Proactive Planning through Active Policy Inference in Stochastic Environments

by

Nolan Poulin

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

by

May 2018

Committee in charge:

Professor Jie Fu, Thesis Advisor, Chair
Professor Zhi Li, Committee Member
Professor Carlo Pinciroli, Committee Member

The dissertation of Nolan Poulin, titled Proactive Planning through Active Policy Inference in Stochastic Environments, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

Worcester Polytechnic Institute

Abstract

In multi-agent Markov Decision Processes, a controllable agent must perform optimal planning in a dynamic and uncertain environment that includes another unknown and uncontrollable agent. Given a task specification for the controllable agent, its ability to complete the task can be impeded by an inaccurate model of the intent and behaviors of other agents. In this work, we introduce an active policy inference algorithm that allows a controllable agent to infer a policy of the environmental agent through interaction. Active policy inference is data-efficient and is particularly useful when data are time-consuming or costly to obtain. The controllable agent synthesizes an exploration-exploitation policy that incorporates the knowledge learned about the environment's behavior. Whenever possible, the agent also tries to elicit behavior from the other agent to improve the accuracy of the environmental model. This is done by mapping the uncertainty in the environmental model to a bonus reward, which helps elicit the most informative exploration, and allows the controllable agent to return to its main task as fast as possible. Experiments demonstrate the improved sample efficiency of active learning and the convergence of the policy for the controllable agents.

Acknowledgments

Thank you so much to my thesis advisor, Professor Jie Fu, who guided me through the thesis process. I am grateful for my improved abilities to perform research and develop algorithms. Thank you so much to my committee members, for providing great feed-back and taking the time to review my work. Also, thank you to the students of CIRL who have always provided

I'm forever grateful to my parents, Deb and Steve Poulin, who have given me so much support and always reminded me that they're proud of me for being who I am.

Finally, thank you to McKayla. Your support and teamwork during my graduate studies have been unparalleled.

Contents

List of Figures

List of Tables

Chapter 1

Motivation for Active Policy Inference

A fundamental assumption of this thesis is that when an autonomous agent moves in an environment it is attempting to fulfill a specified task. The agent could be an autonomous vehicle or a robotic manipulator and in a realistic environment, there will be static obstacles and other autonomous and uncontrollable agents. Tasks for autonomous agents are often multifaceted. Two examples are:

- "*Drive from point-A to point-B and neither leave the road nor collide with another car or pedestrian.*"
- "*Pick up everything on the table and put it in the box that the human is holding.*"

It is clear that accurately representing the probabilistic motion distribution of uncontrollable agents – cars, pedestrians, or people holding boxes – will benefit the autonomous agent as it tries to complete its main objective. This thesis details an algorithm that elicits informative interactions with an uncontrollable agent that a controllable agent uses to effectively plan its future actions.

1.1 Introduction

Modeling stochastic environments with discrete state and action spaces has traditionally been accomplished with a Markov Decision Process (MDP). An agent that takes planned actions in an MDP is following its *policy*. When an autonomous agent is given a task to complete in its environment, the agent is rewarded upon completing this task. In a realistic scenario, there are often too many unknowns to solve for a control policy that has the highest likelihood of returning a reward. The agent may not know how the actions it can take affect its transitions from one state to another and, therefore, can't immediately make an efficient plan. Or, the agent may know what actions to take to earn the reward, but environmental disturbances require that the agent adapt. In the latter case, the agent needs to learn a model of its environment.

In this work, we'll assume that a stochastic environment has been successfully modeled, and that, given an arbitrary task, our controllable agent can solve for an optimal policy that maximizes the likelihood of reward. However, when we introduce a second, uncontrollable, agent into the environment the controllable agent needs learn how the uncontrollable agent will affect it's plan. We build upon *policy gradient* methods to infer the policy of the uncontrollable agent, and subsequently converge to the optimal policy for the controllable agent over the course of many interactions. This implementation parameterizes the policy with a multivariate-normal distribution. We use the learned policy-parameter variance to quantify the uncertainty in the inferred parameters, and use that to guide exploration.

In Sections 1.2-1.3 we'll cover the how current models and algorithms built the foundation for this policy inference algorithm. Section 2.1 will cover common notation in Reinforcement Learning (RL) and MDP to serve as preliminary information

to the main result in Chapter 2. Here, we'll test the inference algorithm by inferring the policy of a single agent. The following chapter, 3, will extend the model and experiments to two agents. Finally, Chapter 4 will introduce our algorithm for active policy inference and demonstrate results with several simulation experiments.

Consider the following: Each day, a mobile robot repeatedly traverses a factory floor to deliver a parcel. Eventually, a second robot also starts working in the area, and the first robot does not know anything about the second robot's task. The first robot must not collide with the second robot and, without knowledge of the second robot's intent, it now takes longer to deliver its parcel each day. After each day, the first robot can use it's observations of the inter-robot interactions to improve its delivery plan. Is there a better solution than greedily updating the plan? Can the first robot plan to move in such a way that will expose more about the second robot?

1.2 Inferring Latent Variables in MDPs

Given an unknown system that is modeled as an MDP, adaptive planning [HL12] means that the agent must learn or infer latent (hidden) environmental parameters and replan when it has a new estimate of the unknown system. The main techniques to accomplish this fall into two distinct categories, model-*based* RL, and model-*free* RL; the survey in [PN17] provides a succinct comparison. Generally, model-based RL attemts to learn the underlying transition model. By learning an approximately correct environmental model, an agent can optimally solve problems [FT14][BWH12][DR11]. Model-free RL does not learn an environmental model, but learns the actions, or control inputs, to take at given states by performing gradient ascent on the likelihood of a reward function [Wil92][PS08]. In general, the

RL community is now focusing on methods that combine model-free with model-based approaches called actor-critic methods [KT00]. Actor-critic methods update an estimated environmental model to help reduce the variance in the policy-gradient common in model-free implementations [PS08].

1.2.1 Model Based Solutions

For the described multi-agent factory scenario, suppose that there are a set of parameters that can characterize the policy of the second, unknown and uncontrollable, robot. To capture these unknown parameters, we'll extend the model from an MDP to a Hidden Parameter Markov Decision Process (HiP-MDP) [DVK16]. HiP-MDPs are well suited to problems where the number of parameters to learn is small relative to the size of the state-action space. This is a very applicable framework when certain aspects of the environment are already known, and sensory information is precise and accurate.

Arguably, simply exploiting information from a multi-agent interaction only requires a controllable agent to capture the *intent* of an uncontrollable agent. Recently, the Mixed-Observability MDP (MOMDP)has been presented in [BWF+13] as a an instantiation of a Partially-Observable MDP (POMDP)[KLC98]. In this MOMDP, an autonomous golf-cart maintains a belief over a pedestrian's intent when they interact. The authors state that "*the [golf-cart]'s ultimate goal is to complete the specified task and not to recognize intention.*" In other words, learning the policy of an uncontrollable agent provides excessive information to handle a vehicle-pedestrian encounter in a crosswalk.

In comparison to HiP-MDP and MOMDP, more general classes of RL algorithms assume that the entire transition model is unknown. To prove that a transition model has been learned, [FT14] puts guarantees on the samples required to

achieve a Probably Approximately Correct MDP (PAC-MDP). The authors show that the number of samples needed to learn a transition system in an MDP that has a bounded model error scales polynomially. The authors also present an Maximum Likelihood Estimate (MLE) of the transition model parameters, complete with empirical mean and variance, but this requires a parameter set that scales with the size of the state and action domains.

The RMAX algorithm from [BT02] is specifically geared to solve a multi-agent MDP. RMAX , however, would require that the controllable agent coerce visits to each *joint*-state[1]enough times to conclude, with a specified confidence, what the probability of future joint states is. This essentially models the action distribution of the other agent(s) as part of the underlying transition model.

Using another approach in the face of uncertain transitions, [BWH12] merge adaptive and robust methods for solving MDPs. Normally a robust solution uses the *minimax* approach, it plans for the worst-case transition. Instead of planning for the worst, an unmanned aerial vehicle (UAV) models its uncertain transitions with Dirichlet distributions. In an algorithm like an Extended Kalman Filter (EKF), transition samples form an empirical covariance matrix of the system's transition model so that the UAV can reduce its exposure to a failed mission. This allows for the system to have a failure *risk-tolerance* instead of using a conservative *minimax* plan.

The objective of [CLP17] is learn the true model of several adversaries by minimizing "*the cost of information gain*". Each adversarial agent is modeled with an MDP and a set of pre-designed temporal logic specifications. The goal is to identify which specification each adversary is trying to satisfy. The controllable agent receives a reward by maximizing the information gain, the entropy of the current

[1]A joint-state is the combination of the states of individual agents.

state belief.

Similarly, [LXM13] assume that an adversary in an MDP may have a non-stationary policy. If the adversary chooses to play "*nicely*", then a minimax policy would be far too conservative. Each time a state transition is observed, the authors record it and subject each new transition to a "*stochasticity check*", essentially trying to detect a change in the adversarial policy using the Chernoff bound. Both [LXM13] and [BWH12] have tried to overcome the conservativeness often encountered in *minimax* policies, but environmental agents can be agnostic, or even complimentary, to the controllable agent's intent.

In general, model-based solutions allow the agent to leverage its belief about future states as it plans. This belief is reflected in the *value* of each state, but is very sensitive to model errors.

In this work, we assume that the second agent's policy is unknown but can be parameterized. We'll use a HiP-MDP to learn the policy from observed state-sequences; we do not know what actions the agent has decided to take at each state. In single agent scenarios, [FT14], [BWH12], [PS08], and [TMZ$^+$14], the agent always knows what action it just took. This work shows that we can learn the hidden action-distribution of an uncontrollable agent, while only observing the outcomes of those actions.

1.2.2 Exploration in Policy Gradient Methods

If one agent needs to interact with another agent to learn that agent's policy, then some level of "exploration" will help it learn faster [NNXS17]. Instead of always taking an action on the path towards the highest reward, an agent can sometimes take *sub*-optimal actions that might provide more information about hidden parameters.

This work will implement policy-gradient methods to infer a policy from observed

state-sequences. Policy-gradients [Wil92] are traditionally applied in model-*free* RL as well as actor-critic RL. Although this work will use policy-gradients in model-based learning, it is worthwhile to describe how exploration is achieved by other policy-gradient implementations, and how it produces more informative samples (state-sequences).

The PILCO algorithm, from [DR11], claims exceptional sample efficiency as it learns a dynamics model of a physical cart-pole swing-up. It learns the optimal control inputs to the system via a policy-gradient, but incorporates the variance of policy parameters into the policy synthesis by modeling the unknown dynamics as Gaussian Processes (GPs).

Many of these algorithms have relied on the random-sampling of policy parameters to effectively add entropy to their learning process. For instance, [PS08] use a stochastic disturbance acceleration policy to control robot manipulator servos. Similarly, [SOR$^+$10] samples actions from a distribution, with a mean value determined by learned feature weightings.

If an agent is synthesizing a policy, regularizing a value function with an entropy term puts value on exploration [NNXS17]. Entropy regularization works in-practice but this is a heuristic that does not *guide* exploration in a formulaic approach. This work will infer the policy of an uncontrollable agent using a policy-gradient method, and directly use the second moment of the policy parameters to guide exploration.

1.2.3 Learning from Demonstration

The stated inference problem is similar to *imitation learning*, also referred to as *learning from demonstration*. These algorithms attempt to learn a policy from expert demonstrations, although the "*expert*" may be following a sub-optimal policy.

Given a set of state-action samples and a set of pre-defined features, [HLZP17]

discuss the number of samples required to learn an unknown transition model *and* policy from demonstration given a maximum model-error threshold. The authors can also estimate the log-loss function used by the expert.

In an imitation learning approach, the DAGGER algorithm learns to complete a task in a supervised fashion [RGB11]. Given an initial set of trajectories from an expert, the agent then attempts to mimic the policy. When an agent is trying to mimic an expert's policy, the agent's probability of making a "wrong" decision over the course of a trajectory grows polynomially with the length of the trajectory. DAGGER will minimize that probability of error by minimizing an upper bound to the 0-1 loss of a learned trajectory; the 0-1 loss would count the number of times the agent took a different action than the expert, but this is not always observable.

At any point during a trajectory, the agent has some probability of asking for the expert for a correction that will last a few time steps, creating a mixture of expert and agent decisions. Early in the learning, the agent is likely to have made several errors, so the expert correction is very informative. The correction will probably start from a state that the expert would not have visited on its own, and the chance of querying the expert for advice diminishes as the learning procedure continues.

By aggregating the observed data from previous learning stages, DAGGER produces a final policy that matches the average loss incurred during the training period. Since the first stages had a high proportion of expert trajectory-segments, this final policy is comparable to the expert's. As the number of stages goes to infinity, the loss of the final policy will converge to the loss of the *best* policy.

1.2.4 Active Learning

When an agent is trying to learn a model, e.g., a transition or value function, collecting data from areas of the largest model error is clearly the best way to

improve a model estimate. The true model error is rarely available to an agent, however. As a proxy, agents can minimize an upper bound to that error, or seek data that minimizes the uncertainty in the model itself. An agent that intentionally seeks data that will minimize that error or uncertainty is employing *active* learning in the RL task. In fact, DAGGER [RGB11] has been described as "*active learning from an oracle*" by [AWD17].

In comparison to DAGGER , [KVTT17] randomly samples policy parameters from normal distributions and actively updates the distribution's moments as a function of the rewards earned by recent trajectory samples from a system. A robot interacts with a human and is rewarded by an engagement metric. This implementation actively varies the distribution of policies that generate the trajectories to converge to the most rewarding policy, but the mechanism to control this active update is heuristically defined.

The key tenant of (most) active learning algorithms is determining what is unknown in a model and adjusting the policy such that some measure of the unknown is minimized. This helps active learning algorithms improve sample efficiency in RL.

Remark 1.2.1. *It should be noted that [FSM12] uses the term "active inference" to describe a policy synthesis algorithm for computing an optimal policy in a belief-MDP. This synthesis algorithm shows that minimizing* surprise*, the negated log-likelihood of an observed state transition, yields the same policy as maximizing the expected reward. This should not be confused with the implementation in this work.*

1.3 Contributions of this work

We apply model-based RL to a problem where one agent must learn the hidden policy of an uncontrollable agent through interaction. The policy of the uncontrollable

agent is fixed, it does not change with time, but might be dependent on the relative position of the the controllable agent. The environment is stochastic, actions lead to a distribution over future states. By learning the policy of the uncontrollable agent, the controllable agent can optimally plan to complete its task.

We contribute three major advances. First, we apply a policy-gradient method to infer the demonstrated policy of the uncontrollable agent. The inferred policy is parameterized as a multivariate distribution that allows this implementation to capture the second-moment of the policy-distribution. The second-moments quantify the ambiguity in the inferred policy. Second, we'll request the uncontrollable agent to sample its initial state based on the ambiguity in the policy-distribution. This provides guide for efficiently requesting policy demonstrations. Third, we'll show that a controllable agent can adaptively plan to proactively infer the policy of the uncontrollable agent. The controllable agent will receive a bonus reward when states with a high uncertainty are visited.

Unlike [LXM13] and [BWH12], we do not limit the case to adversaries; there is a chance that the uncontrollable agent will help earn rewards. Finally, we perform the inference with only state-sequences. The actions taken by the uncontrollable agent are not observable, only the action-outcome.

Chapter 2

Policy Inference with Gaussian Policy Estimate

The multi-agent system presented in this thesis is modeled with two agents, the controllable agent α_1 and the uncontrollable agent α_2. The interaction between the two agents is captured by a Markov Decision Process (MDP). We assume that α_2 has a predefined yet unknown policy. If α_1 is given a task, it can only robustly plan for this task when it has an accurate estimate of how α_2 will act. Determining the action distribution of an uncontrollable agent given its current state is known as *policy-inference*. The following definitions formally present a hidden policy in an MDP.

After defining the Hidden-Parameter MDP, this chapter presents an inference method that uses Monte-Carlo integration of policy parameters sampled from a multivariate distribution. We'll use Stochastic Gradient Ascent (SGA) to improve the log-likelihood of an observed set of data given a policy parameterization. Section 2.2.2 details a Q-function approximation which is used to form a softmax policy like in [NNXS17], also known as a Boltzmann [HLZP17] or a Gibbs [SO15] policy.

2.1 Hidden-parameter MDP

Definition 2.1.1. *The interaction between two agents is captured by a hidden-parameter MDP,*

$$\mathcal{M} = (S, A_1 \times A_2, R, T, \pi_2, I_0, \gamma)$$

with the tuple defined as in [SO15] plus the hidden parameter, π_2:

- $S \equiv (S_1 \times S_2)$ *is a set of joint states with cardinality* $0 < |S| < \infty$.
- $A_1 \times A_2$ *is a finite set of actions, where* A_1 *is the set agent one can execute and* A_2 *is the set available to agent two.*
- $R : S \times A_1 \to \mathbf{R}$ *is the real-valued state-action reward function given to the controllable agent.*
- $T : S \times (A_1 \times A_2) \to \mathsf{Dist}(S)$ *is the probabilistic state transition function* $T(s'|s, (a_1, a_2))$ *which yields the probability of reaching state* s′ *after both agents take action pair* (a_1, a_2) *at the state* s.
- $\pi_2 : S \to \mathsf{Dist}(A_2)$ *The distribution of agent two's actions given a state. The probability of each action is* $\pi_2(a_2|s)$.
- $I_0 \in \mathsf{Dist}(S)$ *is the initial state distribution.*
- $\gamma \in (0, 1]$*: The discounting factor.*

This definition also leads us to a couple pivotal assumptions in this work:

Assumption 2.1.2. *The state-action transition function* $T(\cdot)$ *is known.*

We'll also need two definitions from [GHL09] about policies to continue:

Definition 2.1.3. *Markov Policy: A policy $\pi_t(s)$ is considered to be a Markov policy if $P(a_j|h_t) = P(a_j|s^{(N)}), \forall a_j \in A$, where $h_t = s^{(0)}, ..., s^{(N)}$ is a state history ending at some discrete time $t \in [0, \infty)$.*

Definition 2.1.4. *Stationary Policy: A policy $\pi_t(s)$ is considered to be a stationary policy if $\pi_t(s_i) = \pi(s_i), \forall s_i \in S$, for all time $t \in [0, \infty)$.*

Thus, a stationary Markov policy is fixed with respect to time, and is only dependent on the current state of the system; it is independent of history.

Assumption 2.1.5. *The unknown policy of α_2, π_2 is a stationary Markov policy.*

Given that α_1 takes some action a_1, the distribution of next states is

$$P(s'|s, a_1) = \sum_{a_2 \in A_2} T\left(s'|s, (a_1, a_2)\right) \pi_2(a_2|s). \tag{2.1}$$

With this model, the true probability of a future state s' given the current state s is

$$p(s'|s) = \sum_{a_1, a_2 \in A} T\left(s'|s, (a_1, a_2)\right) \pi_2(a_2|s) \pi_1(a_1|s), \tag{2.2}$$

where, for simplicity, this report uses identical action sets, $A_1 \equiv A_2 \equiv A$. Finally, the MDP formed by the tuple $(S, A_1 \times A_2, R, T, \pi_2, I, \gamma)$ will be referred to as $\mathcal{M}$.

2.2 Unknown Policy Parameterization

2.2.1 Preliminaries

The parameterization for π_2 requires an understanding of how a reward function, R, influences the optimal action distribution at a state s. Although we do not wish to learn the reward function of α_2, we will build approximations of the *value* and

state-action value functions used by α_2. These are often a function of a reward function.

