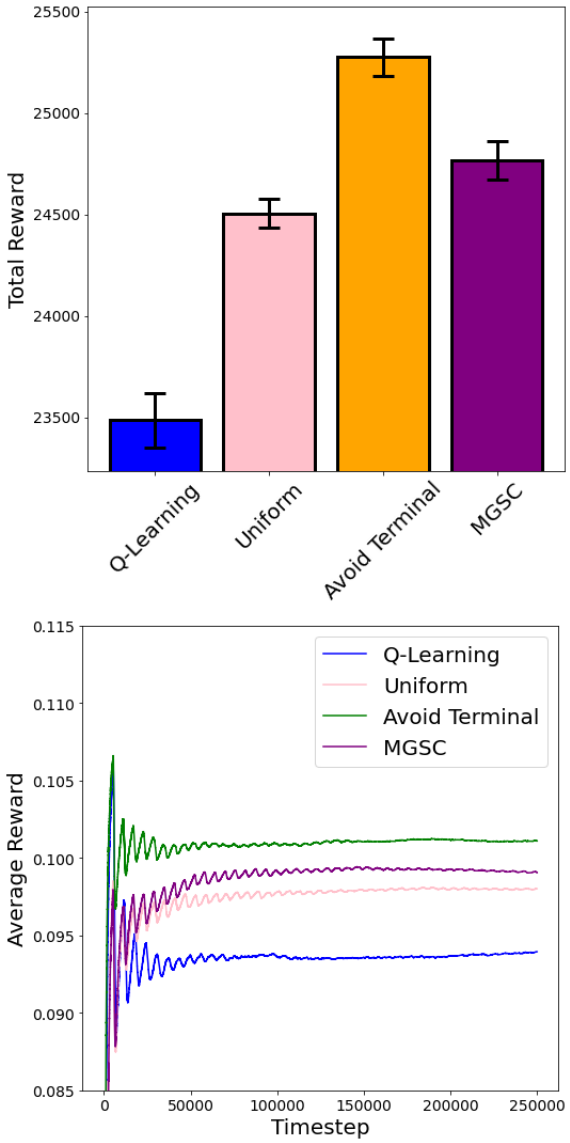


Figure B.9: Total reward and average reward for agents with one step of planning per real update in the imperfect model setting.

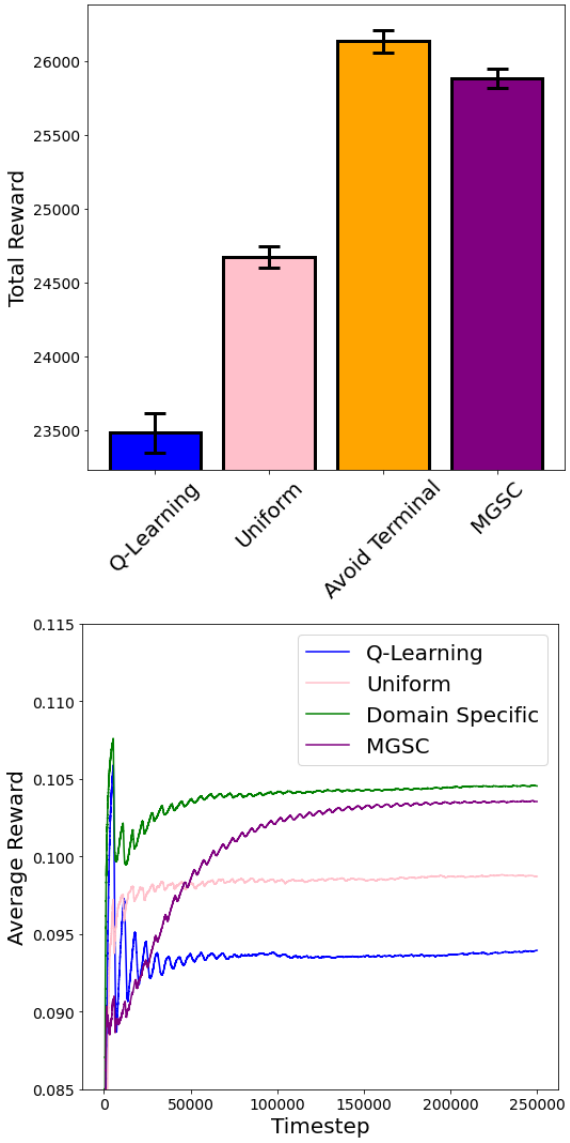


Figure B.10: Total reward and average reward for agents with five steps of planning per real update in the imperfect model setting.

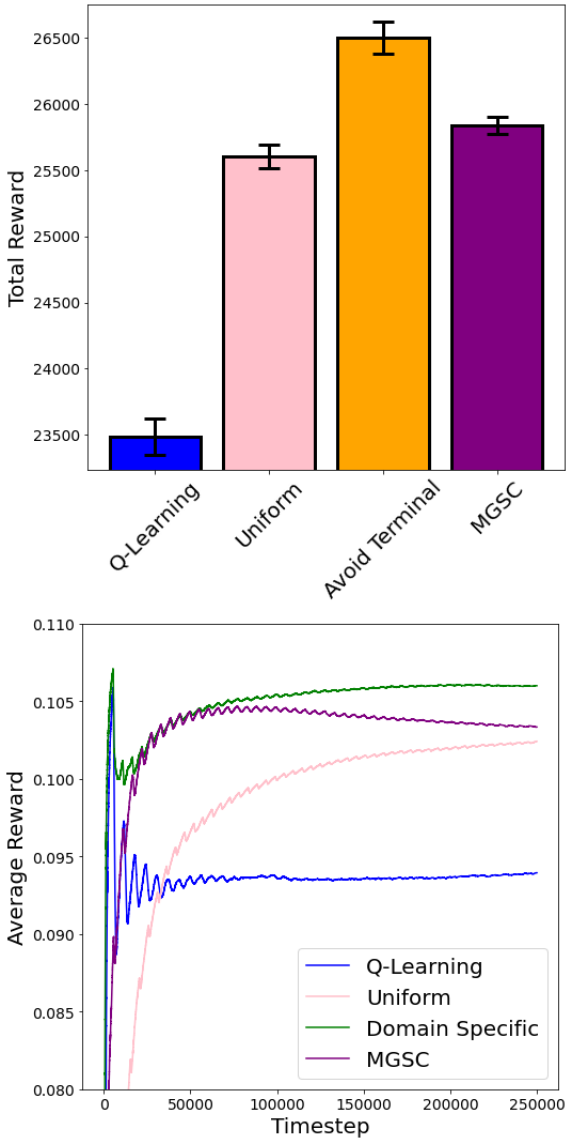


Figure B.11: Total reward and average reward for agents with ten steps of planning per real update in the imperfect model setting.