Value Functions

Following [HL12] and [SO15], the value of a state s is defined as the expectation over all discounted future rewards that could be earned from s. For an arbitrary, single-agent MDP, if an agent α_i follows a policy π_i:

$$V_{\pi_i}(s) = \mathbb{E}_{\pi_i}\left\{\sum_{t=0}^{\infty}\gamma^t R\left((s,a_i)^{(t)}\right)\middle| s^{(t)}=s\right\}.$$

The subscript in $\mathbb{E}_{\pi_i}\{\cdot\}$ means that the expectation is taken over the single agent Markov chain that is induced by α_i playing policy π_i in the MDP,

$$V_{\pi_i} = \mathbb{E}_{\pi_i}\left\{R\left((s,a_i)^{(t)}\right) + \gamma V_{\pi_i}(s^{(t+1)})\right\}.$$

State-action Value (Q) Functions

Likewise, the value of an action at a particular state is the expected discounted future reward that could be earned by following policy π_i:

$$Q_{\pi_i}(s,a_i) = \mathbb{E}_{\pi_i}\left\{\sum_{t=0}^{\infty}\gamma^t R\left((s,a_i)^{(t)}\right)\middle| s^{(t)}=s, a_i^{(t)}=a_i\right\}.$$

2.2.2 Q-function Approximation

To build an estimate of α_2's policy, $\hat{\pi}_2$, we'll parameterize α_2's *state-action value function*, $Q(s,a_2)$, with a linear-in-parameter model. Sugiyama describes this model in detail in Section 2.2.1 of [SO15], it is also used by [HLZP17] and [AHS10]. It is advantageous that α_1 does not need to learn the reward function of α_2, but only the

distribution of α_2's action given the joint state set, $\pi_2(s),\ s \in S$. Each parameter element determines the weighting of a feature $\phi : S \times A_2 \to \mathbf{R}$. If a total of W features are used then α_2's Q-function is approximated as

$$Q(s, a_2) \approx \hat{Q}(s, a_2) = \sum_{w=1}^{W} \hat{\theta}_w \phi_w(s, a_2) = \hat{\boldsymbol{\theta}}^\top \boldsymbol{\phi}(s, a_2).$$

The estimated parameter vector, $\hat{\boldsymbol{\theta}}$, represents feature weightings that must be learned for the best estimate of the Q-function.

Following Chapter 3 of [SO15], we'll use Geodesic Gaussian Kernels (GGKs) because they intuitively incorporate any obstacles[1] that exist in S into a distance metric between a state s, and the center of a GGK, c. Consider the state-graph of $\mathcal{M}$. If there are K kernels, and the l-th kernel has center c_l, then the value of the kernel at a state s is

$$k(s, c_l) = \exp\left(\frac{-\mathrm{SP}(s, c_l)^2}{2\sigma_l^2}\right).$$

The Shortest Path (SP) between a state s and the l-th kernel's center c_l can be precomputed, and the kernel standard deviation, σ_l, determines the effective support of the kernel. The feature function is defined for each action, so the number of features is $W = |A| \times K$. The w-th feature function is

$$\phi_w(s_2, a_2) = I(a_2 == a_2^{(j)}) \sum_{s_2' \in S_2} P\left(s_2' | s_2, a_2^{(j)}\right) k(s_2', c_l), \tag{2.3}$$

where $w = j + (l|A_2| - 1)$ represents the index used for both feature-vector functions and parameter elements, θ_w. Also, for each action a_2 the indicator function

[1] Obstacles could be considered as known sink-states in $\mathcal{M}$, and will be clarified in Sect. 2.4.

$I(a_2 == a_2^{(j)})$ is defined as:

$$I(a_2 == a_2^{(j)}) = \begin{cases} 0 & \text{if } a_2 \neq a_2^{(j)} \\ 1 & \text{if } a_2 = a_2^{(j)} \end{cases}$$

$$j = 1, \ldots, |A_2|,$$

assuming all actions are enabled from each state $s_2 \in S_2$.

Given a Q-function, [NNXS17] represents the policy at a state as represented the softmax over all actions,

$$\pi_2(a_2|s) = \exp((Q(s, a_2) - V(s))/\kappa), \tag{2.4}$$

where $V(s) = \kappa \log \sum_{a_2 \in A} \exp(Q(s, a)/\kappa)$. Also, κ is a temperature parameter. If $\kappa \to \infty$, the distribution $\pi_2(s)$ becomes uniform over A_2. If $\kappa \to 0$, the distribution $\pi_2(s)$ becomes a Dirac Delta distribution that peaks on the action $a_2^* = \text{argmax}_{a_2} Q(s, a_2)$. In the inference algorithm, we consider κ as a predefined hyperparameter in the range $(0, \infty)$.

This is the model used for π_2 in the rest of this report. The advantage of this parameterization is that we have reduced the number of parameters to learn from $|S| \times A$ to W. We'll show that we can effectively infer a policy with $K \ll |S|$.

The form used in Eq. 2.4 is equivalent to the Gibbs policy from [SO15]. It is referred to as a Boltzmann policy in [HLZP17], [KVTT17], and [HGW+16], although the authors have described this parameterization without reference to a temperature parameter, in which case $\kappa = 1$.

2.3 Policy Inference

For α_1 to plan a near-optimal policy for its own task, it must learn the policy of the other agent, which gives the model of the MDP. The estimate of α_2's policy, $\hat{\pi}_2$, is parameterized by a vector $\boldsymbol{\theta}$. Therefore, the estimated probability of a state transition is

$$q(s'|s, \boldsymbol{\theta}) = \sum_{a_1, a2 \in A} P(s'|s, (a_1, a_2))\hat{\pi}_2(a_2|s, \boldsymbol{\theta})\pi_1(a_1|s). \tag{2.5}$$

As the α_2 moves through S, the robot agent can observe the outcomes of α_2's actions, and build a set of observed state sequences.

Definition 2.3.1. *A trajectory τ is a sequence of joint states $s = (s_1, s_2)$ with time-step index t,*

$$s^{(0)}, s^{(1)}, \ldots, s^{(t)}, \ldots, s^{(|\tau|)};\ 0 \leq t \leq |\tau|.$$

Remark 2.3.2. *This definition is a key difference from comparable policy gradient algorithms, such as policy gradients with parameter-based exploration (PGPE) [TMZ+14] [SOR+10]. In lieu of state-action sequences, Section 2.3.1 will present an inference procedure that uses observed state-*action-outcome *sequences. The action-outcome is assigned to be the most probable* (maximum a posteriori) *action that can lead from* s *to* s′ *using the known transition function, $T(\cdot)$.*

Suppose the policy of agent one, π_1 is known. We'll define the probability of a trajectory as the joint probability of each set state-transition tuple for each of the

two distributions, p and q:

$$p(\tau) = \prod_{t=1}^{|\tau|} p\left(s^{(t)}|s^{(t-1)}\right),$$
$$q(\tau|\boldsymbol{\theta}) = \prod_{t=1}^{|\tau|} q\left(s^{(t)}|s^{(t-1)}, \boldsymbol{\theta}\right).$$

By using the parameter $\boldsymbol{\theta}$, $q(\tau, \boldsymbol{\theta})$ is the probability of replicating a trajectory given the parameterization.

Assumption 2.3.3. *The observed demonstration set, D, is sampled i.i.d. from the set of all possible demonstrations $\mathcal{D}$.*

The best inference of an environmental policy has a high likelihood of replicating the trajectories in D.

Lemma 2.3.4. *Minimizing the Kullback-Leibler divergence (KL-divergence) of the replica distribution from the observed trajectory distribution is equivalent to maximizing the log-likelihood of the observed state sequences given a parameterized policy. With a fixed policy for α_1, π_1,*

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, KL(p||q_{\boldsymbol{\theta}}).$$

Proof. Consider a pair of stationary policies for the two agents, π_1 and π_2. The induced Markov chain is $\mathcal{M}_{\pi_1,\pi_2}$. Let p be the probability distribution of paths in the chain $\mathcal{M}_{\pi_1,\pi_2}$. Let $q_{\boldsymbol{\theta}}$ be the probability distribution of paths in the chain $M_{\pi_1,\hat{\pi}_2}$. The KL-divergence from $q_{\boldsymbol{\theta}}$ to p is

$$\mathrm{KL}(p||q_{\boldsymbol{\theta}}) = \sum_{\tau_d \in D} p(\tau_d) \ln\left(\frac{p(\tau_d)}{q(\tau_d|\boldsymbol{\theta})}\right) = \sum_{\tau_d \in \mathcal{D}} P(\tau_d|D) \ln\left(\frac{P(\tau_d|\mathcal{D})}{P(\tau_d|\boldsymbol{\theta})}\right),$$

where $P(\tau_d|\mathcal{D})$ is the maximum likelihood probability of the state sequence, and $P(\tau_d|\boldsymbol{\theta})$ is the probability of obtaining that state sequence by our inferred policy that is parameterized by the vector $\boldsymbol{\theta}$.

Minimizing the deviation of $q_{\boldsymbol{\theta}}$ from p is equivalent to maximizing the expectation of the observing D, given that the environment actions are distributed as $\pi_2(s;\boldsymbol{\theta})$:

$$\begin{aligned}
\underset{\boldsymbol{\theta}}{\operatorname{argmin}}(\mathrm{KL}(p||q_{\boldsymbol{\theta}})) = & \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left(\sum_{\tau_d \in \mathcal{D}} P(\tau_d|\mathcal{D}) \ln \left(\frac{P(\tau_d|\mathcal{D})}{P(\tau_d|\pi_1, \boldsymbol{\theta})} \right) \right) \\
& = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left(\underbrace{\sum_{\tau_d \in D} P(\tau_d|D) \ln(P(\tau_d|D))}_{constant} - P(\tau_d|D) \ln \left(P(\tau_d|\pi_1, \boldsymbol{\theta}) \right) \right) \\
& = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left(\sum_{\tau_d \in \mathcal{D}} P(\tau_d|\mathcal{D}) \ln \left(P(\tau_d|\pi_1, \boldsymbol{\theta}) \right) \right) \\
& = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, \mathbb{E}_{P(\tau_d|\mathcal{D})} \left\{ \ln(P(\tau_d|\pi_1, \boldsymbol{\theta})) \right\} \\
& \approx \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\tau_d \in D} \ln(P(\tau_d|\pi_1, \boldsymbol{\theta})) \\
& = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, \mathcal{L}(D|\pi_1, \boldsymbol{\theta}),
\end{aligned} \tag{2.6}$$

where we estimate the expectation using the empirical mean. We will write the final line of Eq. 2.6 as $\mathcal{L}(D|\boldsymbol{\theta})$ for compactness and consistency with Lemma 2.3.4.

□

For the rest of this report, lets assert that an optimal parameter exists.

Assumption 2.3.5. *There exists an optimal parameter vector that can represent the true distribution of $\pi_2(s)$ to within a threshold ξ, given that a set of basis functions are properly defined;*

$$\exists \; \boldsymbol{\theta}^* \;\Big|\; ||\hat{\pi}_2(s;\boldsymbol{\theta}^*), \pi_2(s)||_1 \leq \xi.$$

The infinite L_1-norm between two policies,

$$||\pi_x(s), \pi_y(s)||_1 = \sum_{s \in S} \sum_{a \in A} \pi_x(a|s) - \pi_y(a|s),$$

is a measurable distance unlike KL-divergence; $\mathrm{KL}(p||q) \neq \mathrm{KL}(q||p)$. For all following experiments, we'll use the L_1-norm to compare two policies.

We are now ready to discuss the inference procedure used to identify the best estimate of $\pi_2(s, \boldsymbol{\theta})$ that maximizes the R.H.S of Lemma 2.3.4.

2.3.1 Gaussian Distribution of Policy Parameters

After a data set of trajectories, D, has been collected, α_1 needs to maximize $\mathcal{L}(D|\boldsymbol{\theta})$, the log-likelihood of the dataset when π_2 is parameterized by an estimated parameter vector $\hat{\boldsymbol{\theta}}$. Let $\hat{\boldsymbol{\theta}} = [\theta_w]_{w=1}^W$ be a vector of independently sampled random variables with Gaussian distributions $\mathcal{N}(\mu_w, \nu_w^2)$ for $w = 1, \ldots, W$. The variance, ν_w^2, will capture the uncertainty of θ_w in the inference from dataset D.

The following is similar to the analysis in [TMZ+14], [HGW+16], and [SOR+10] except that we do not include a reward function because the policy sought must replicate the observed data D, not earn a reward.

We denote $\rho_w = (\mu_w, \nu_w)$ as the tuple of mean and variance for θ_w and denote $\rho = \{\rho_w\}_{w=1}^W$ to be the collection of variable tuples. Given ρ, the probability of the demonstrations is

$$P(D|\rho) = \int_{\boldsymbol{\theta}} P(D|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\rho) d\boldsymbol{\theta}.$$

The log-likelihood of the demonstrations can be lower-bounded using Jensen's inequality:

$$\mathcal{L}(D|\rho) = \log P(D|\rho) \ = \log \left(\int_{\boldsymbol{\theta}} P(D|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\rho) d\boldsymbol{\theta} \right) \ \geq \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\rho) \log \big(P(D|\boldsymbol{\theta}) \big) d\boldsymbol{\theta}.$$

Denote this lower bound as $\tilde{\mathcal{L}}(D|\rho) = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\rho) \log P(D|\boldsymbol{\theta}) d\boldsymbol{\theta}$. This is the lower bound on the objective function derived in Eq. 2.6. By taking derivative of $\tilde{\mathcal{L}}(D|\rho)$ with respect to ρ, we obtain the gradient of the objective function:

$$\begin{aligned}
\nabla_\rho \tilde{\mathcal{L}}(D|\rho)) &= \int_{\boldsymbol{\theta}} \nabla_\rho p(\boldsymbol{\theta}|\rho) \log P(D|\boldsymbol{\theta}) d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}} [p(\boldsymbol{\theta}|\rho) \nabla_\rho \log p(\boldsymbol{\theta}|\rho)] \log P(D|\boldsymbol{\theta}) d\boldsymbol{\theta} \\
&\approx \frac{1}{m} \sum_{i=1}^{m} \left[\nabla_\rho \log P(\boldsymbol{\theta}^{(i)}|\rho) \right] \log P(D|\boldsymbol{\theta}^{(i)})
\end{aligned} \tag{2.7}$$

where $\boldsymbol{\theta}^{(i)}$, $i = 1, \ldots, m$ are samples generated from the multi-variant Gaussian distribution with mean $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_W]^\top$ and covariance matrix diag $(\nu_1, \ldots, \nu_W)$. Let $\boldsymbol{\nu}$ be an equivalent representation for diag $(\nu_1, \ldots, \nu_W)$. Each sampled parameter element, $\theta_w^{(i)}$, has probability:

$$P(\theta_w^{(i)}|\rho_w) = \frac{1}{\sqrt{2\pi\sigma_w^2}} \exp\left(-\frac{(\theta_w^{(i)} - \mu_w)^2}{2\sigma_w^2} \right).$$

The bracketed gradient components in the last line of Eq. 2.7 with respect to each element of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are:

$$\nabla_{\mu_w} \log P(\theta_w^{(i)}|\rho_w) = \frac{\theta_w^{(i)} - \mu_w}{\nu_w^2}, \text{ and}$$

$$\nabla_{\nu_w} \log P(\theta_w^{(i)}|\rho_w) = \frac{(\theta_w^{(i)} - \mu_w)^2 - \nu_w^2}{\nu_w^3}.$$

Note that superscripts enclosed in parenthesis represent sample indexes, e.g., the i-th sample of parameter element w is $\theta_w^{(i)}$. All purely numeric superscripts are exponents.

We can obtain the optimal collection of parameters $\rho^* = \text{argmax}_\rho \tilde{\mathcal{L}}(D|\rho)$ by

performing gradient ascent on the parameter distributions, $\rho = (\boldsymbol{\mu}, \boldsymbol{\nu})$. The policy parameterized by $\hat{\boldsymbol{\theta}}^* \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\nu}^*)$ is $\hat{\pi}_2(s; \hat{\boldsymbol{\theta}}^*)$ and it maximizes the log likelihood of the demonstration set D. The log likelihood of observed demonstrations for a given $\boldsymbol{\theta}^{(i)}$ can be computed as

$$\log P(D|\boldsymbol{\theta}^{(i)}) = \sum_{\tau_d \in D} \log P\left(\tau_d | \boldsymbol{\theta}^{(i)}\right) \tag{2.8}$$

$$= \sum_{d=1}^{|D|} \left[\sum_{t=0}^{|\tau_d|-1} \log P\left(s^{(t+1)} | (s, a_1, o_2)^{(t)}\right) + \sum_{t=0}^{|\tau_d|-1} \log \pi_2\left(o_2^{(t)} | s^{(t)}; \boldsymbol{\theta}^{(i)}\right) \right] \tag{2.9}$$

$$= \sum_{s \in S} \sum_{o_2 \in A} C(s, o_2) \log \pi_2\left(o_2 | s; \boldsymbol{\theta}^{(i)}\right) + Const. \tag{2.10}$$

Above, $C(s, o_2)$ is the number of times the state action pair (s, o_2) is observed from in D. The constant term, $Const = \sum_{d=1}^{|D|} \sum_{t=0}^{|\tau_d|-1} \log P\left(s^{t+1} | (s, a_1, o_2)^{(t)}\right)$, is independent of $\boldsymbol{\theta}^{(i)}$ and can be precomputed for a demonstration D. The observed action outcome at time-step t is o_2^t, which is the only action information available in a trajectory, per Definition 2.3.1. If a trajectory fragment $(s_1, s_2)^{(t)}, (s_1', s_2')^{(t+1)}$ is observed, the action of the controllable agent, $a_1^{(t)}$, is known but the uncontrolled action, $a_2^{(t)}$, is not. Therefore $o_2^{(t)}$ is assigned to be the nominal motion that causes the transition $s_2^{(t)} \to s_2^{(t+1)}$ in the graph of $\mathcal{M}$.

Remark 2.3.6. *If the policy does not depend on a basis function ϕ_w, $\mu_w = 0$ and $\nu_w \ll 1$, the basis function can be removed from the parameterization since it has no influence on the inferred policy.*

Gradient Ascent

Using the gradient defined in Eq. 2.7, for each iteration n we sample a set of m parameter vectors, and update the distribution parameters on each iteration:

$$\begin{aligned} \dot{\boldsymbol{\mu}}_n &\leftarrow \eta\dot{\boldsymbol{\mu}}_{n-1}\lambda\nabla_{\boldsymbol{\mu}_n}\tilde{\mathcal{L}}(D|\rho_n) \\ \boldsymbol{\mu}_{n+1} &\leftarrow \boldsymbol{\mu}_n + \dot{\boldsymbol{\mu}}_n \text{ and} \\ \dot{\boldsymbol{\nu}}_n &\leftarrow \eta\dot{\boldsymbol{\nu}}_{n-1}\lambda\nabla_{\boldsymbol{\nu}_n}\tilde{\mathcal{L}}(D|\rho_n) \\ \boldsymbol{\nu}_{n+1} &\leftarrow \boldsymbol{\nu}_n + \dot{\boldsymbol{\nu}}_n. \end{aligned} \tag{2.11}$$

The step-size parameter, λ, limits the rate of change of the distribution moments, and the velocity memory, η helps the iteration bootstrap itself through local minimums as suggested by [KB14]. The "velocity" of each gradient is stored in $\dot{\boldsymbol{\mu}}_n$ and $\dot{\boldsymbol{\nu}}_n$, respectively.

Remark 2.3.7. *We do notice that the gradient variability is also a function of the size of D, as concluded by [TMZ+14]. Therefore the step size, λ, is a hyper-parameter that is dependent on the experiment.*

Algorithm Termination

In general, the gradient ascent should be terminated at some final iteration N when the update to the parameters no longer improves $\tilde{\mathcal{L}}(D|\rho)$. This log-likelihood is bounded,

$$\log P(D|\boldsymbol{\theta}^{(i)}) \leq 0, \ \forall\boldsymbol{\theta} \in \Theta,$$

where Θ is the domain of the parameter vector. Due to the nature of sampling, there is no guarantee that for every iteration $\mathcal{L}(D|\rho_{n+1}) > \mathcal{L}(D|\rho_n)$. Therefore, we'll use

a moving average of the past Λ log-likelihoods,

$$\mathsf{HIST}(\mathcal{L}_n) = \frac{1}{\Lambda}\sum_{v=0}^{\Lambda-1}\mathcal{L}(D|\rho_{n-v}).$$

We record the previous value of the moving average, $\mathsf{HIST}(\mathcal{L}_{n-1})$, and if the improvement in the moving average is below a defined threshold,

$$\Delta\mathsf{HIST}(\mathcal{L}_n) = \mathsf{HIST}(\mathcal{L}_n) - \mathsf{HIST}(\mathcal{L}_{n-1}) \leq \zeta,$$

the algorithm will terminate. We'll require that at least N iterations are performed before termination. Upon termination the mean values of ρ_n are assigned to the parameter vector used to build $\hat{\pi}_2(s;\hat{\boldsymbol{\theta}})$, $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\mu}_n$.

2.4 Single-agent policy inference experiment

The algorithm and parameterization presented in Sections 2.2.2-2.3.1 are initially tested in a single agent environment. In this example, the states are just $s = s_2 \in S$, and $\pi_1(a_1|s)$ is dropped from Eq. 2.2.