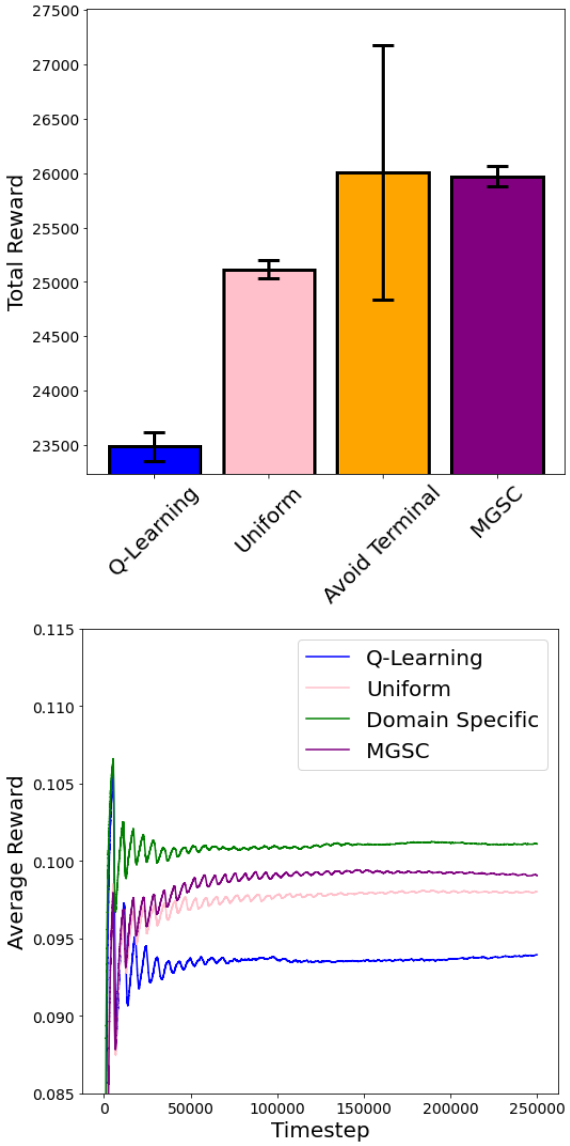
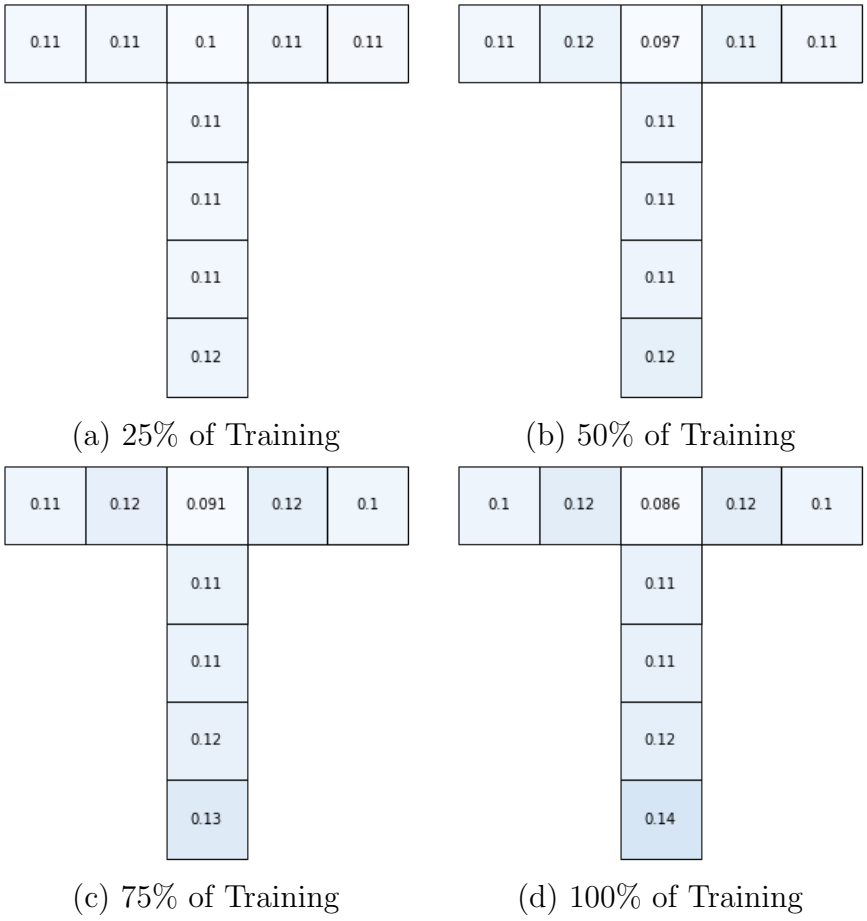


Figure B.12: Total reward and average reward for agents with twenty steps of planning per real update in the imperfect model setting.



(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.13: Plots of the distribution learned by MGSC at different points during training in the imperfect model setting with one step of planning.

(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.14: Plots of the distribution learned by MGSC at different points during training in the imperfect model setting with five steps of planning.

(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.15: Plots of the distribution learned by MGSC at different points during training in the imperfect model setting with ten steps of planning.

(a) 25% of Training (b) 50% of Training

(c) 75% of Training (d) 100% of Training

Figure B.16: Plots of the distribution learned by MGSC at different points during training in the imperfect model setting with twenty steps of planning.