2.4.1 Simulation environment

Assume that α_2 exists alone in a 5×5 grid world. In a single agent simulation, states and grid-cells are synonymous. The available action set is

$$A = \{Empty, North, South, East, West\},$$

which correspond to their motion primitives

$$\{Stay, Up, Down, Right, Left\}.$$

The motion resulting from an action a_2 is stochastic; the intended motion is actually executed with a probability of 0.8, and there is a probability of 0.1 of sliding in the two perpendicular motion directions. The exception is the *Empty* action which results in a *Stay* motion with probability 1. The boarders of the grid-world are considered to be walls. If the resulting motion would cause the agent to leave the grid then *Stay* is selected as the outcome, $s' = s$. Table 2.4.1 clarifies the motion model for the *East* action based on the starting cell location.

	Starting grid-cell location type			
Resulting Motion	Middle	Right Column	Top Row	Upper Right Corner
North	0.1	0.1	0.0	0.0
South	0.1	0.1	0.1	0.1
East	0.8	0.0	0.8	0.0
West	0.0	0.0	0.0	0.0
Stay	0.0	0.8	0.1	0.9

Table 2.1: Agent motion model given attempted *East* (right) action.

The *hidden* policy of α_2 is visualized in Fig. 2.1. The policy is deterministic, $\kappa = 0$, and the agent has a hidden goal of the yellow cell. The red cells are obstacles, are *not* hidden, and are sink states. The Shortest Path (SP) in metric from a cell to a kernel center, Eq. 2.3, accounts for the distance around known obstacles. Arrows represent the direction an action is selected with probability 1, and dots represent that the *Empty*/*Stay* action is selected with probability 1. A demonstration set D is generated with a uniform initial distribution $s^{(0)} \sim \mathcal{U}$, $\forall \tau_d \in D$. The state visitation count for the example D is visualized in Fig. 2.2

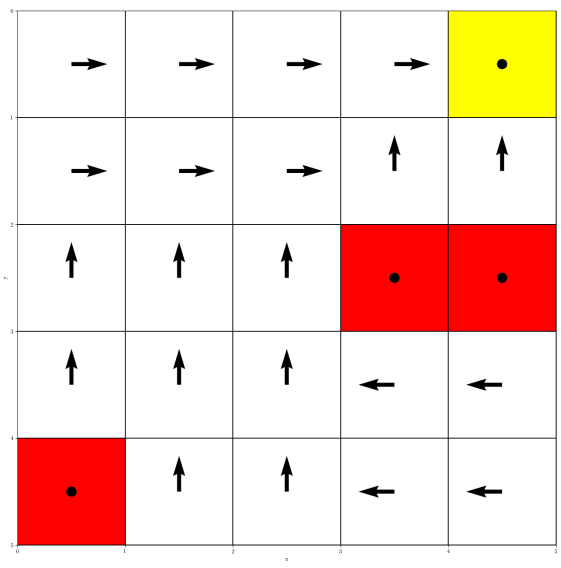

Figure 2.1: True Policy of α_2.

2.4.2 Experiment Hyper-parameters

To approximate the Q-function of α_2, per Section 2.2.2, let's start by placing a kernel centered at every grid cell. This sets $W = |S| \times A$, which is an unrealistic number of parameters but, it's a good test. See Fig. 2.3 for a visualization of how the the kernel function values, $k(s, a_2)$, at each state are mapped to the feature vector-function $\boldsymbol{\phi}(s, a_2)$. Also, the initial guess of π_2 is set to be uniform across the entire action set available to α_2. We'll set the following parameters:

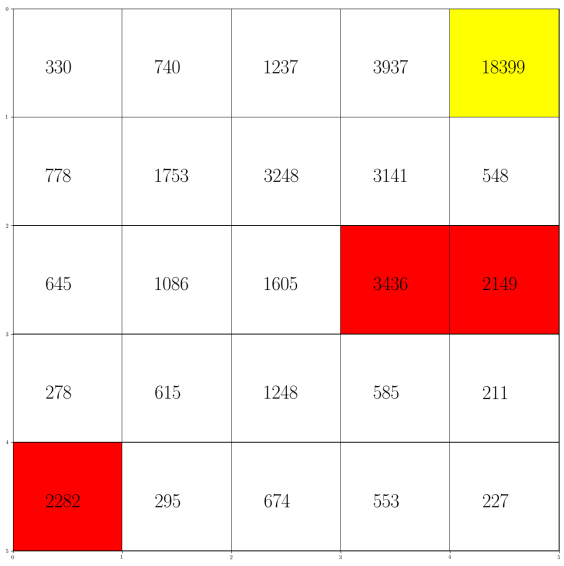

Figure 2.2: State visitation count in single agent demonstration.

σ_l	1.1, $\forall l$	Identical kernel standard-deviations
κ	0.1	Temperature of $\hat{\pi}_2$. Eq. (2.4)
λ	1e$-$5	Gradient update rate. Eq. (2.11)
η	0.0	Gradient velocity memory
m	5000	Per iteration sample size of $\boldsymbol{\theta} \sim \rho$
Λ	60	Moving average buffer length for $\mathsf{HIST}(\mathcal{L})$
ζ	0.001	Gradient ascent termination when $\Delta\mathsf{HIST}(\mathcal{L}) < \zeta$
μ_0	0.0	Initial parameter means
ν_0	1.0	Initial parameter standard-deviations
ν_{min}	0.2	Minimum parameter standard-deviation

Table 2.2: Hyper-parameters used for single agent inference.

$\|\tau\|$	10	Number of steps in a trajectory
$\|D\|$	5000	Number of trajectories observed
I_0	$\mathcal{U}(0, 24)$	Uniform distribution of $s_2^{(0)}$

Table 2.3: Observed data for single agent inference.

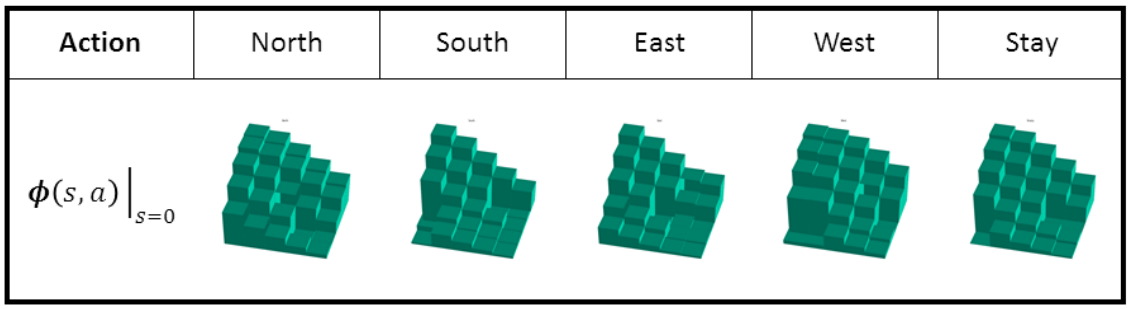

Figure 2.3: Feature values for a kernel centered at cell 0.

Algorithm bounds

As noted in footnote 2 of [Wil92], there is not gradient step-size, λ (this reports notation), that will keep the parameter variance elements $\nu_w > 0$. Our remedy is to enforce a lower bound on the covariance of ρ_n after each gradient update in Eq. 2.11:

$$\nu_{min} \leq \boldsymbol{\nu}_n, \ \forall \nu_w \in \boldsymbol{\nu}_n. \tag{2.12}$$

2.4.3 Results

Our Assumption 2.3.5 is verified by Figures 2.4 and 2.5. The recorded L_1-norm was neither used during gradient ascent, nor as a termination criteria. Note that the final L_1-norm ≈ 0.7. This final error has a range of about $[0.01 - 5]$ with these parameters over different trials. We can also examine the dynamics of the distribution ρ as well, see Figures 2.6 and 2.7.

Note that the legend entry "$\max_{i \in m} \tilde{\mathcal{L}}_n(D|\theta^{(i)})$" represents the *sampled* parameter vector that maximizes the log-likelihood of D at the iteration n. The policy estimate $\hat{\pi}_2(\hat{\boldsymbol{\theta}})$ is constructed using μ_N, the mean vector of the multi-variate distribution ρ at the final iteration, N.

As the iterations progress, notice that the parameter variances decrease in Fig. 2.12, implying that the value of of each parameter is known with more confidence.

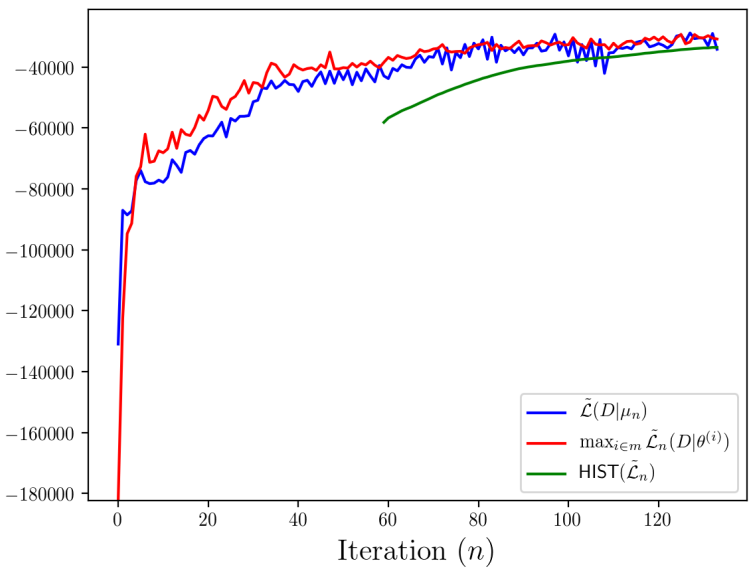

Figure 2.4: $\tilde{\mathcal{L}}(D|\rho)$ with parameters in Table 2.2.

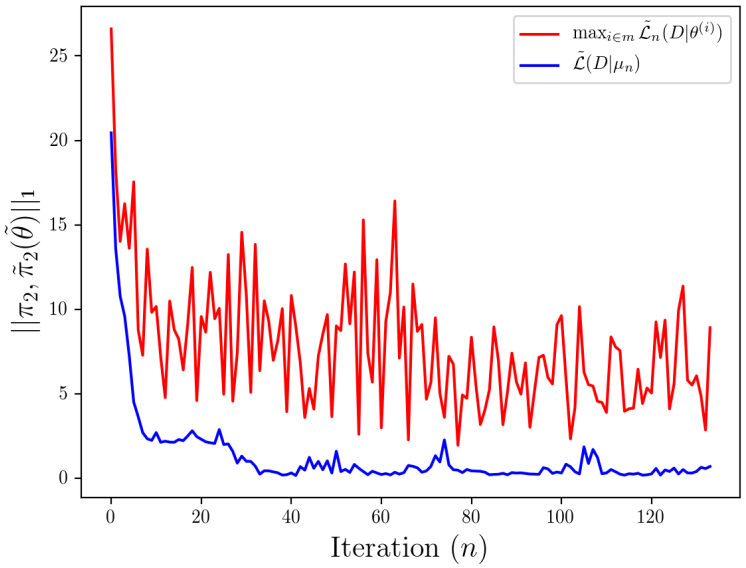

Figure 2.5: $||\pi_2, \hat{\pi}_2||_1$ with parameters in Table 2.2.

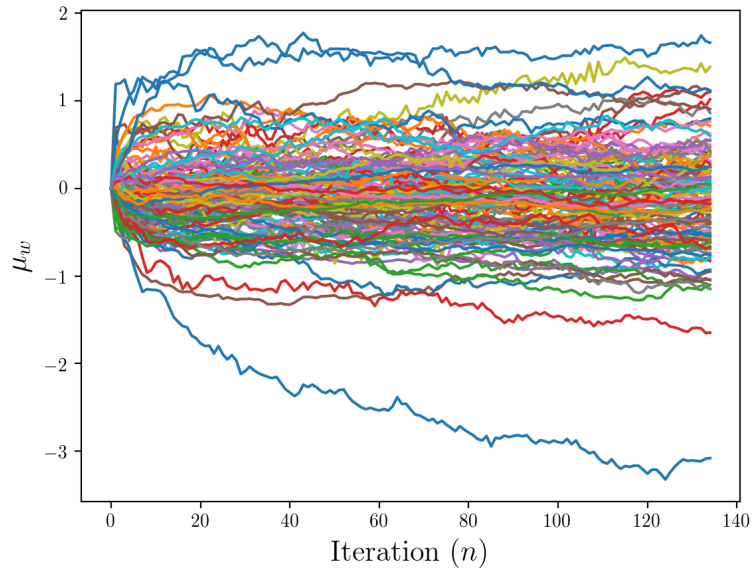

Figure 2.6: μ_w for each iteration with parameters in Table 2.2.

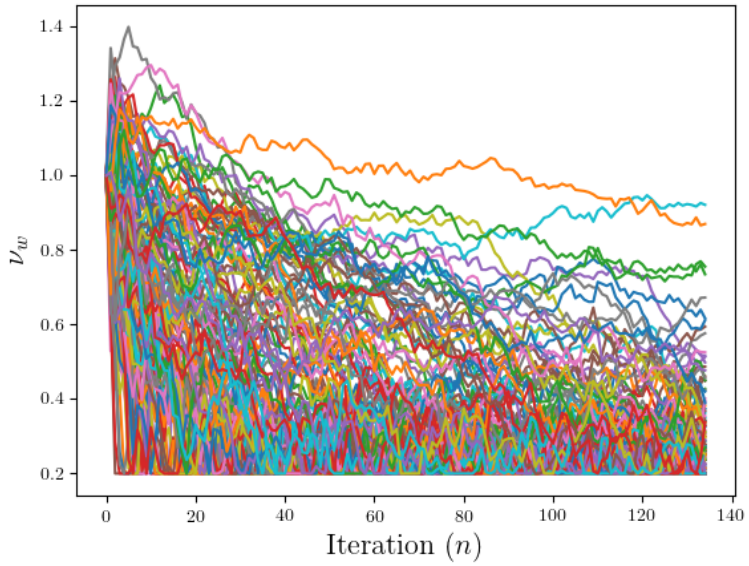

Figure 2.7: ν_w for each iteration with parameters in Table 2.2.

2.4.4 Experiment with Fewer Kernels

Note that if we use $K = 7$, and use the parameters in Table 2.4, we can achieve a final L_1-norm ≈ 6.5. There is some obvious generalization error in the inferred policy shown in Fig. 2.8. Note that the grid-cells with blue circles in them note only that a kernel is centered at that location, the circle is not proportional to σ_l. However, the magnitudes of the dots and arrows are proportional to the probability of selecting that action; the yellow cell is in the most North-East cell.

σ_l	$\mathbf{2.0}$, $\forall l$	Identical kernel standard-deviations.
κ	0.1	Temperature of $\hat{\pi}_2$. Eq. (2.4).
λ	$\mathbf{1e{-}6}$	Gradient update rate. Eq. (2.11)
η	$\mathbf{0.2}$	Gradient velocity memory.
m	5000	Per iteration sample size of $\boldsymbol{\theta} \sim \rho$.
Λ	60	Moving average buffer length for $\mathsf{HIST}(\mathcal{L})$.
ζ	0.001	Gradient ascent termination when $\Delta\mathsf{HIST}(\mathcal{L}) < \zeta$.
ν_{min}	0.2	Minimum parameter standard-deviation.

Table 2.4: Hyper-parameters for inference with $K = 7$.

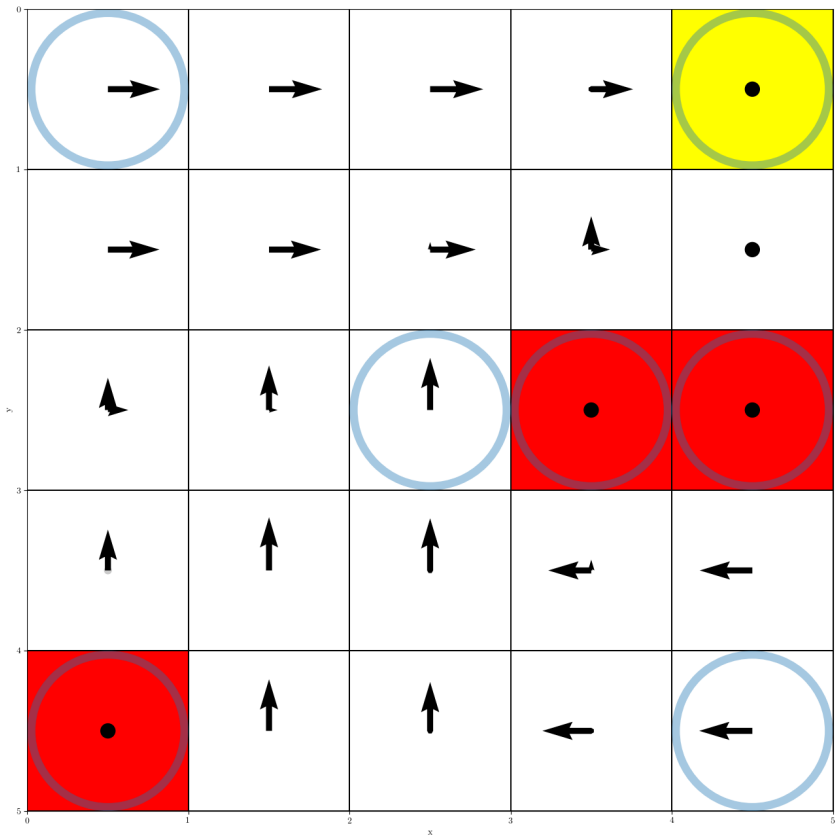

Figure 2.8: $\hat{\pi}_2$ with parameters in Table 2.4.

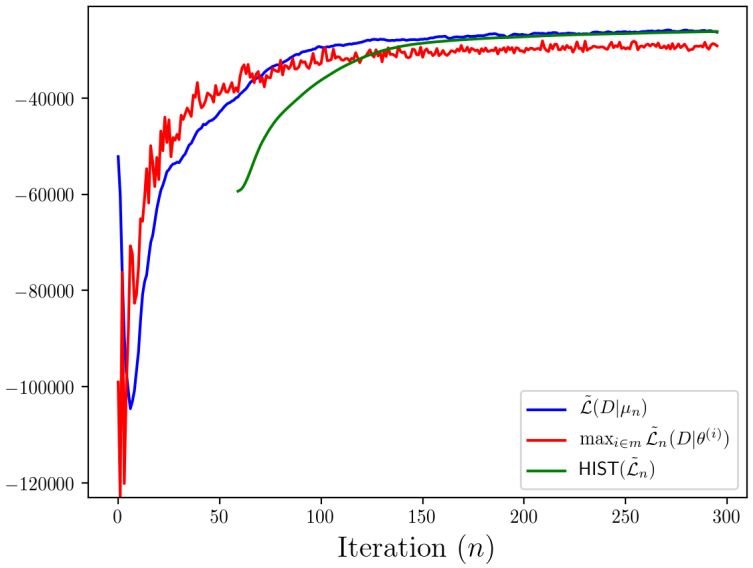

Figure 2.9: $\tilde{\mathcal{L}}(D|\rho)$ with parameters in Table 2.4.

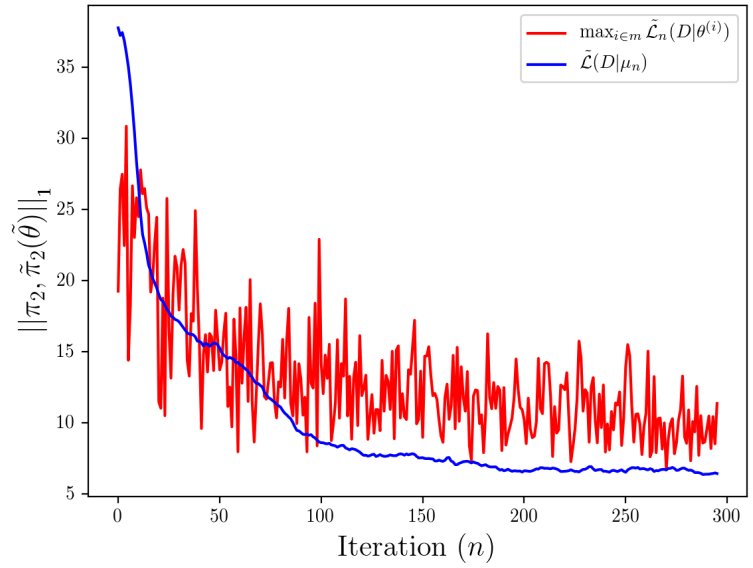

Figure 2.10: $||\pi_2, \hat{\pi_2}||_1$ with parameters in Table 2.4.