B.3 The Learned Model Setting

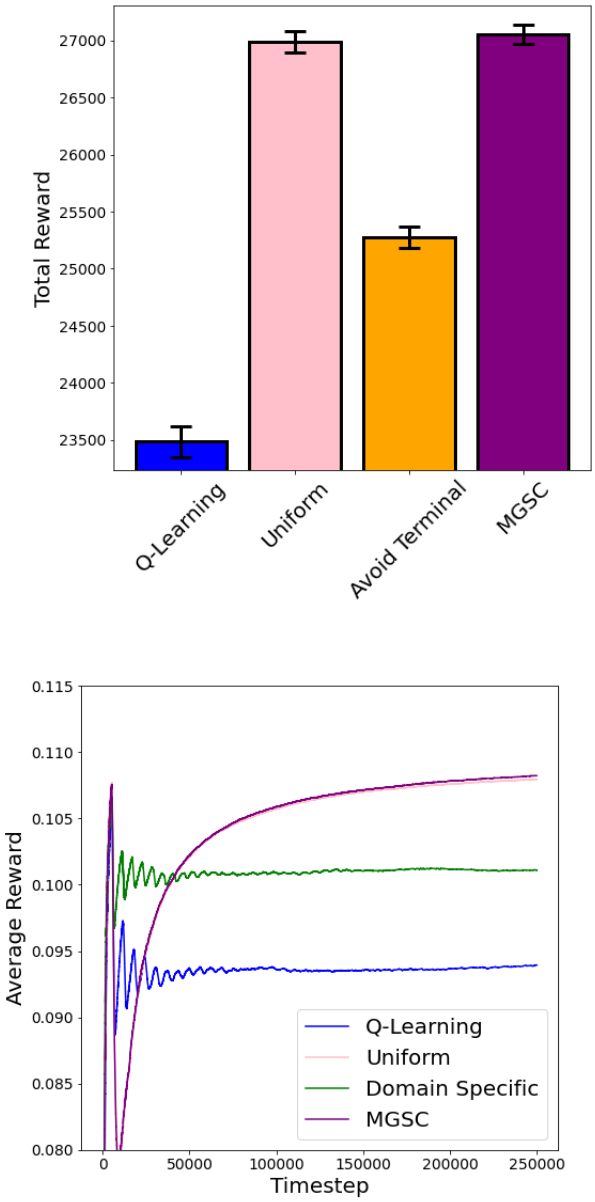
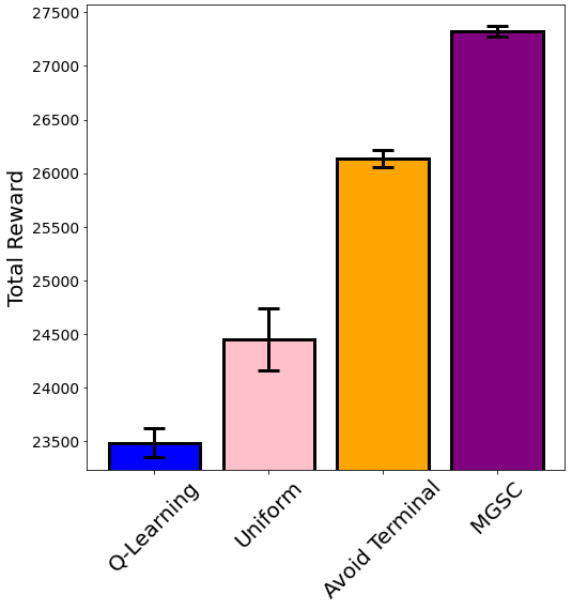


Figure B.17: Total reward and average reward for agents with one step of planning per real update in the learned model setting.