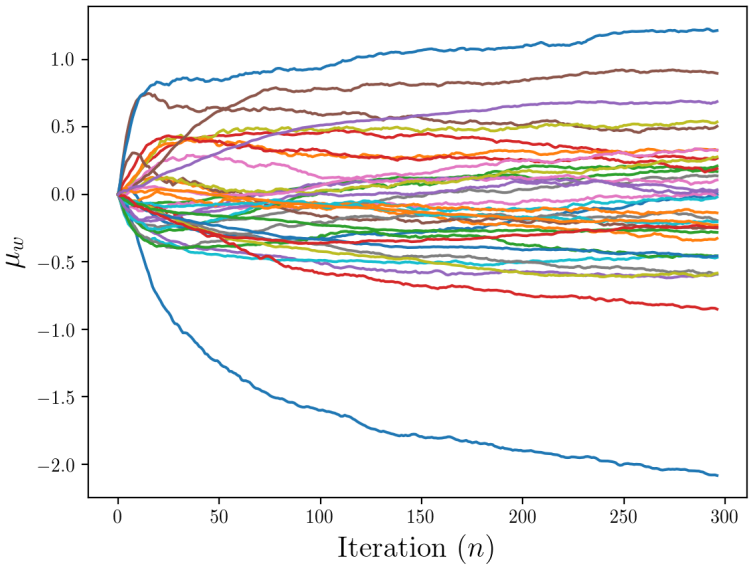

Figure 2.11: μ_w with parameters in Table 2.4.

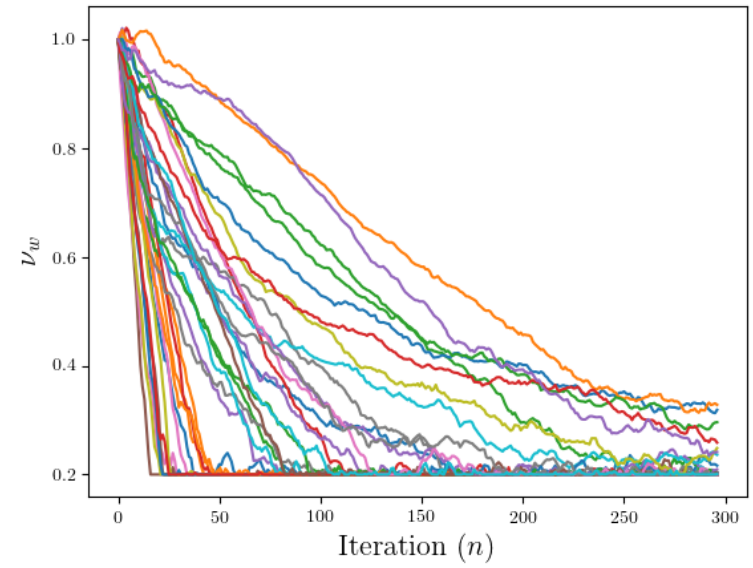

Figure 2.12: ν_w with parameters in Table 2.4.

Chapter 3

Policy Inference in a Multi Agent Environment

Now that Chapter 2 has introduced the inference algorithm, we'll explore the inference procedure in a multi agent environment. This requires two additional topics to be covered, (1) how α_1 will build a policy to complete it's task, and (2) how α_1 can determine the dependence of π_2 on the relative position between α_1 and α_2. Experimental results will show the inference algorithm in the multi-agent environment. To perform the experiment, we'll modify true π_2 from Fig. 2.1 to be repulsed by the position of α_1.

3.1 Policy Synthesis for the Controllable Agent

Policy Solution

For some agent α_i, a policy π_i^* is said to be *optimal* if it maximizes the expected total discounted reward from a given initial distribution I_0. For a deterministic

policy this is,

$$\pi_i^* \leftarrow \arg\max_{\pi_i} \sum_{s \in S} V_{\pi_i}(s) I_0(s). \tag{3.1}$$

In an MDP with stochastic transitions, a common way to solve for a policy is by iteratively updating the value function presented in Section 2.2.1. If the policy of α_2 was known, then iteratively solving the Bellman equation [HL12] will converge to a policy that takes a discount optimal action at each state. This is known as Value Iteration (VI). Note that if we use the true policy of α_2, an optimal policy, within a prescribed tolerance, will eventually be found via:

$$\begin{aligned} V^*(s) &= \max_{a_1} \left[R(s, a_1) + \gamma \sum_{s'} \sum_{a_2 \in A} T(s'|s, (a_1, a_2)) \pi_2(a_2|s) V^*(s') \right], \text{ and} \\ \pi_1^*(s) &= \operatorname*{argmax}_{a_1} \left[R(s, a_1) + \gamma \sum_{s'} \sum_{a_2 \in A} T(s'|s, (a_1, a_2)) \pi_2(a_2|s) V^*(s') \right]. \end{aligned} \tag{3.2}$$

Given some estimated policy of agent two, $\hat{\pi}_2$, the best α_1 can do is to solve for some sub-optimal policy $\pi_1(s; \hat{\boldsymbol{\theta}})$:

$$\begin{aligned} V(s; \hat{\boldsymbol{\theta}}) &= \max_{a_1} \left[R(s, a_1) + \gamma \sum_{s'} \sum_{a_2 \in A} T(s'|s, (a_1, a_2)) \hat{\pi}_2(a_2|s; \hat{\boldsymbol{\theta}}) V(s') \right], \text{ and} \\ \pi_1(s; \hat{\boldsymbol{\theta}}) &= \operatorname*{argmax}_{a_1} \left[R(s, a_1) + \gamma \sum_{s'} \sum_{a_2 \in A} T(s'|s, (a_1, a_2)) \hat{\pi}_2(a_2|s; \hat{\boldsymbol{\theta}}) V(s') \right]. \end{aligned} \tag{3.3}$$

In Section 4.3.1, we'll discuss how to iteratively improve $\hat{\boldsymbol{\theta}}$ and cover the convergence of $\pi_1(s; \hat{\boldsymbol{\theta}}) \rightarrow \pi_1^*(s)$.

3.1.1 EM-based Approximate Optimal Control

Solving Equations 3.2 or 3.3 can be very tedious, especially as the state space grows. To approximately solve for α_1's policy, we'll use the Expectation Maximization (EM)

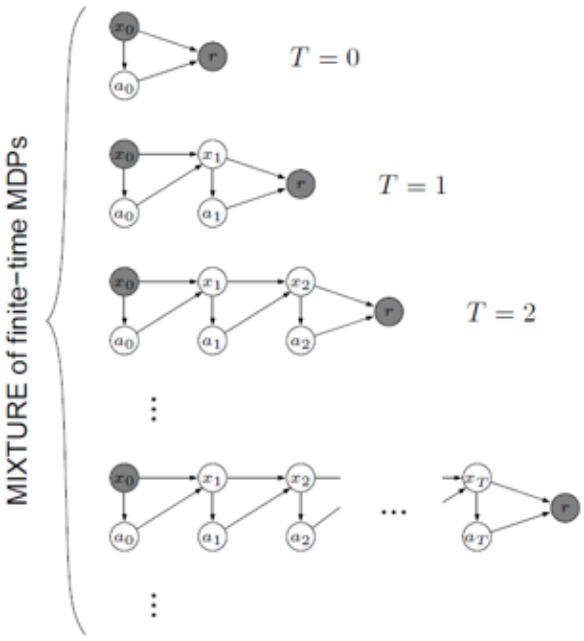

Figure 3.1: Mixture of finite-time MDPs. Note this report uses notation that each finite-time MDP ends at time $\mathcal{T}$ instead of T. Image courtesy of [TSH10].

solution from [TSH10]. The computational requirements to solve for a sub-optimal policy for α_1 are much less than traditional value iteration; see Figure 1.4 in [TSH10]. Although the solution is sub-optimal, the solution is complete for the entire state-action-space. Since EM is traditionally used as a clustering algorithm [DLR77], the resulting policy is not a hardmax (deterministic) like Eq. 3.1 but rather a softmax (stochastic).

Policies from Expectation Maximization

This section presents a summary of the EM procedure for policy synthesis. Toussaint and Storkey showed in [TSH10] that EM can be used to approximate the MDP as a mixture of finite-time MDPs, as shown in Fig. 3.1.

Their derivation starts with the joint distribution of the probability of a reward $\mathcal{R}$, a sequence of states τ, and the probability of the sequence ending at time $\mathcal{T}$,

subject to a policy π_1. Note that $s^{(\mathcal{T})}$ is the state at time $\mathcal{T}$ in the joint distribution,

$$\begin{aligned} P(R, \tau, \mathcal{T}; \pi_1) &= P(R|\tau; \pi_1) P(\tau|\mathcal{T}; \pi_1) P(\mathcal{T}) \\ &= P(R|s^{(\mathcal{T})}; \pi_1) \left[\prod_{t=0}^{\mathcal{T}-1} P(s^{(t+1)}|s^{(t)}; \pi_1) \right] P(s^{(0)}; \pi_1) \delta_{|\tau|\mathcal{T}} P(\mathcal{T}), \end{aligned} \tag{3.4}$$

where $\delta_{|\tau|\mathcal{T}} = 1$ when the trajectory ends at step $\mathcal{T}$ and is zero otherwise. The algorithm requires that each MDP in the mixture only emits a reward at a single final time step; the final states, r, in Fig. 3.1 emit rewards from the distribution $\mathcal{R}(s^{(\mathcal{T})}, a_1^{(\mathcal{T})})$ which is defined:

$$P(\mathcal{R}|s, a_t) = \frac{R(a_1, s) - \min_{a_1,s}(R)}{\max_{a_1,s}(R) - \min_{a_1,s}(R)}.$$

The goal of EM is to maximize the following energy function,

$$F(\pi_1, \omega) := \log P(\mathcal{R}; \pi_1) - KL(\omega(\tau, \mathcal{T})||P(\tau, \mathcal{T}|\mathcal{R}; \pi_1).$$

$F(\pi_1, \omega)$ is the difference between the log-likelihood of the reward probability subject to π_1, and the KL-divergence of the latent variable distribution ω from the latent variables given a reward and subject to π_1. The latent, unknown, variables of the MDP are the state-action sequence that brings the system to a final state, $s^{(\mathcal{T})}$, at time $\mathcal{T}$. Note that only sequences where $|\tau| = \mathcal{T}$ provide a non-zero probability of reward in the mixture of finite-time MDPs.

The M-step returns the energy function maximized over the latent variables using

the previous, *old*, iteration's policy:

$$F(\pi_1, \omega^*) = \sum_s \left[P(\mathcal{R}|s^{(T)}; \pi_1^{old})\alpha(s)\right] \log P(\mathcal{R}|s^{(T)}; \pi_1^{old}) + \sum_{s',s} \left[\beta(s')P(s'|s; \pi_1^{old})\alpha(s)\right] \log P(s^{(t+1)}|s^{(t)}; \pi_1^{old}), \tag{3.5}$$

where

$$\alpha(s^{(t)}) = \sum_{t=0}^{\infty} P(s^{(t)}; \pi_1^{old})P(\mathcal{T} = t), \text{ and}$$
$$\beta(s^{(t+1)}) = \frac{1}{1-\gamma} \sum_{\upsilon=0}^{\infty} P(R|s^{(t)\prime}, \mathcal{T} = t + \upsilon; \pi_1^{old})P(\mathcal{T} = \upsilon + 1).$$

In the equation for β, above, the term $s^{(t)\prime}$ refers to reachable states from $s^{(t)}$.

Essentially, the M-step provides a policy update. The E-step, given a policy, finds the state-action sequence that maximizes the expectation of a reward; the E-step calculates α and β. These two terms are then used in the M-step to provide the policy for future iterations, up to Z iterations. The terms α and β are updated on a state-action horizon time-limit of H. Toussaint et al. in [TSH10] show that as $Z \to \infty$, and $H \to \infty$, the EM algorithm returns a policy equivalent to that from VI. The parameters Z and H are now added to the set of hyper-parameters for a multi-agent simulation

Remark 3.1.1. *In the experiment implementation in the following sections and chapters, note that if the horizon parameter H is too short and the reward space is sparse, states that are too far from receiving a reward will select the* Empty *action.*

Section 3.3 will compare the policy synthesis of EM and VI, but we first must define the true policy of α_2 in a multi-agent environment.

3.2 Multi-agent Policy Model

When α_1 exists in the environment with α_2, the state space in our HiP-MDP, $\mathcal{M}$, is again $S \equiv S_1 \times S_2$. In the case of the experiment in Section 2.4, the cardinality of S, $|S|$, has exploded to 25^2. Given the number of actions available at each state, 5, an approximation of π_2 must represent the probability of selecting each action at each state; the cardinality of the policy estimate, $|\hat{\pi}_2|$, has increased from 125 to 3125. How can we design a set of kernels, so that $K \ll |S|$ and minimize $\left|\left|\pi_2(s), \hat{\pi}_2(s; \hat{\boldsymbol{\theta}})\right|\right|_1$?

For the example tasks described in Chapter 1, a human would infer that the uncontrollable agents would only change their actions if the controllable agent is within a relative distance. Other cars might only swerve away from a driver if two cars become dangerously close. Alternatively, a human holding a basket might only extend their arms to accept a load of items when the manipulator is sufficiently close to them. With this assumption, we'll make the inference task computationally manageable.

For simplicity, we've adjusted the grid-world example from that used in Section 2.4 one with no obstacles and deterministic transitions. The motivation for this will be discussed in Chapter 4, but by limiting the stochasticity of the environment, the interactions of α_1 and α_2 are easier to influence.

3.2.1 Fixed and Mobile kernels for policy inference

Assumption 3.2.1. *The uncontrollable agent, α_2, has a nominal policy, and when the agents are within some unknown distance* x*, π_2 is biased either towards or away*

from the position of α_1*:*

$$\pi_2 = \begin{cases} f(s_2) & \textit{if } \mathrm{SP}(s_1, s_2) > \mathsf{x} \\ f(s_1, s_2) & \textit{if } \mathrm{SP}(s_1, s_2) \leq \mathsf{x}. \end{cases}$$

Using Assumption 3.2.1, then we can segment our parameter and feature vectors into fixed and mobile elements. The fixed kernels are still scattered at locations $c_l \subset S_2,\ l = 0, \ldots, K-1$. Now, consider that the next, K^{th}, kernel is mobile; it is attached to the α_1's location. When this kernel is far from the current state of α_2 its anticipated influence is very small due to Eq. 2.3. This simply requires that another $|A|$ parameters elements are included in the $\boldsymbol{\theta}$ from the experiments in Section 2.4.

3.2.2 True Multi-Agent Policy of Agent 2

To perform multi-agent experiments, let's now synthesize a true policy for α_2 that is a combination of the policy shown in Fig. 3.2 and a repulsive effect from the proximity to α_1. To form the policy in Fig. 3.2, the experiment in Sec. 2.4.3 was repeated, thus the light-blue kernel markers on each grid-cell. With these inference results, we have a set of parameters, $\boldsymbol{\theta}^*_{\mathsf{FIXED}}$ that satisfy Assumption 2.3.5 for that stationary distribution such that the true Q-function of α_2 is:

$$Q(s, a_2) = \hat{Q}(s, a_2; \boldsymbol{\theta}^*_{\mathsf{FIXED}}) = \sum_{w=1}^{W_{\mathsf{FIXED}}} \theta^*_w \phi_w(s, a_2),$$

where $W_{\mathsf{FIXED}} = K_{\mathsf{FIXED}} \times |A|$ and $K_{\mathsf{FIXED}} = 25$ and $|A| = 5$. These 25 kernels are at fixed locations in the grid-world.

To represent the repulsion that α_2 should desire as it α_1 approaches, we'll attach

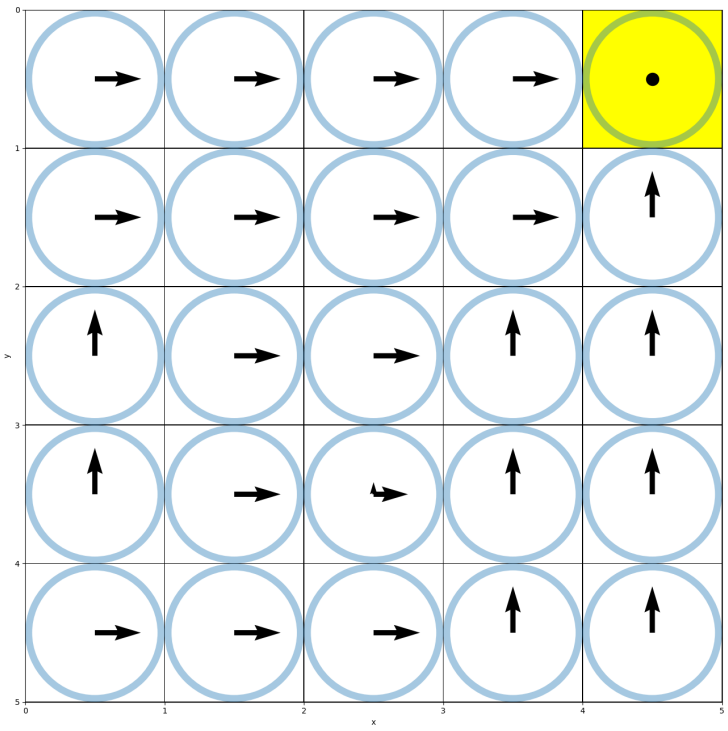

Figure 3.2: Hidden Policy of α_2 in an empty, deterministic, world.

a single mobile kernel to the location of α_1. The kernel function for this kernel is identical to Eq. 2.3, except that its center is mobile: $c_{\mathsf{MOBILE}} = s_1^{(t)}$ for all $t \in [0, |\tau|)$. With the associated feature-function ϕ_{MOBILE}, we'll associate five parameter values that will decrease the value of α_2's Q-function if α_1 is close. All the parameters used to synthesize a true policy for α_2 are:

$\boldsymbol{\theta}^*_{\mathsf{FIXED}}$	–	Results used for Fig. 2.5
c_{FIXED}	–	Grid Cells $0, \dots 24$
σ_{FIXED}	2.0	Fixed kernel standard deviations
c_{MOBILE}	–	Described above
σ_{MOBILE}	**1.0**	Mobile kernel standard-deviations.
$\theta_{\mathsf{MOBILE}\ \forall\ a\in\ \{N,S,E,W\}}$	-0.9	Negative action value for $North, South, East, West$
$\theta_{\mathsf{MOBILE}\ a=\{Empty\}}$	0.0	No additional weight for *Empty* action
κ	1.0	Temperature of $\pi_2(s)$ Eq.2.4

Table 3.1: Parameters to synthesize hidden $\pi_2((s_1, s_2))$.

The resulting policy is harder to visualize. In Figure 3.3, the black arrow magnitudes represent the probability of α_2 taking each action **given that α_1 is located in the blue cell**. Grey dots become larger and blacker as the probability of α_2 selecting the *Empty* action increases. The effect of the mobile kernel is obvious in comparison to Figure. 3.2. This realization of $\pi_2((s_1, s_2); \boldsymbol{\theta}^*)$ will be used as the benchmark for multi-agent inference experiments in both Section 3.4 and Chapter 4.

3.3 Policy Synthesis Comparison

In future experiments, α_1 will be tasked to reach the upper left (*North-West* grid cell, $s_1 = 0$, *without* landing in the same cell as α_2. The reward function for α_1 is

$$R(s, a_1) = \begin{cases} 1 & \text{if } s_1 == 0, \forall s_2 \in S_2 \ \&\ a_1 == Empty \\ 0 & \text{Otherwise} \end{cases}$$

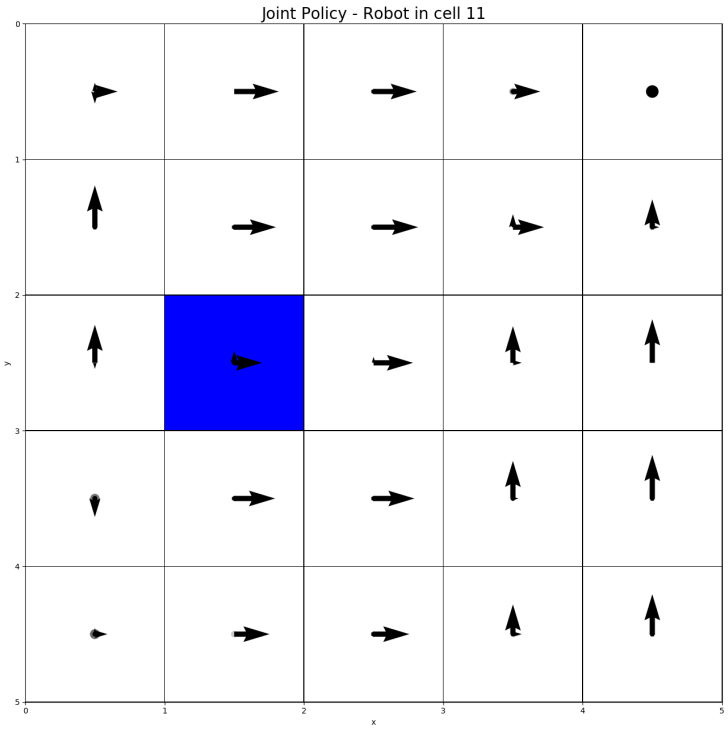

Figure 3.3: Hidden Policy of α_2 in an empty, deterministic, world with $s_1 = 11$.

In the case that $s_1 == s_2$, the transition probability of $\mathcal{M}$ is set to a self-loop under all actions.

To create an optimal policy benchmark for α_1, $\pi_1(s; \boldsymbol{\theta}^*)$, we'll solve Eq. 3.2 with VI and maximize Eq. 3.5 using $\pi_2((s_1, s_2); \boldsymbol{\theta}^*)$. The parameters necessary for EM and VI are listed in Table 3.2.

γ	0.9	Discount rate in $\mathcal{M}$
Z	100	EM-only: Number of M-step evaluations
H	15	EM-only: Trajectory Horizon Length

Table 3.2: Parameters to synthesize optimal $\pi_1((s_1, s_2); \boldsymbol{\theta}^*)$.

As a comparison, when *agent two* is in the middle cell, marked in blue, the optimal policy of α_1 to reach the green cell is shown in Figure 3.4. The agent is

clearly exploiting its knowledge of the true $\pi_2(s; \boldsymbol{\theta}^*)$ when $(s) = (13, 12)$ as that is normally a very dangerous action[1]. Figure 3.5 shows the approximately optimal $\pi_2(s; \boldsymbol{\theta}^*)$ from EM.

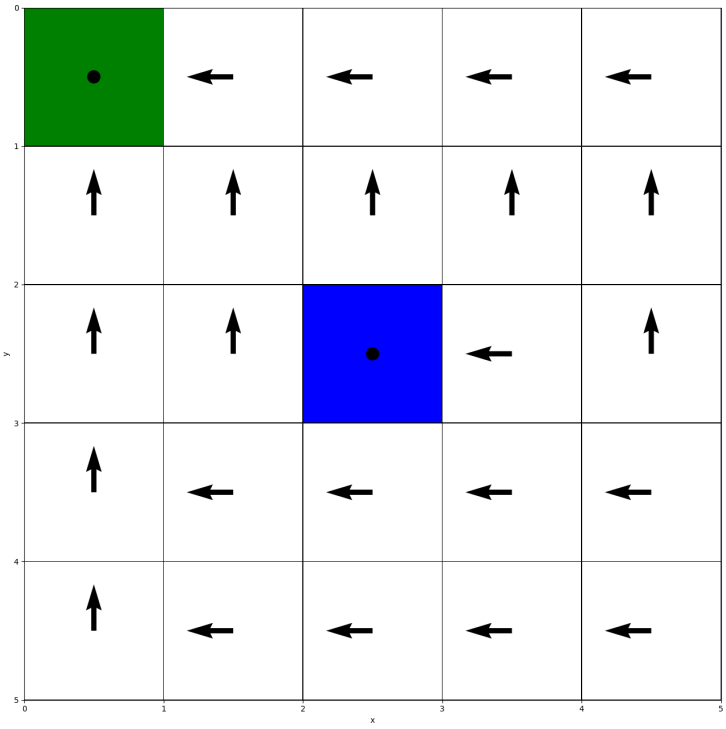

Figure 3.4: Optimal policy of α_1 from VI with $s_2 = 12$.

3.4 Multi Agent Inference Experiment

Using the observed data-set described by Table 3.3 we can perform similar experiments to those in Section 2.4. We'll also use the inference parameters in Table 3.4 that notes, among other things, that this experiment uses 25 fixed kernel locations

[1]Actually, a bug was recently discovered in the simulation engine written by your's truly. Without a memory variable in the MDP, the agent's are currently allowed to *swap* places without triggering a collision. The root of the problem is the definition of the transition probability matrix format and is not an easy fix. However, the policy of α_1 clearly exploits this when $(s) = (13, 12)$ (the blue cell is 12, the one to the right of it is cell 13).

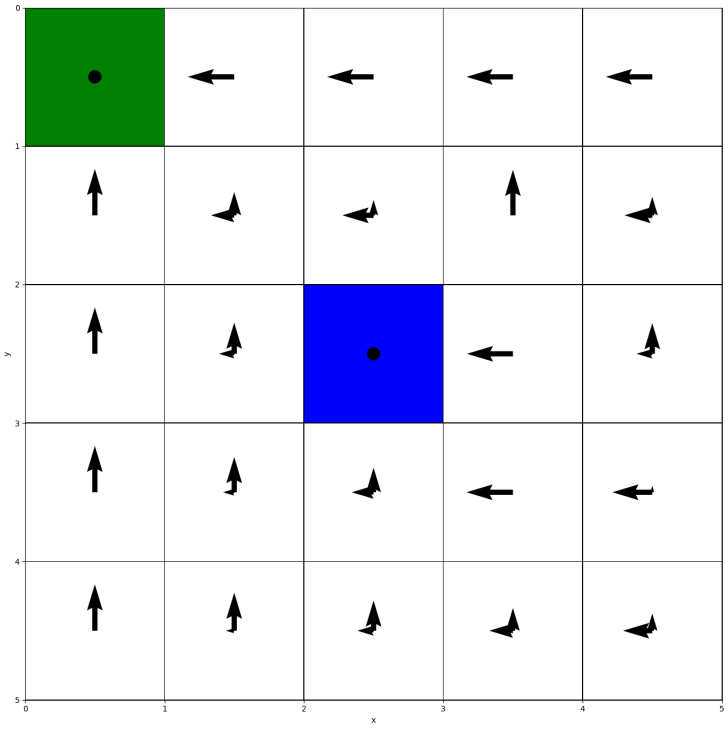

Figure 3.5: Approximately optimal policy of α_1 from EM with $s_2 = 12$.

and a mobile kernel.

The error metric plotted in Fig. 3.7 is now the fraction of the maximum possible L_1-norm. That is, for each iteration n,

$$Error(n) = \frac{||\pi_2, \hat{\pi}_2||_1}{2|S|}. \tag{3.6}$$

With this, we can represent the likelihood of $\hat{\pi}_2$ selecting different actions at each joint state s from the true π_2. For instance, if $Error(n) == 1$, then all actions sampled from $\hat{\pi}_2$ will be different from the true π_2 at every state. Recognize that there are infinitely many softmax policies that have $Error(n) == 1$, and a finite number of hardmax policies with this upper error value.

Given that the feature-space is much smaller than the joint state-action-space, $W = 26 \times 5 = K \times |A| \ll |S| \times |A| = 625 \times 5$, the inference result in Fig. 3.7 produced a reasonable policy estimate, $\hat{\pi}_2$.

$\|\tau\|$	10	Number of steps in a trajectory
$\|D\|$	3000	Number of trajectories observed
I_0	$\big(\mathcal{U}(0,24), \mathcal{U}(0,24)\big)$	Uniform distribution of $s^{(0)}$

Table 3.3: Observed data for multi agent inference.

σ_l	2.0, $\forall l$	Identical kernel standard-deviations for all mobile and fixed kernels
c_l	–	(Kernel Centers) Grid Cells $0, \dots 24 \cup s_1^{(t)}$
κ	0.5	Temperature of $\hat{\pi}_2$. Eq. (2.4)
λ	1e$-$5	Gradient update rate. Eq. (2.11)
η	0.0	Gradient velocity memory
m	2000	Per iteration sample size of $\boldsymbol{\theta} \sim \rho$
Λ	60	Moving average buffer length for $\mathsf{HIST}(\mathcal{L})$
ζ	0.001	Gradient ascent termination when $\Delta\mathsf{HIST}(\mathcal{L}) < \zeta$
μ_0	0.0	Initial parameter means
ν_0	1.0	Initial parameter standard-deviations
ν_{min}	0.2	Minimum parameter standard-deviation

Table 3.4: Hyper-parameters used for multi-agent inference.

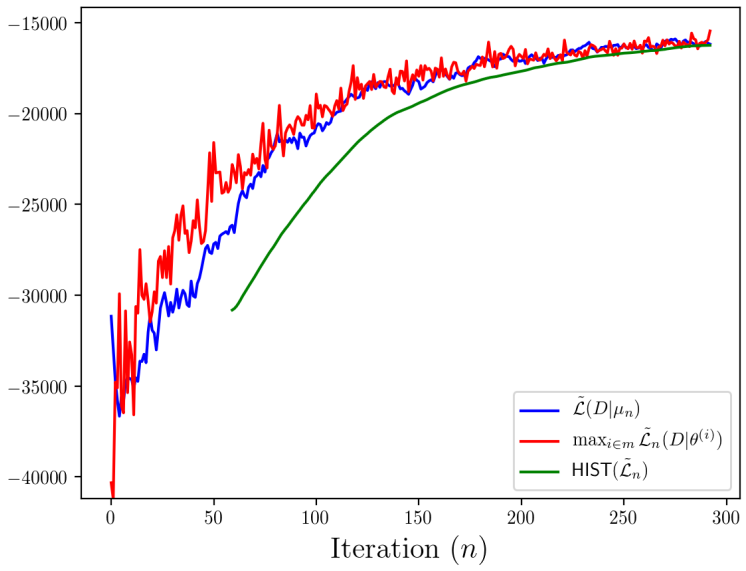

Figure 3.6: $\tilde{\mathcal{L}}(D|\rho)$ with parameters in Table 3.4.

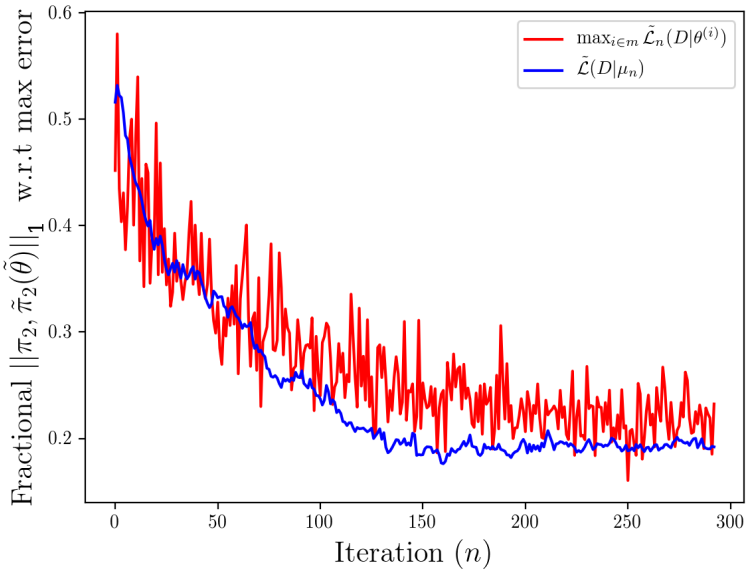

Figure 3.7: Fraction of maximum possible $||\pi_2, \hat{\pi}_2||_1$ (see Eq. 3.6) with parameters in Table 3.4.

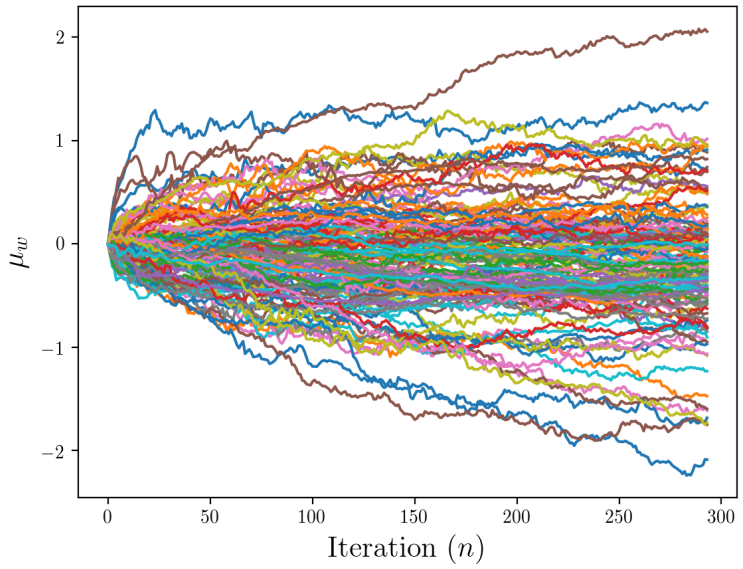

Figure 3.8: μ_w for each iteration with parameters in Table 3.4.

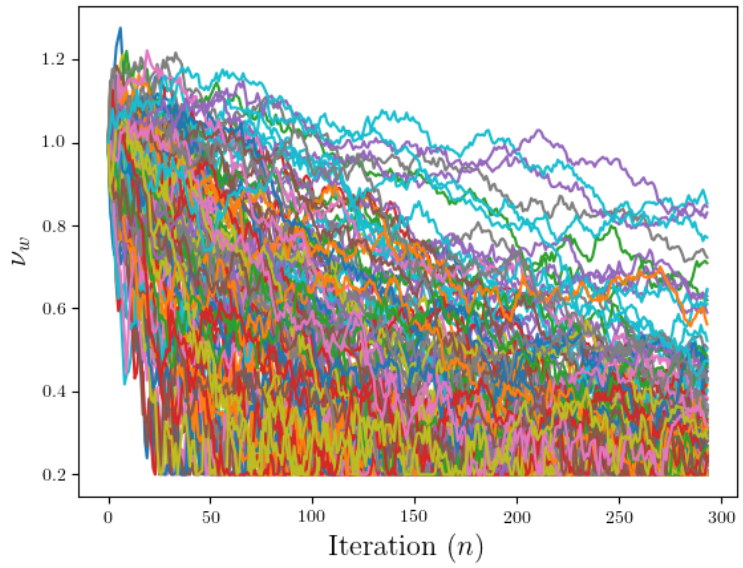

Figure 3.9: ν_w for each iteration with parameters in Table 3.4.

Chapter 4

Proactive Policy Inference

In the previous sections, an autonomous agent eventually infers a reasonable estimate of an uncontrollable agent's policy. By approximating the uncontrollable agent's Q-function as a linear combination of fixed and mobile features, the autonomous agent can eventually learn the hidden parameters in the transition model of a HiP-MDP, where the hidden parameter is the policy of the uncontrollable agent.

The successes in previous chapters have ignored a realistic constraint. Often, sampling a system is expensive, costing both time and wear-and-tear on mechatronic systems. This chapter will introduce active learning to enable an autonomous agent can adjust its policy to improve sample efficiency.

4.1 Proactive Inference

Assuming that α_1 has received some initial data $D^{(1)}$, the estimated policy of the uncontrollable agent can be improved from some initial guess $\hat{\pi}_2^{(0)}$ to form $\hat{\pi}_2^{(1)}$. Let's assume that $D^{(1)}$ was incomplete, i.e., it did not contain enough data for the following algorithm to meet the tolerance required by Assumption 2.3.5. Therefore, α_1 will need to gather more data, sets of trajectories, $D^{(b)},\ b = 1, \ldots, B$, to improve

its hypothesis of the other agent's policy. Like the DAGGER algorithm [RGB11], whenever a new dataset $D^{(b)}$ is received, the data used for inference is updated as $D \leftarrow D \cup D^{(b)}$. While DAGGER imitates the expert policy, and can request corrective actions from the expert during a roll-out, this implementation can only request another demonstration from α_2. Ideally, the requested $D^{(b)}$ will contain data that were lacking in all previous batches, because, when there are not enough data, some estimated parameter elements, $\hat{\theta}_w$, will be incorrect.

How can α_1 glean more informative data? To achieve proactive policy inference, α_1 needs to recognize what parts of $\hat{\pi}_2$ it thinks are known and unknown. Explicitly, after observing $D^{(b)}$, α_1 still might have very little knowledge about $\pi_2(x)$ – perhaps $s_2 = x$ has not yet been visited. This section will discuss how to distinguish known and unknown policy parameters, and how α_1 can proactively influence the future data, $D^{(b+1)}$.

Remark 4.1.1. *From this point on, the mean parameter vector inferred from any batch, $\boldsymbol{\mu}^{(b)}$, will be used to parameterize the agents policy, e.g., $\pi_2(s; \boldsymbol{\mu}^{(b)})$.*

4.1.1 Characterizing Unknown Parameters

By Eq. 2.2, the transition function is determined by the policies of both agents,

$$P((s_1', s_2')|(s_1, s_2), (a_1, a_2)) = T(s_1'|s_1, a_1)\pi_2(a_2|(s_1, s_2))\pi_1(a_1|(s_1, s_2)).$$

Since we've made Assumption 2.1.5 that π_2 is a stationary Markov policy, learning π_2 is equivalent to learning the dynamics of the MDP. Both problems have the same sample complexity, they are polynomial in the size of the joint state space and the action space of the α_2. With the Gaussian policy approximation from Section 2.3.1, if the number of parameters, W is much less than $|S| \times A$, then sample complexity

will be significantly reduced.

Definition 4.1.2. *Given two parameters $\epsilon, \varphi \in (0, 1)$, a parameter θ_w is (ϵ, φ)-known if, with probability $1 - \varphi$, the probability of the true value for θ_w is ϵ close to the mean of the Gaussian distribution $\mathcal{N}(\mu_w, \nu_w)$, i.e.,*

$$P(|\theta_w - \mu_w| \geq \epsilon) \leq \varphi.$$

If the estimated unknown policy parameter $\hat{\theta}_w$ is ϵ-close to the true (unknown) mean μ_w with probability $1 - \varphi$, then we could claim that enough knowledge has been obtained for θ_w and treat it as a *known* policy parameter. This claim is based on the Chernoff bound [KMT11].

Let the parameter elements, θ_w, form a set Θ. Then let Θ_{K} be the subset of known policy parameters and Θ_{UK} be the remaining unknown parameters. Next, we present a method to determine whether a state is unknown from the knowledge of these sub-sets of Θ.

First, given that each sampled parameter vector in Eq. 2.7, $\boldsymbol{\theta}^{(i)}$, is a Gaussian variable, the $\hat{Q}$-function is distributed as a multi-variate normal:

$$\hat{Q}(s, a; \boldsymbol{\theta}^{(i)}) \sim \sum_{w=1}^{W} \phi_w(s, a_2) \mathcal{N}\left(\mu_w, \nu_w^2\right),$$

The features, $\phi_w(s, a_2),\ w = 1, \ldots, W$ can be viewed as the mixing parameters.

A state s is known if and only if for any $a_2 \in A$, $Q(s, a_2)$ is known. Since the Q-value is a mixture of Gaussians, we'll use the mean and variance of $Q(s, a; \boldsymbol{\mu}^{(b)})$ returned from Eq. 2.11 to determine the distribution of a random variable – the estimate of $\pi_2(a|s)$.

If we were trying to solve a Probably-Approximately-Correct- (PAC)-MDP [FT14],

we would require an accuracy in learning the transition function. The model is considered to be correct if the model error $\bar{T}(s'|s,a_1) - T(s'|s,a_1) \leq \epsilon$ with probability $1-\varphi$, where $\bar{T}$ is the estimated transition function and T is the true transition function.

Based on the needed accuracy and the relation between T and π_2,

$$P(s'|s,a_1) = T(s'|s,(a_1,a_2))\pi_2(a_2|s),$$

we can show for any batch b:

$$\begin{aligned}\bar{P}(s'|s,a_1) - P(s'|s,a_1) &= T(s'|s,(a_1,a_2))\left(\pi_2(a_2|s) - \bar{\pi}_2(a_2|s;\boldsymbol{\mu})\right) \\ &\approx T(s'|s,(a_1,a_2)) \\ &\quad \cdot \Big(\exp(Q(s,a_2) - V(s)/\kappa) \\ &\qquad - \exp(Q(s,a_2;\boldsymbol{\mu}) - V(s;\boldsymbol{\mu})/\kappa)\Big)\end{aligned}$$

Since $Q(s,a_2;\boldsymbol{\mu})$ is a Gaussian, $\exp Q(s,a_2;\boldsymbol{\mu})$ is log-normal. Remember that to the first two moments, the sum of lognormal random variables can be approximated by a lognormal random variable [Fen60]. The approximated value function is the log-summation of log-normal random variables: $V = \kappa \log \sum \exp Q(s,a;\boldsymbol{\mu})/\kappa$. This can be approximated as a Gaussian. Therefore, we can quantify the bounded error between the estimated transition and true transition model, $\pi_2(a_2|s) - \bar{\pi}_2(a_2|s;\boldsymbol{\mu}^{(b)})$, by directly analyzing the distribution of the log-normal random variable $\exp(Q(s,a_2;\boldsymbol{\mu}) - V(s;\boldsymbol{\mu}))$.