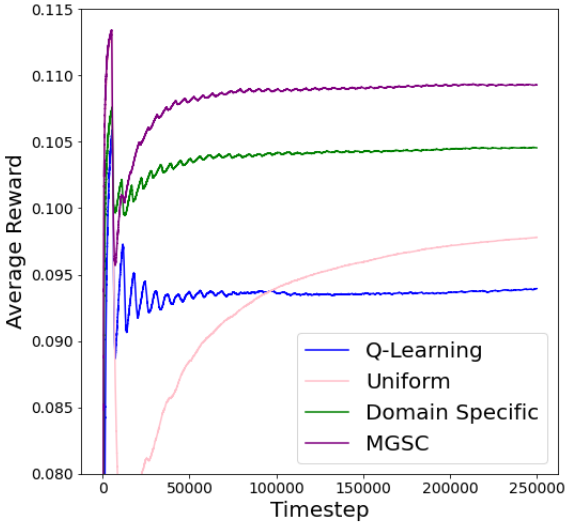


Figure B.18: Total reward and average reward for agents with five steps of planning per real update in the learned model setting.

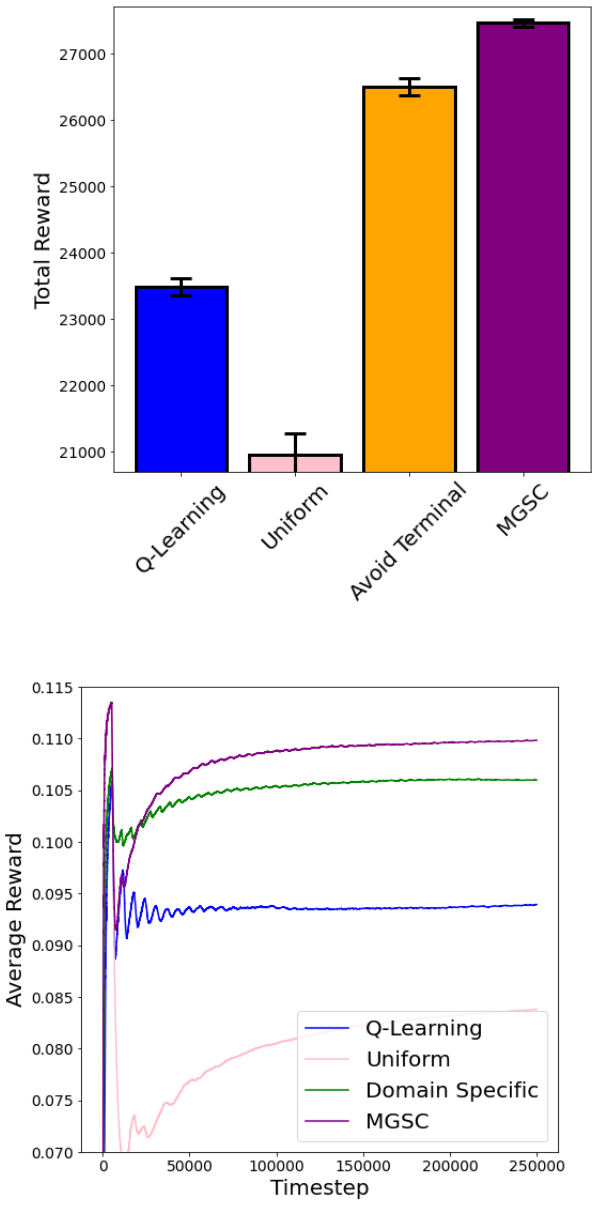


Figure B.19: Total reward and average reward for agents with ten steps of planning per real update in the learned model setting.