We will use the linear combination of parameter-variances to quantify the vari-

ance of the Q-function,

$$\Omega(s, a_2) = \sum_{w=1 \in \Theta_{\mathsf{UK}}} \nu_w^2 \phi_w^2(s, a_2). \tag{4.1}$$

This can now be used as a metric for α_1 to either query, or explore, for the most informative data-batch $D^{(b)}$. We'll use a threshold to determine the assignment of parameters into the known and unknown parameter sets.

$$\theta_w \in \begin{cases} \Theta_{\mathsf{K}} & \text{if } \nu_w \leq \vartheta \\ \Theta_{\mathsf{UK}} & \text{Otherwise} \end{cases} \tag{4.2}$$

4.2 Single Agent Proactive Inference

In a scenario where α_1 simply watches α_2 but sample efficiency is still a concern, assume that α_1 can request α_2 to give a demonstration that starts from a state that has a high value of $\Omega(s, a_2)$. However, α_1 cannot directly request α_1 to start at state s and take a_2 because that might violate the policy of α_2.

Instead, after new data is observed $D^{(b)}$, we'll allow agent one to set the initial state-distribution of α_2, I_0. The initial state distribution will be proportional to the Q-function variance summed over all actions:

$$I_0^{(b)} = \frac{\sum_{a_2} \Omega(s, a_2)}{\sum_s \sum_{a_2} \Omega(s, a_2)} \tag{4.3}$$

Given a minimal demonstration, Fig. 4.1, the *next* initial state distribution that α_1 would request $s_2^{(1)}$ be drawn from looks like Fig. 4.2. Note that in this figure, the highest uncertainty is the cell 24, the *South-East* (lower-right) corner.

The algorithm for proactive inference in the single agent case, Algorithm 1, re-

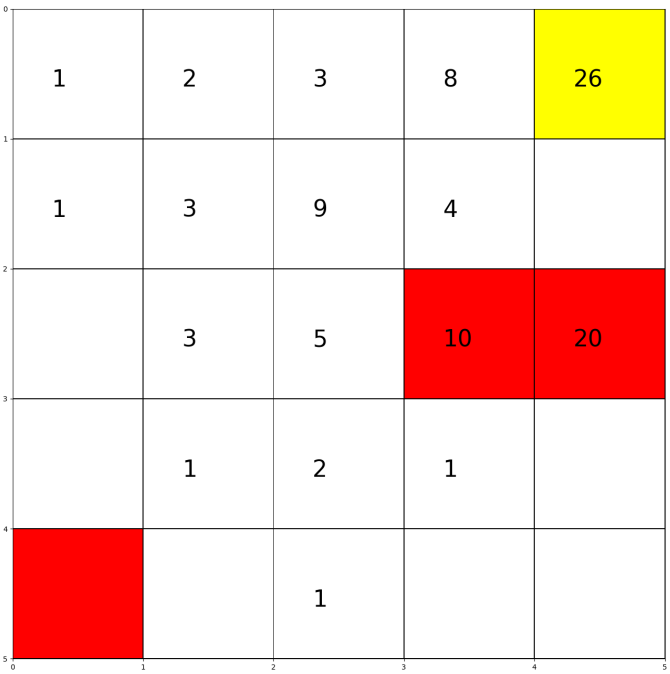

Figure 4.1: A set of roll-outs of π_2 that contains very few state-action samples.

quires the user to determine if do_active_update is set to True or False. If do_active_update = True then the initial distribution for the trajectories is set proportional to the policy uncertainty in Eq. 4.3. Otherwise, $s_2^{(b)} \sim \mathcal{U}(0, |S|)$.

Another note is that the gradient step size, λ, must now adapt as the data set size, $|D|$, grows; see Remark 2.3.7. Setting it to be inversely proportional to the nominal log-probability of the observed dataset, the constant term in Eq. 2.8 is often a good choice. If that constant term is zero, i.e., transitions are deterministic, the cardinality of the data-set can be used:

$$\begin{aligned} \lambda &= \frac{1}{-\Gamma\left[\sum_{t=0}^{|\tau_d|-1} \log P\left(s^{(t+1)} | (s, a_1, o_2)^{(t)}\right)\right]}, \text{ or} \\ &= \frac{1}{\Gamma(|D|+1)}, \end{aligned} \tag{4.4}$$

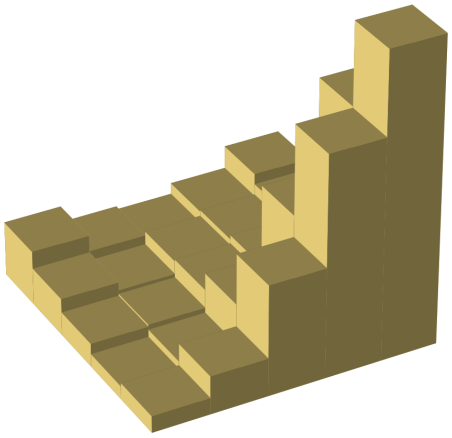

Figure 4.2: $P(s_2^{(0)}) \sim I_0$ as defined by Eq. 4.3

where $\Gamma = 10$ as a scaling parameter.

For simplicity, we'll define a *roll-out* procedure, Algorithm 2 which is equivalent to sampling a set of trajectories.

4.2.1 Experimental Results

Here, we'll show that we can accelerate the convergence of $\hat{\pi}_2^{(b)}$ as we observe more batches of data $D^{(b)}$. Similar to Section 2.4, the transition function is stochastic, see the motion model in Table 2.4.1. The true agent policy is again described by Fig. 2.1 where α_2 tries to avoid the red obstacles. The algorithm parameters are listed in Table 4.1 and inference parameters are listed in Table 4.2.

If do_active_update = True, we'll consider that trial of Alg. 1 to be *active*, otherwise, it was *passive* inference. Figure 4.3 shows that updating the initial distribution

Algorithm 1 Single agent mini-batch inference

Define Inference model $Y = (S_2, A_2, T, \boldsymbol{\phi}(s, a_2))$
Set do_active_update = (True|False)
$I_0^{(0)} = \mathcal{U}(0, |S|)$ ▷ Sample first data set uniformly.
$D = \emptyset$ ▷ Initialze dataset to be empty
$\boldsymbol{\mu}^{(0)} = \mathbf{0}$, $\boldsymbol{\nu}^{(0)} = \mathbf{1}$
for $b \in [1, B]$ **do**
    $D^{(b)} \leftarrow \mathsf{Rollout}(|D|, |\tau|, I_0^{(b)})$ ▷ Alg. 2
    $D \leftarrow D \cup D^{(b)}$
    $\lambda \leftarrow$ Eq. 4.4
    $\boldsymbol{\nu}^{(b)}, \hat{\pi}_2(s; \boldsymbol{\mu}^{(b)}) \leftarrow \mathsf{Infer}(D, Y, \boldsymbol{\mu}^{(b-1)}, \boldsymbol{\nu}^{(b-1)})$ ▷ Sect. 2.3
    if do_active_update **then**
        $I_0^{(b)} = \mathsf{Update}(\boldsymbol{\nu}^{(b)}, \boldsymbol{\phi}(s, a_2))$ ▷ Eq. 4.1 & 4.3
    end if
end for

Algorithm 2 Rollout

procedure ROLLOUT($|D|, |\tau|, I_0, \pi_1, \pi_2$)
    $D = \emptyset$
    for $d \in |D|$ **do**
        Sample $s_2^{(t=0)}$ from I_0
        $\tau_0 = s_2^{(0)}$
        for $t = 1$ *to* $|\tau_d|$ **do**
            Sample $s_2^{(t)}$ from $p(s_2^{(t)}|s_2^{(t-1)}, \pi_1, \pi_2)$ ▷ Eq. 2.2
            $\tau_t = s_2^{(t)}$
        end for
        $D = D \cup \tau_d$
    end for
    return D
end procedure

B	20	Number of batches (updates to $\hat{\pi}_2$)
$\|D\|$	2	Trajectories added per batch
$\|\tau\|$	4	Trajectory length

Table 4.1: Parameters for single-agent active inference (Alg. 1)

K	**7**	Number of kernels
σ_l	2.0, $\forall l$	Identical kernel standard-deviations
c_l	–	(Kernel Centers) Same as Fig. 2.8
κ	0.5	Temperature of $\hat{\pi}_2$. Eq. (2.4)
λ	–	See Eq. 4.4
η	0.2	Gradient velocity memory
m	1000	Per iteration sample size of $\boldsymbol{\theta} \sim \rho$
Λ	60	Moving average buffer length for $\mathsf{HIST}(\mathcal{L})$
ζ	0.001	Gradient ascent termination when $\Delta\mathsf{HIST}(\mathcal{L}) < \zeta$
μ_0	0.0	Initial parameter means
ν_0	1.0	Initial parameter standard-deviations
ν_{min}	0.4	Minimum parameter standard-deviation
ϑ	0.4	Threshold for inclusion in Θ_{K}

Table 4.2: Hyper-parameters used for multi-agent inference.

with Eq. 4.3 produces faster convergence to a minimum inference error. The solid lines represent the average $||\pi_2, \hat{\pi}_2||_1$ over 50 trials.

Ergodicity concerns

In the previous experiment, the trajectories were very short and only 2 were added to D to each batch. This frugal amount of data is what allowed the *active* update to I_0 to outperform the original uniform sampling. With the stochastic transition model in Table 2.4.1, any trajectory of appreciable length is likely to contain several "slips" to an unintended grid-cell. This, essentially, increases the available information in an observed data-set. This was previously mentioned in Section 3.2, we can now justify that claim.

Following Definition 1 in [HLZP17], the fundamental matrix of the Markov chain

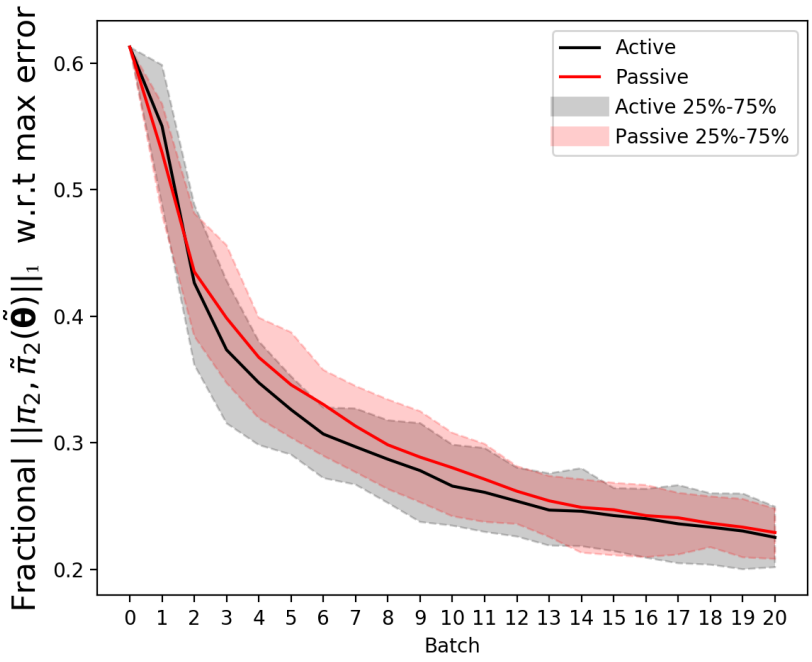

Figure 4.3: 50 trial average: Fraction of maximum possible $||\pi_2, \hat{\pi}_2||_1$ (see Eq. 3.6) with parameters in Table

$\mathcal{M}_{\pi_2}$ is:

$$Z = (T + \mathbf{1}\boldsymbol{\pi_2}^\intercal)^{-1},$$

where T is the transition probability matrix from Sec. 2.1, $\mathbf{1}$ represents a vector of ones, and $\boldsymbol{\pi_2}$ is the vector representation of the true policy of α_2. Now, Definition 2 in [HLZP17] defines the *ergodic coefficient* of a probability matrix (one with equal row sums) as:

$$e(Z) = \frac{1}{2} \max_{i,j} \sum_s |z_{is} - z_{js}|.$$

"*The ergodic coefficient of* $[\mathcal{M}_{\pi_2}]$ *indicates the sensitivity of [the] stationary distribution*"[HLZP17].

To show this assertion, we'll rerun the single-agent *active/passive* experiment with more data available as stated in Table 4.3.

B	20	Number of batches (updates to $\hat{\pi}_2$)
$\lvert D \rvert$	**10**	Trajectories added per batch
$\lvert \tau \rvert$	**6**	Trajectory length

Table 4.3: Parameters for longer single-agent active inference (Alg. 1)

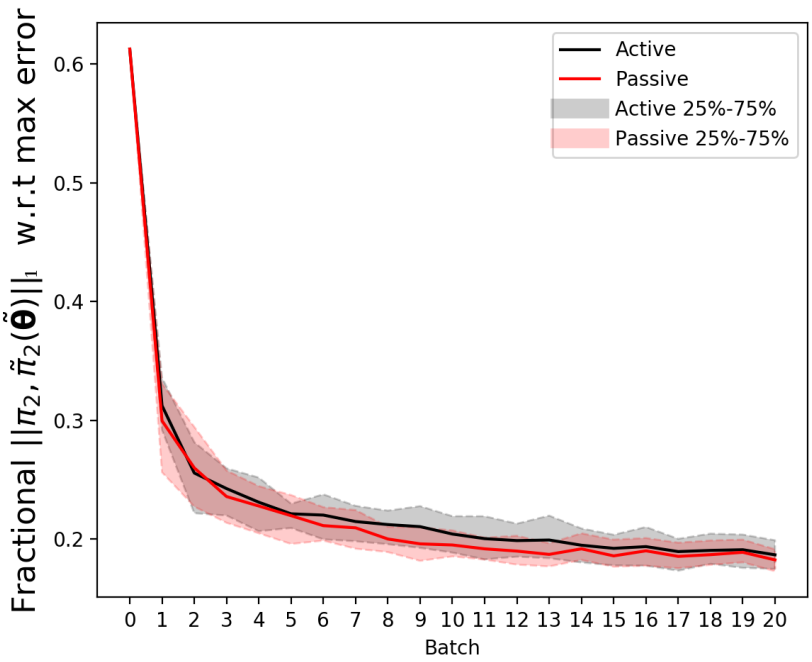

Figure 4.4: 50 trial average: Fraction of maximum possible $||\pi_2, \hat{\pi}_2||_1$ (see Eq. 3.6) with Algorithm parameters in Table 4.3.

In Figure 4.4, we still average over 50 trials of both *active*/*passive* flavors of Algorithm 1. The averages of the passive trial outperform the active algorithm. Uniformly sampling 10 initial states produces variability over the trajectories added to D in the b^{th} batch and adjusting the initial distribution with Eq. 4.3 produces less informative data in comparison.

Single-update algorithm

Here, we'll try to directly compare how active and passive inference improve the results from an initial data set. Specifically, by rolling out an initial set of data, inferring a policy, and storing the learned distribution values $\rho^{(1)}$, we can compare the resulting error of $\rho^{(2)}_{\text{active}}$ with $\rho^{(2)}_{\text{passive}}$[1]. This procedure is described in Algorithm 3. Running this algorithm 100 times shows that the active inference results in a 2% improvement over the passive update, see Figure 4.5. This experiment was run in an 8-by-8 grid world, see Figure A.1, and used the parameters in Table 4.4.

B	2	Number of batches (updates to $\hat{\pi}_2$) See Alg. 3.
$\lvert D\rvert$	**5**	Trajectories added per batch
$\lvert\tau\rvert$	**2**	Trajectory length (two states yield one observed action-outcome)

Table 4.4: Parameters for single single-agent active inference (Alg. 3)

Algorithm 3 Single agent active/passive single-update comparison

Define Inference model $Y = (S_2, A_2, T, \boldsymbol{\phi}(s, a_2))$
$I_0^{(0)} = \mathcal{U}(0, |S|)$ ▷ Sample first data set uniformly.
$D \leftarrow \mathsf{Rollout}(|D|, |\tau|, I_0^{(0)})$
$\lambda \leftarrow$ Eq. 4.4
$\boldsymbol{\mu}^{(0)} = \mathbf{0}$, $\boldsymbol{\nu}^{(0)} = \mathbf{1}$
$\boldsymbol{\nu}^{(1)}, \hat{\pi}_2(s; \boldsymbol{\mu}^{(1)}) \leftarrow \mathsf{Infer}(D, Y, \boldsymbol{\mu}^{(0)}, \boldsymbol{\nu}^{(0)})$ ▷ Shared initial inference results

$D^{(1)}_{\text{passive}} \leftarrow \mathsf{Rollout}(|D|, |\tau| I_0^{(0)})$ ▷ Rollout with original, uniform, I_0
$D_{\text{passive}} \leftarrow D \cup D^{(1)}_{\text{passive}}$
$\boldsymbol{\nu}^{(2)}_{\text{passive}}, \hat{\pi}_{\text{passive}}(s; \boldsymbol{\mu}^{(2)}_{\text{passive}}) \leftarrow \mathsf{Infer}(D_{\text{passive}}, Y, \boldsymbol{\mu}^{(1)}, \boldsymbol{\nu}^{(1)})$ ▷ Passive results

$I_{0-\text{active}} = \mathsf{Update}(\boldsymbol{\nu}^{(1)}, \boldsymbol{\phi}(s, a_2))$ ▷ Eq. 4.1 & 4.3
$D^{(1)}_{\text{active}} \leftarrow \mathsf{Rollout}(|D|, |\tau| I_{0-\text{active}})$ ▷ Rollout with new, active, I_0
$D_{\text{active}} \leftarrow D \cup D^{(1)}_{\text{active}}$
$\boldsymbol{\nu}^{(2)}_{\text{active}}, \hat{\pi}_{\text{active}}(s; \boldsymbol{\mu}^{(2)}_{\text{active}}) \leftarrow \mathsf{Infer}(D_{\text{active}}, Y, \boldsymbol{\mu}^{(1)}, \boldsymbol{\nu}^{(1)})$ ▷ Active results

[1] Remember $\rho = (\boldsymbol{\mu}, \boldsymbol{\nu})$.

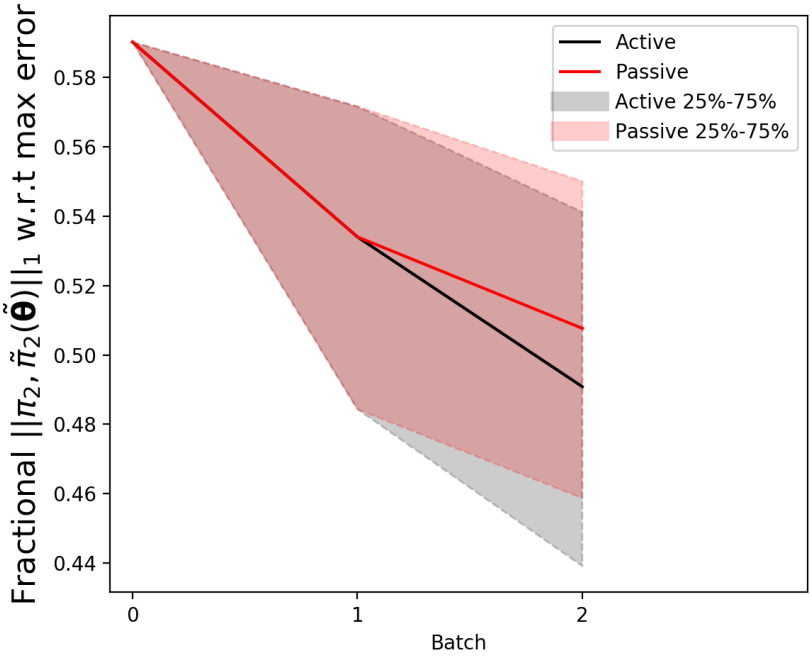

Figure 4.5: 100 trial average: Fraction of maximum possible $||\pi_2, \hat{\pi}_2||_1$ (see Eq. 3.6) with Algorithm parameters in Table 4.4.

Large world example

All previous examples have used a relatively uniform distribution of kernels in the state space, all having equal size, for inference. Given some initial idea about the hidden policy of α_2, a user could intentionally design a set of kernels that might be effective at inferring $\hat{\pi}_2$. Consider the large grid world, visitation count, and kernels described by figures A.2 and A.3 in Appendix A.2. The referenced figures in this section are included in the appendix due to the size required for adequate visualization.

Specifically, note the relative sizes of the kernels used and the approximate number of state-visits that fall within each kernels standard deviation, the blue circles. The inference results for a single stage are presented in Figure A.4. If the next set

of trajectories were to be sampled base on Equation 4.3, the probability of the first state would be proportional to the heat-map shown in Figure A.5, in which white represents the highest probability. Finally, the error between the true and estimated π_2 is in Figure A.6.

Effect of kernel location and size

Although the new initial distribution correctly identifies some areas of large policy error, e.g., near the red obstacle and upper left corner, there is also high uncertainty in the upper center of the grid where the error is relatively low. Compare figures A.5 and A.6 row 3, column 16.

Another interesting point as that the lower quadrant of the error plot, Figure A.6 rows 15-32 columns 0-16, contains 23% of the inference error; see Equation 3.6. This is the location of the largest kernel and where the true π_2 always took the northward action. There is enough cumulative error between the true and learned probability of the *North* action that this area would benefit from more samples. However, note that the policy error *at each state* is low, and the algorithm clearly marked this parameter with a low variance, ν_w^2. In this case, the uniform sampling (passive algorithm) will result in a slight improvement to the inferred policy in all regions, including the lower left quadrant. The active sampling will most likely only improve the policy in the uncertain regions, which cover a comparatively small area. For instance, compare the error and the uncertainty (figures A.5 and A.6) in cells near the block of red obstacles and the lower right corner.

Perhaps the error metric described in Equation 3.6 has obscured the benefit of the active inference procedure in this section. The active inference method will provide an improvement in the inferred policy in locations of high uncertainty, but these areas might not be large with respect to the entire state space. In future work,

analyzing the error in parameters that cover large areas of the state-space could be penalized less than errors in features that are placed at *interesting* locations.

4.3 Multi-agent Proactive Inference

Finally, we can address how α_1 can adjust its policy to glean the most information about π_2 in a dataset. This section will introduce a bonus reward used to help α_1 proactively explore the state-space. Like the experiments in Chapter 3, the environment has no obstacles and the transition function is deterministic.

4.3.1 Asymptotic Discount Optimal Policies

First, we'll justify why this algorithm allow α_1 to solve for the its optimal policy. In Chapter 2.5 of [HL12], Hernández-Lerma shows that an *adaptive* MDP will converge to the optimal value function if the parameter estimate converges to the true parameter. Remember Assumption 2.3.5; the true π_2 can be sufficiently represented with an optimal parameter. The author formally proves that if $\hat{\boldsymbol{\theta}}^{(b)} \to \boldsymbol{\theta}^*$ as $b \to \infty$, then $V(s; \hat{\boldsymbol{\theta}}^{(b)}) \to V^*(s; \boldsymbol{\theta}^*)$. In other words, if α_1 can eventually infer the policy of α_2, α_1 will then be able to plan an optimal policy to complete its task.

4.3.2 Multi-agent Algorithm

There are two different options for Algorithm 4. The *passive* implementation uses use_bonus_reward = False, while the *active* implementation uses use_bonus_reward = True.

Algorithm 4 Multi-agent mini-batch inference

Define HiP-MDP $\mathcal{M} = (S, A, T, \gamma, R)$
Define Inference model $Y = (S, A_2, T, \boldsymbol{\phi}(s, a_2))$
Set $\mathsf{use\_bonus\_reward} = (\mathsf{True}|\mathsf{False})$
Set I_0
$\hat{R}(s, a) = R$ ▷ Initial synthesis with true reward.
$D = \emptyset$
$\boldsymbol{\mu}^{(0)} = \mathbf{0}$, $\boldsymbol{\nu}^{(0)} = \mathbf{1}$
Set $\hat{\pi}_2^{(0)}(s)$ so $\mathsf{DIST}(A_2) \sim \mathcal{U}$
for $b \in [1, B]$ **do**
    $\pi_1^{(b)} \leftarrow \mathsf{Solve}(\mathcal{M}, \hat{\pi}_2^{(b-1)}(s), \hat{R}(s, a_1))$ ▷ Use EM from Sect. 3.1.1
    $D^{(b)} \leftarrow \mathsf{Rollout}(|D|, |\tau| I_0^{(b)}, \pi_1^{(b)}, \pi_2)$ ▷ Always roll out with *true* π_2.
    $D \leftarrow D \cup D^{(b)}$
    $\lambda \leftarrow$ Eq. 4.4
    $\boldsymbol{\nu}^{(b)}, \hat{\pi}_2(s; \boldsymbol{\mu}^{(b)}) \leftarrow \mathsf{Infer}(D, \mathcal{M}, Y, \boldsymbol{\mu}^{(b-1)}, \boldsymbol{\nu}^{(b-1)})$ ▷ Sect. 2.3
    if $\mathsf{use\_bonus\_reward}$ **then**
        $\hat{R}(s, a_1) = R(s, a_1) + \mathsf{Bonus}(\boldsymbol{\nu}^{(b)}, \boldsymbol{\phi}(s, a_2))$ ▷ Eq. 4.1 & 4.5
    end if
end for

Passive Implementation

In the passive implementation, α_1 simply updates $\pi_1^{(b)}$ by including the new inference $\hat{\pi}_2^{(b)}$; α_1 will simply exploit the information from $D^{(b-1)}$ so that $\pi_1^{(b)}(s; \hat{\boldsymbol{\theta}})$ maximizes the expected discounted reward, as shown in Section 3.3.

Active Implementation

The second option is for α_1 to *proactively* seek the most informative $D^{(b)}$. Gathering data about any unknown parameter elements will be given to α_1 as a bonus reward; α_1 proactively improves the inference of $\hat{\pi}_2$. Therefore, α_1 needs to determine which $\hat{\theta}_w$'s are unknown. This is discussed in Section 4.3.3; after each batch, a bonus reward is added to the original reward function R and used for policy synthesis.

4.3.3 Bonus Reward

What we would like to do is give α_1 a bonus reward whenever α_2 takes an action for which $\hat{Q}(s, a_2)$ has a large $\Omega(s, a_2)$, see Eq. 4.1. Since α_1 can not actually control α_2 to take an "uncertain" action a_2 at a joint state s, we build an exploration bonus reward at each joint state.

Using the original reward function of the robot from Def. 2.1.1 the reward including an exploration bonus is defined as

$$\hat{R}(s, a) = R(s, a) + \delta R(s)$$

and

$$\delta R(s) \propto \sum_{a_2 \in A} \sum_{\theta_w \in \Theta_{\mathsf{UK}}} \nu_w^2 \phi_w^2(s, a_2)$$

In practice, we can select a constant ε and define

$$\delta R(s) = \varepsilon \sum_{a_2 \in A} \sum_{\theta_w \in \Theta_{\mathsf{UK}}} \nu_w^2 \phi_w^2(s, a). \tag{4.5}$$

The constant ε has a role of weighting between exploration and exploitation.

The exploration bonus has the following property: When all parameters becomes known, the exploration bonus will become zero and the policy returns to the optimal policy with respect to the original reward function. By definition, for the same estimate of $\boldsymbol{\theta}$, and for any two pairs $(x, a_1), (y, a_2),\ \forall\ x, y \in S_2 | x \neq y$, the uncertainty in $Q(x, a_1)$ is greater than that of $Q(y, a_2)$ if $\sum_{a_2 \in A} \sum_{\theta_w \in \Theta_{\mathsf{UK}}} \nu_2^2 \phi_w^2(x, a_2) > \sum_{a_2 \in A} \sum_{\theta_w \in \Theta_{\mathsf{UK}}} \nu_w^2 \phi_w^2(y, a_2)$, and thus exploring any joint states $s = (s_1, x), \forall s_1 \in S_1$ will receive a higher exploration bonus.

4.3.4 Proactive Multi-agent Experiment

For this experiment, we're interested in a configuration where α_1 and α_2 are essentially forced to interact. The initial state for α_1 will always be the *South-East* (lower-left) corner, cell 24. Its goal is to reach the *North-West* (upper-left) corner, cell 0. Similar to the multi-agent experiment in Section 3.4, α_2 will be trying to reach the *North-East* (upper-right) corner, cell 4. In this experiment, α_2 will always start in the *South-West* (lower-left) corner, cell 20. The transition model is deterministic, actions succeed with probability 1. Also, a collision results in a sink-state. The transition to a sink-state is known to α_1, so it tries to avoid α_2 in general but the inclusion of the bonus-reward does affect that slightly.

The bonus-reward coerces α_1 to make sub-optimal actions so that an informative joint-state is reached, α_2 makes informative actions, and the $||\pi_2, \hat{\pi}_2||_1$ converges faster than with the passive approach; see Figure 4.6. The hyper parameters used for this experiment can be found in tables 3.2, 4.5, and 4.6.

B	30	Number of batches (updates to $\hat{\pi}_2$ and π_1)
$\|D\|$	10	Trajectories added per batch
$\|\tau\|$	10	Trajectory length

Table 4.5: Parameters for multi-agent active inference (Alg. 4)

4.3.5 Discussion

The proactive inference algorithm presents a way for an agent in a multi-agent environment to *guide* its exploration. In the example presented in the previous section, using the weighted parameter-variance combined with squared-feature values leads to proactive planning. Agent-1 is able to improve its inference of $\hat{\pi}_2$, however, in the demonstrated example the fraction of batches in which α_1 reached its goal dropped relative to the greedy (passive) policy synthesis; see Figures 4.6 and 4.7.

K	**9**	Number of kernels
σ_l	2.0, $\forall l$	Identical kernel standard-deviations
c_l	$[0:4:24]$	Kernel Centers every 4^{th} cell (forms an "X")
κ	0.5	Temperature of $\hat{\pi}_2$. Eq. (2.4)
$\mathbf{I_0}$	$(20, 24)$	Constant initial state for all trajectories.
λ	–	See Eq. 4.4
η	0.2	Gradient velocity memory
m	1000	Per iteration sample size of $\boldsymbol{\theta} \sim \rho$
Λ	60	Moving average buffer length for $\mathsf{HIST}(\mathcal{L})$
ζ	0.001	Gradient ascent termination when $\Delta\mathsf{HIST}(\mathcal{L}) < \zeta$
μ_0	0.0	Initial parameter means
ν_0	1.0	Initial parameter standard-deviations
ν_{min}	0.4	Minimum parameter standard-deviation
ϑ	**0.8**	Threshold for inclusion in Θ_{K}

Table 4.6: Hyper-parameters used for multi-agent inference.

The trade-off between the faster inference and count of batches in which α_1 reaches its goal is a function of both the bonus-reward weight, ε in Eq. 4.5, as well as the threshold for the unknown parameter set, ϑ in Eq. 4.2. By increasing ε, α_1 will visit states with high parameter uncertainty more and possibly never reach the original goal location. Subsequently, decreasing epsilon to zero recovers the original reward function, which will not provide an improvement in the inference results. If no parameter variances are above the value chosen for ϑ, the original reward function is recovered.

For this proactive planning algorithm to yield more rewards then a passive plan, there must be something *worth* learning. If this was the case, proactive exploration would be the only, or faster, way to learn a critical subset of π_2. An interesting experiment for future work would be to augment the policy of α_2 to be aggressive, where it would try to collide with α_1. Something "*worth*" learning might be a subset of the state-space where α_2 gets stuck; add a set of parameters to Table 3.1 such that $P(a_2 == Empty|s_2) = 1,\ \forall s_1 \in S$.

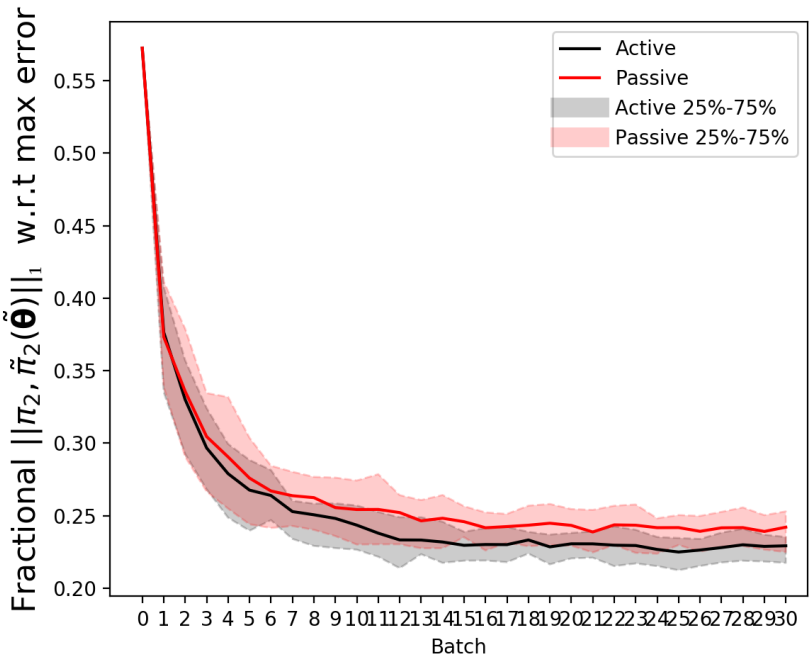

Figure 4.6: 50 trial average: Fraction of maximum possible $||\pi_2, \hat{\pi}_2||_1$ (see Eq. 3.6) with Algorithm parameters in Table 4.5.

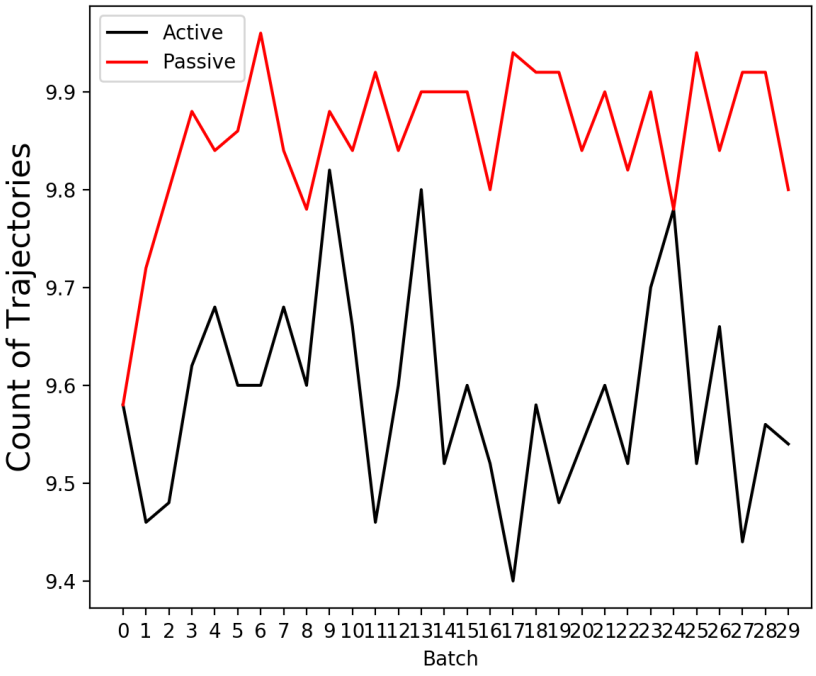

Figure 4.7: 50 trial average of trajectories where α_1 reaches cell=0 (goal) with Algorithm parameters in Table 4.5.

Chapter 5

Conclusion

This thesis has successfully designed a new parameter-based policy inference algorithm that can be used in model-based RL. We applied policy-gradient techniques to maximize the likelihood of observed state-sequences from an uncontrollable agent with an unknown policy, given a parametric estimate of that policy. By assuming that the parameter vector is distributed as a multi-variate normal distribution, the policy inference algorithm captures the parameter variance, which we show can quantify the uncertainty in the estimated Q-function of the uncontrollable agent. This report approximated the Q-function as a linear combination of parameters.

Using the parametric uncertainty, we also provided two platforms for interacting with an uncontrollable agent. First, if we're allowed to ask for policy demonstrations, trajectories, from the uncontrollable agent and select the initial state of the trajectory, we can actively seek to aggregate more informative data and improve the sample efficiency of the inference. We select the the initial state of a new trajectory from a with probability proportional to the total estimated variance of the Q-function at a state. This was applied to a single-agent scenario where we learned the agent's policy from demonstrations. We discussed that this active-re-sampling

can be used to provide more insight into features that have a high parameter uncertainty.

The single-agent active inference algorithm can accelerate the policy inference, but is highly dependent on the feature selection. To automatically determine pertinent features, future work could initialize the algorithm with many redundant features, and then remove the features that the inference algorithm determines to have a parameter mean of zero with high confidence. Alternatively, on-line hyperparameter optimization could help adjust feature locations and size to better estimate the demonstrated policy

Second, in a multi-agent experiment, we defined a new bonus-reward that is a function of the estimated variance of the uncontrollable agent's Q-function. This bonus reward convinced a controllable agent to choose actions that led to informative interactions and accelerated the policy inference of the uncontrollable agent. Future extensions of this work also include a joint planning and active learning algorithm. That algorithm would lead to the controllable agent eventually exploiting the actively-learned information so that it can plan an optimal policy as soon as possible.

Appendix A

A.1 8-by-8 Grid World

This figure shows an example initial data set, $D^{(0)}$ used in line 3 of Algorithm 3.

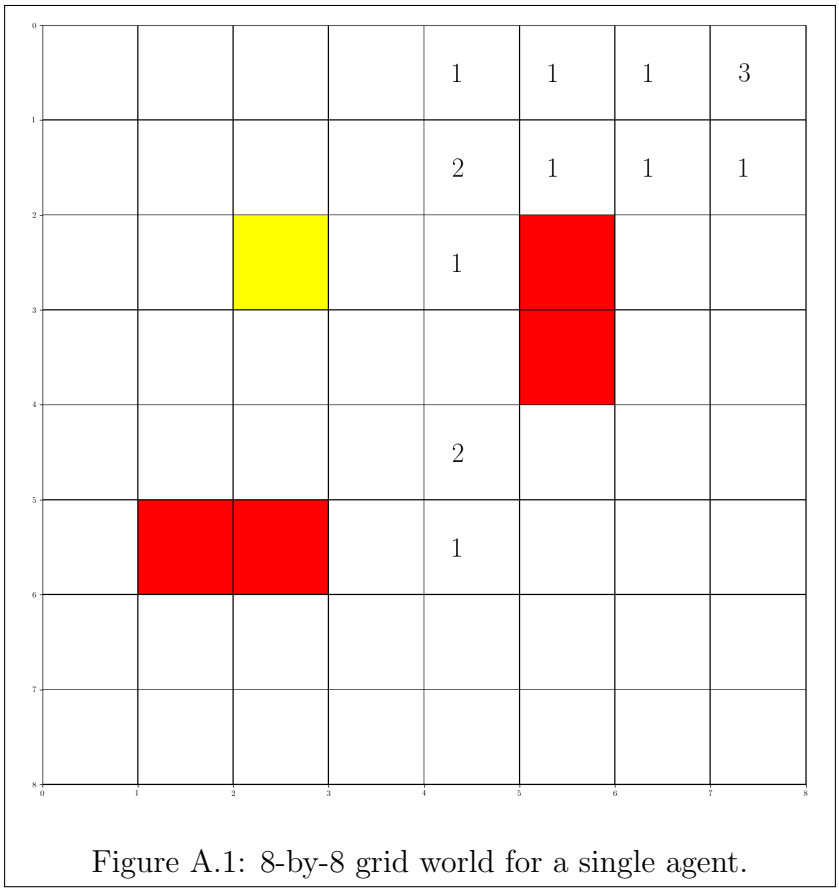

Figure A.1: 8-by-8 grid world for a single agent.