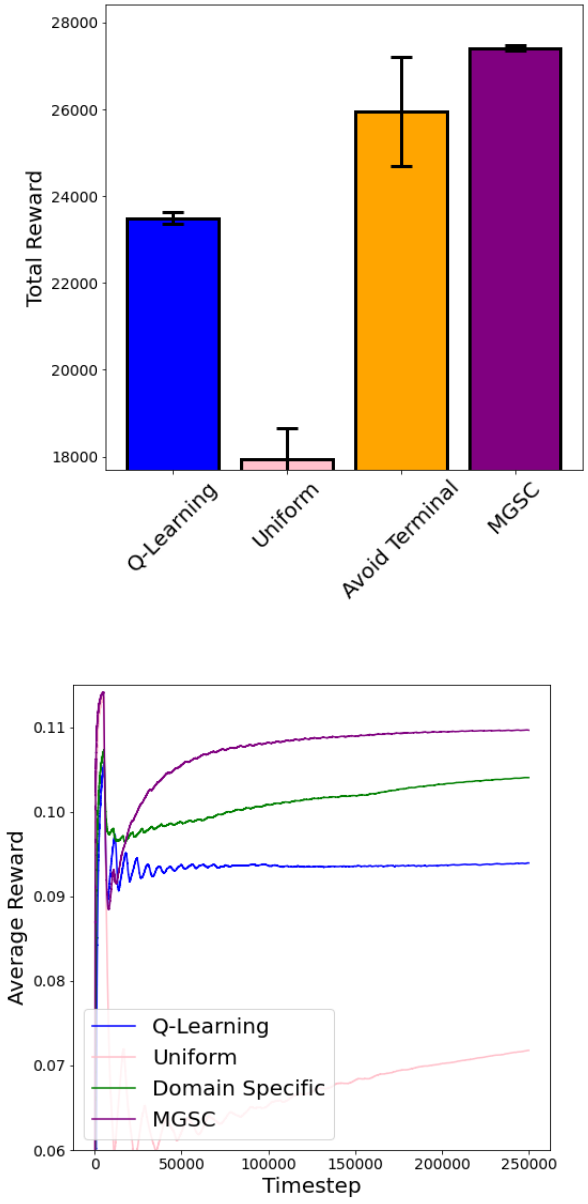
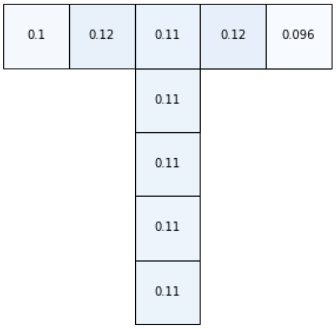
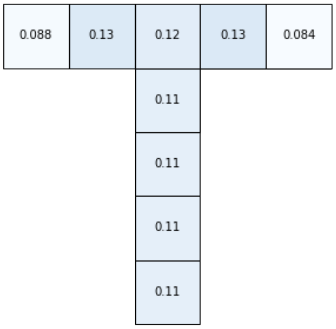


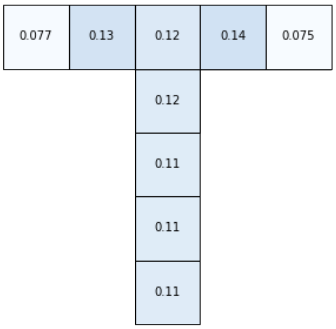
Figure B.20: Total reward and average reward for agents with twenty steps of planning per real update in the learned model setting.



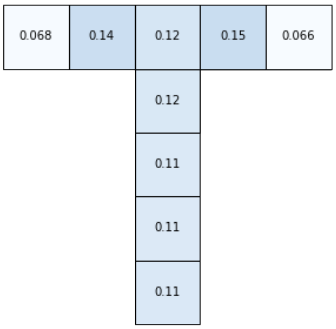
(a) 25% of Training



(b) 50% of Training

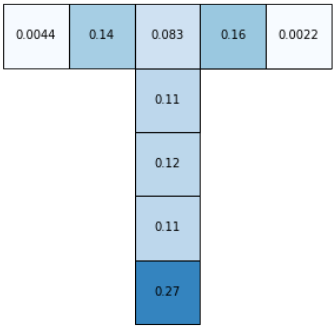


(c) 75% of Training

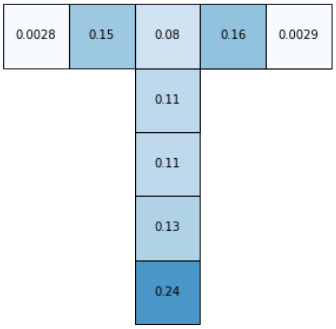


(d) 100% of Training

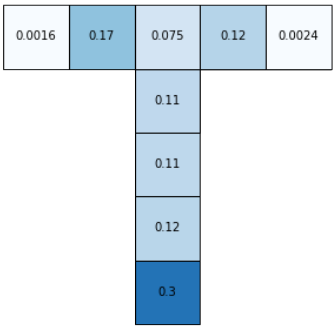
Figure B.21: Plots of the distribution learned by MGSC at different points during training in the learned model setting with one step of planning.



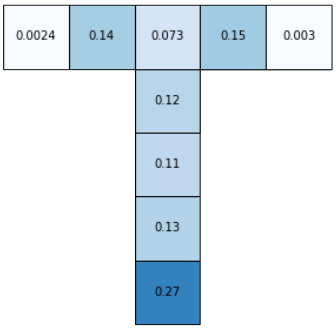
(a) 25% of Training



(b) 50% of Training

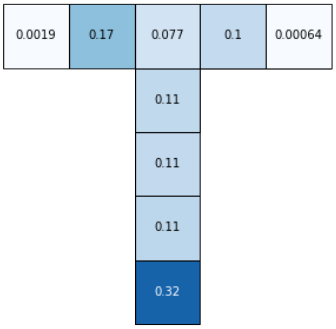


(c) 75% of Training

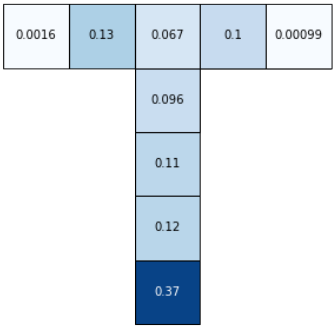


(d) 100% of Training

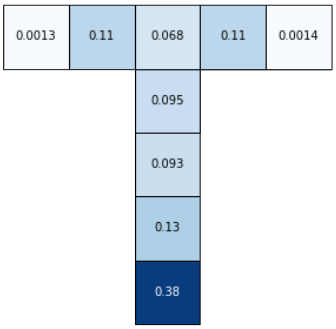
Figure B.22: Plots of the distribution learned by MGSC at different points during training in the learned model setting with five steps of planning.



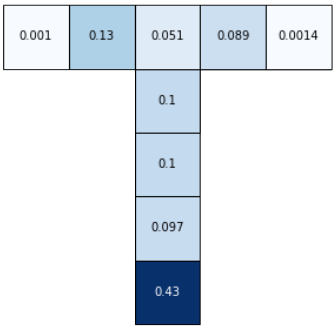
(a) 25% of Training



(b) 50% of Training

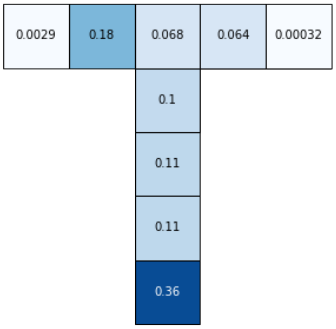


(c) 75% of Training

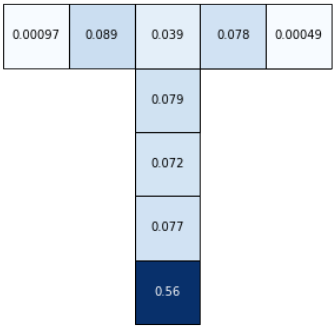


(d) 100% of Training

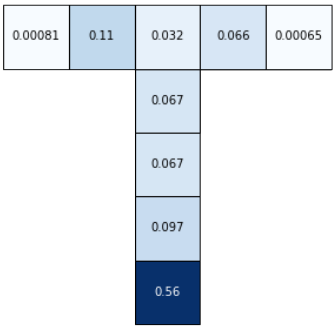
Figure B.23: Plots of the distribution learned by MGSC at different points during training in the learned model setting with ten steps of planning.



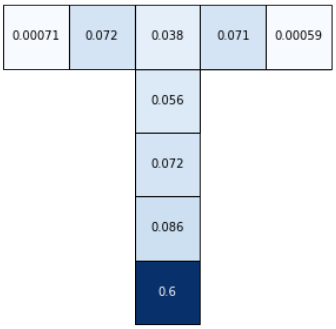
(a) 25% of Training



(b) 50% of Training



(c) 75% of Training



(d) 100% of Training

Figure B.24: Plots of the distribution learned by MGSC at different points during training in the learned model setting with twenty steps of planning.