A.2 32-by-32 Grid World

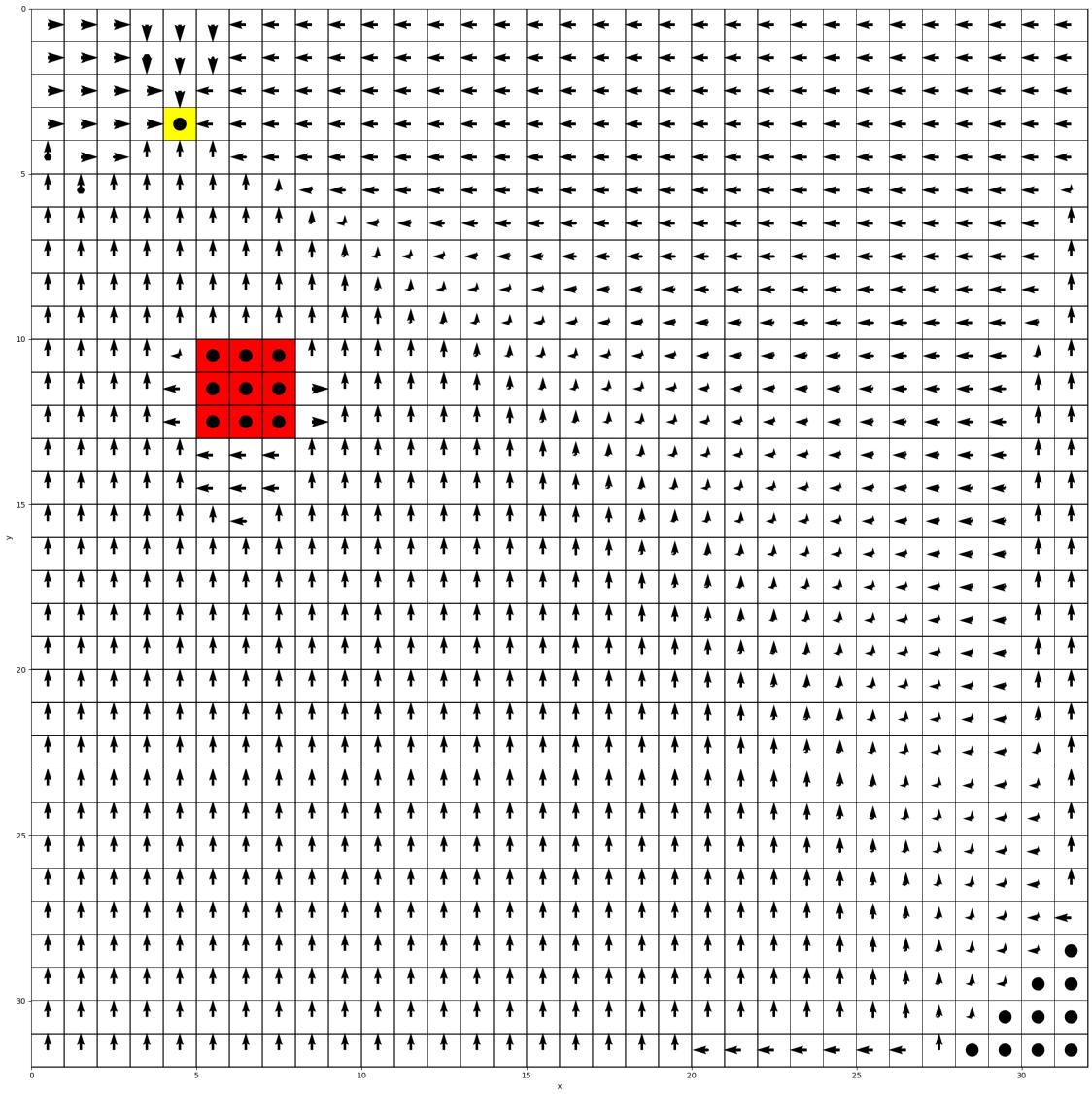

Figure A.2: True policy of α_2 in a single agent environment. Grid size is 32-by-32. Arrow sizes are proportional to probability of taking the action in each direction. Dots represent the stay-action.

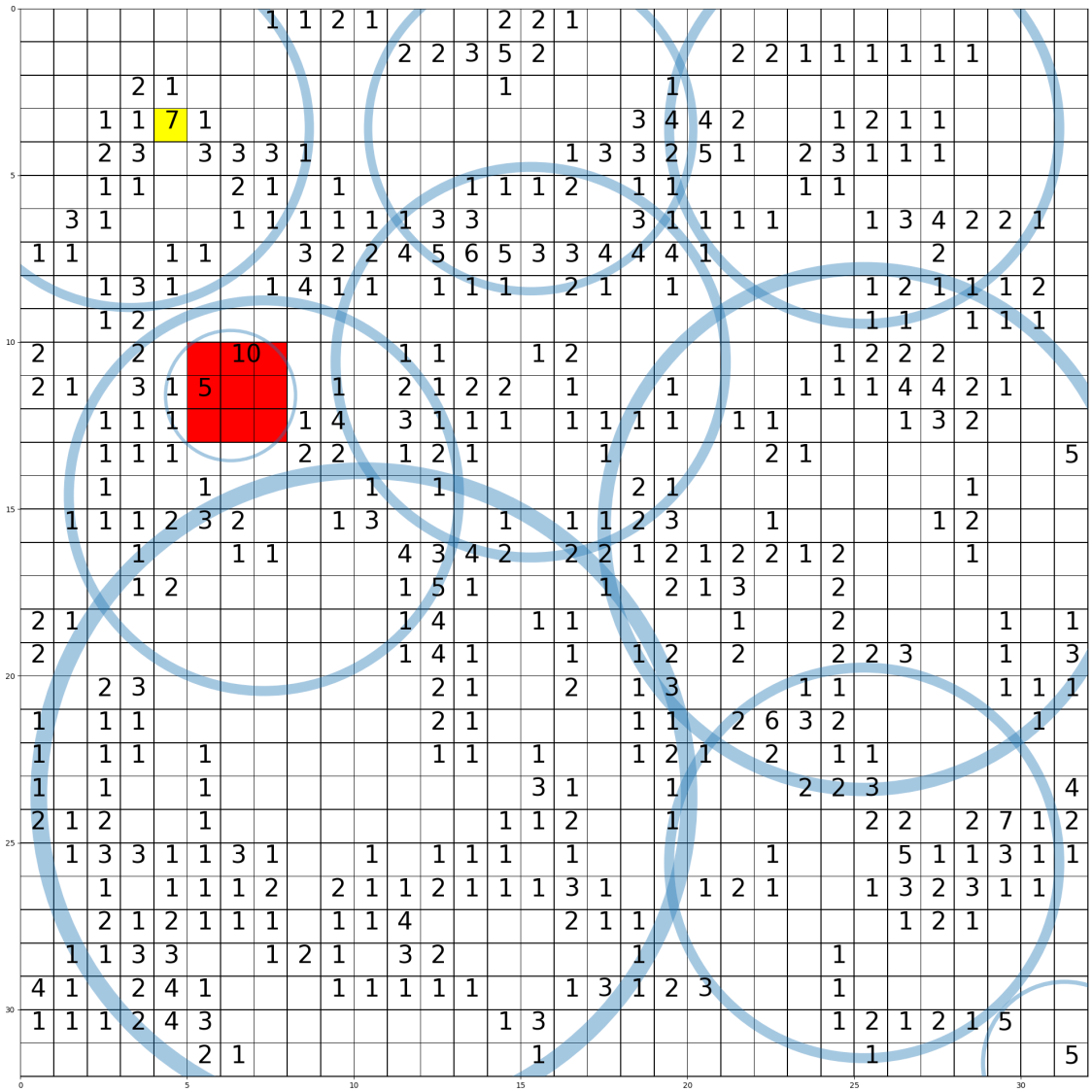

Figure A.3: Visitation count given 150 trajectories with 5 time steps using the true policy of α_2 (Fig. A.2) in a single agent environment. Grid size is 32-by-32. Blue circles represent the standard deviation of kernels used.

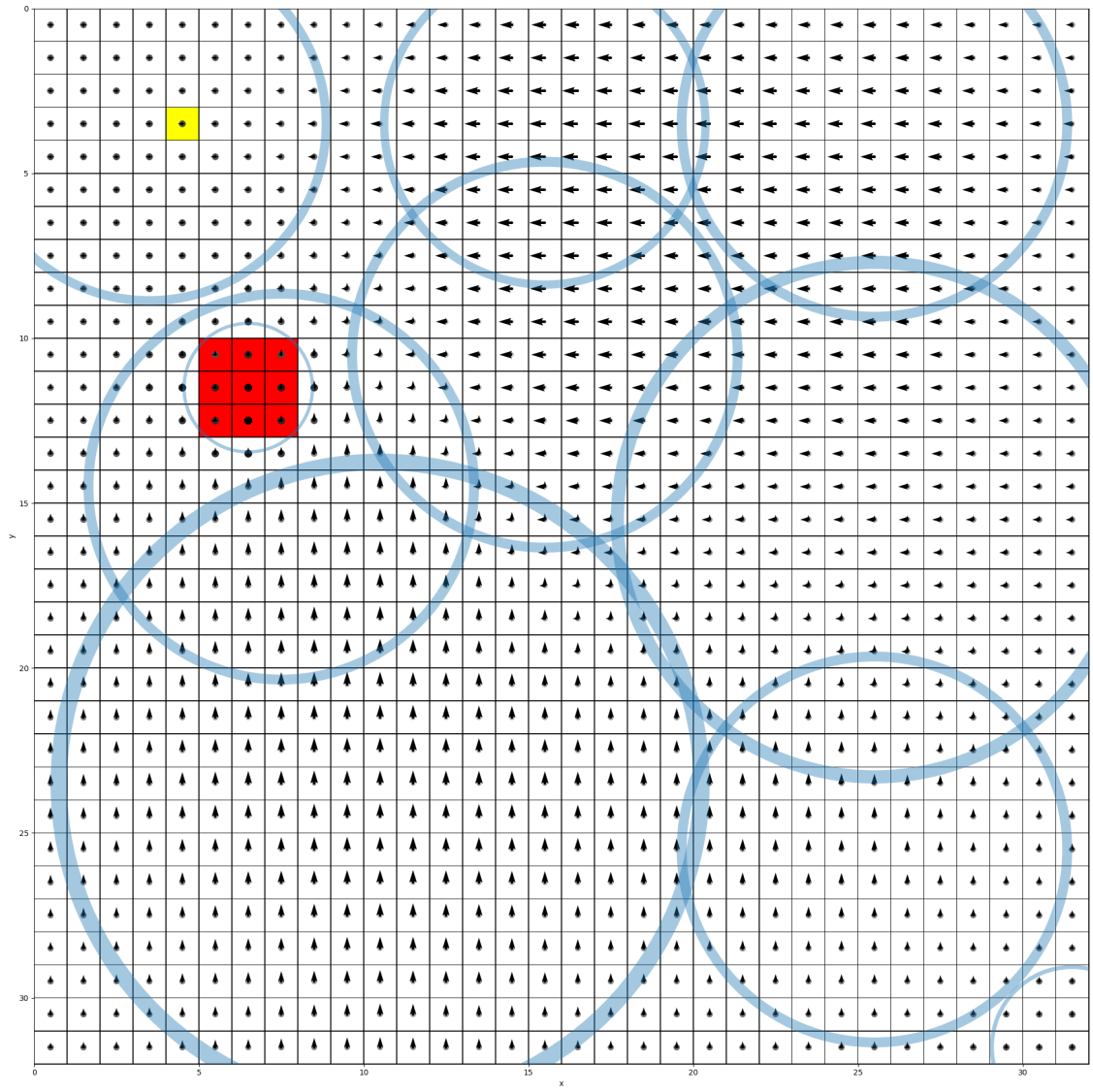

Figure A.4: Inferred policy of α_2 in a single agent environment given visitation count (Fig. A.3) in a single agent environment. Grid size is 32-by-32. Arrow sizes are proportional to probability of taking the action in each direction.

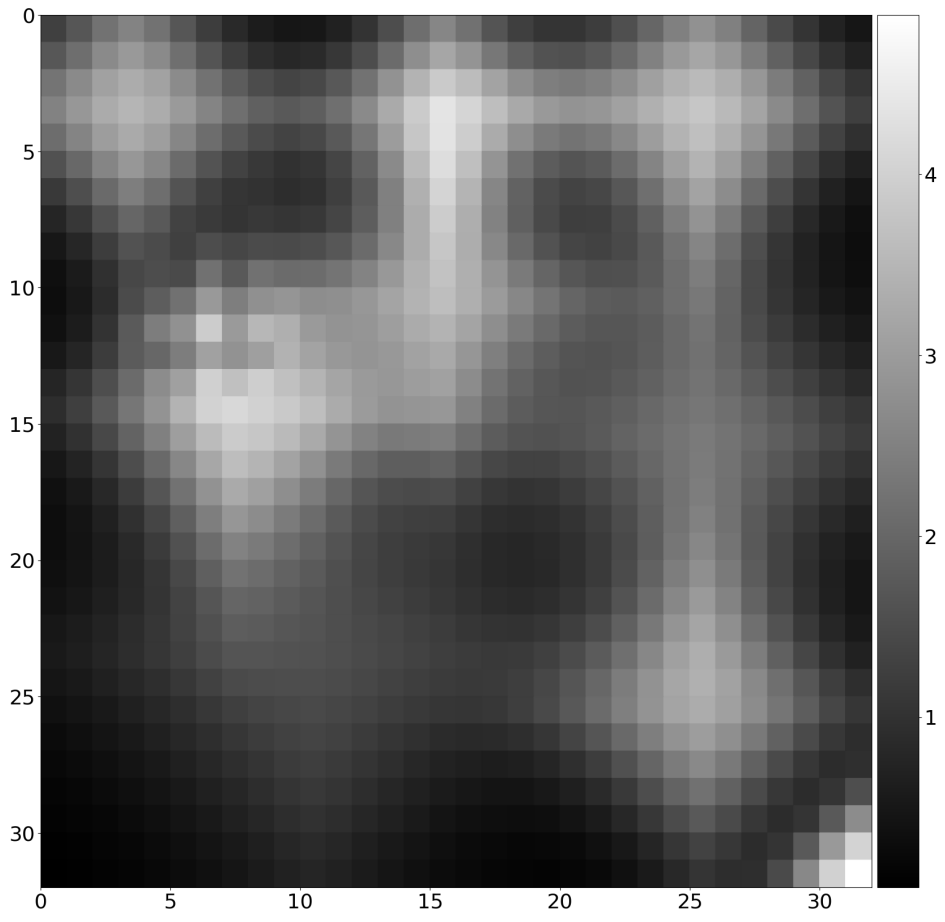

Figure A.5: The numerator of Eq. 4.3, $\sum_{a_2} \Omega(s, a_2)$, is proportional to the probability of initial state being selected after inference results in Fig. A.4. This heat-map is an alternative visualization of the 3D bar-plot like Fig. 4.2

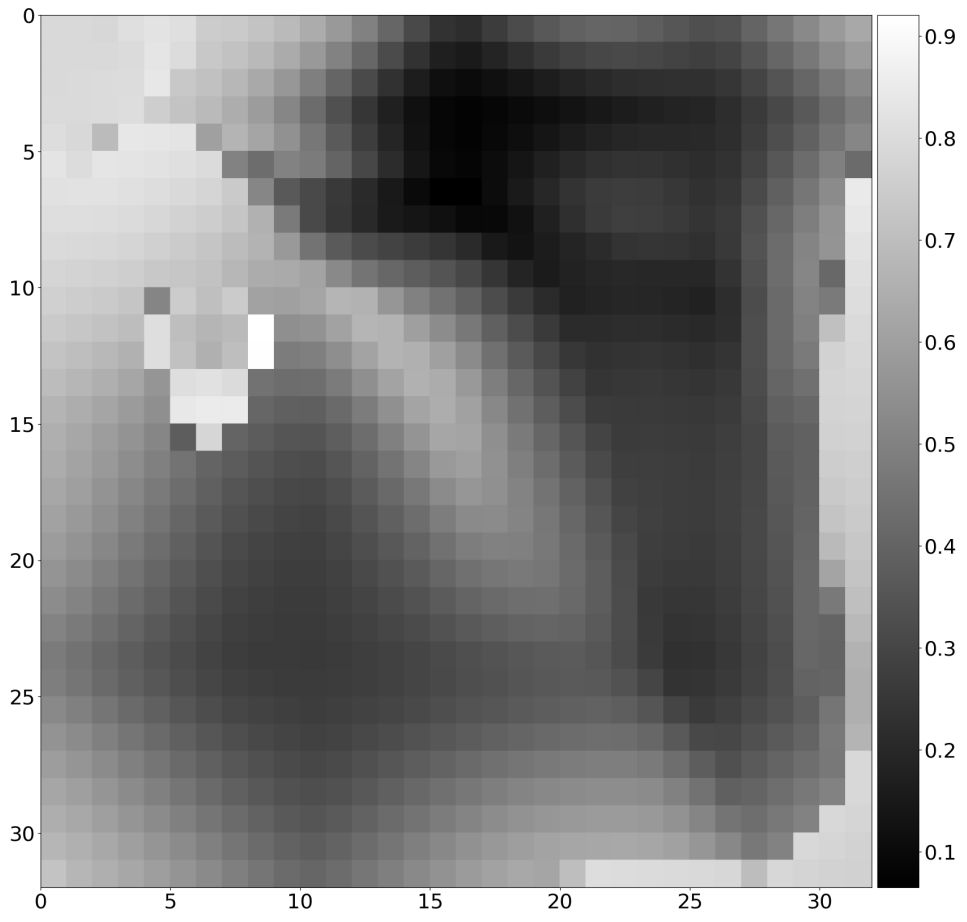

Figure A.6: The true error, $||\pi_2, \hat{\pi}_2||_1$, after inference results in Fig. A.4

Bibliography

[AHS10] Takayuki Akiyama, Hirotaka Hachiya, and Masashi Sugiyama. Efficient exploration through active learning for value function approximation in reinforcement learning. *Neural Networks*, 23(5):639–648, 2010.

[AK17] Garrett Andersen and George Konidaris. Active exploration for learning symbolic representations. In *Advances in Neural Information Processing Systems*, pages 5016–5026, 2017.

[AWD17] Olov Andersson, Mariusz Wzorek, and Patrick Doherty. Deep learning quadcopter control via risk-aware active learning. In *AAAI*, pages 3812–3818, 2017.

[BT02] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[BWF+13] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

[BWH12] Luca F Bertuccelli, Albert Wu, and Jonathan P How. Robust adaptive markov decision processes: Planning with model uncertainty. *IEEE Control Systems*, 32(5):96–109, 2012.

[CLP17] S. P. Chinchali, S. C. Livingston, and M. Pavone. Multi-objective optimal control for proactive decision-making with temporal logic models. In *Int. Symp. on Robotics Research*, dec 2017.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[DR11] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th In-*

ternational Conference on machine learning (ICML-11), pages 465–472, 2011.

[DVK16] Finale Doshi-Velez and George Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, page 1432. NIH Public Access, 2016.

[Fen60] Lawrence Fenton. The sum of log-normal probability distributions in scatter transmission systems. *IRE Transactions on Communications Systems*, 8(1):57–67, 1960.

[FSM12] Karl Friston, Spyridon Samothrakis, and Read Montague. Active inference and agency: optimal control without cost functions. *Biological cybernetics*, 106(8-9):523–541, 2012.

[FT14] Jie Fu and Ufuk Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[GHL09] Xianping Guo and Onésimo Hernández-Lerma. *Continuous-Time Markov Decision Processes*, pages 9–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[HGW+16] Michael Herman, Tobias Gindele, Jörg Wagner, Felix Schmitt, and Wolfram Burgard. Inverse reinforcement learning with simultaneous estimation of rewards and dynamics. In *Artificial Intelligence and Statistics*, pages 102–110, 2016.

[HL12] Onésimo Hernández-Lerma. *Adaptive Markov control processes*, volume 79. Springer Science & Business Media, 2012.

[HLZP17] Manjesh K. Hanawal, Hao Liu, Henghui Zhu, and Ioannis Ch Paschalidis. Learning policies for markov decision processes from data. 01/2017.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KLC98] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[KMT11] Hisashi Kobayashi, Brian L Mark, and William Turin. *Probability, random processes, and statistical analysis: applications to communications, signal processing, queueing theory and mathematical finance*. Cambridge University Press, 2011.

[KT00] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[KVTT17] Mehdi Khamassi, George Velentzas, Theodore Tsitsimis, and Costas Tzafestas. Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task. In *Robotic Computing (IRC), IEEE International Conference on*, pages 28–35. IEEE, 2017.

[LXM13] Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 701–709, 2013.

[MCdFDC07] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, volume 3, pages 334–341, 2007.

[NNXS17] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2772–2782, 2017.

[PN17] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[PS08] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[SO15] Masashi Sugiyama and Safari Books Online. *Statistical reinforcement learning: modern machine learning approaches*. CRC Press, Taylor & Francis Group, Boca Raton, FL, 1 edition, 2015.

[SOR+10] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[TMZ+14] Voot Tangkaratt, Syogo Mori, Tingting Zhao, Jun Morimoto, and Masashi Sugiyama. Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. *Neural networks*, 57:128–140, 2014.

[TSH10] Marc Toussaint, Amos Storkey, and Stefan Harmeling. Expectation-maximization methods for solving (po) mdps and optimal control problems. *Inference and Learning in Dynamic Models*, 2010.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.