Mitigating Partial Observability in Sequential Decision Processes via the Lambda Discrepancy

Cameron Allen[*] UC Berkeley[†]
Aaron Kirtland[*] Brown University
Ruo Yu Tao[*] Brown University
Sam Lobel Brown University
Daniel Scott Georgia Tech
Nicholas Petrocelli Brown University
Omer Gottesman Amazon[‡]
Ronald Parr Duke University
Michael L. Littman Brown University
George Konidaris Brown University

Abstract

Reinforcement learning algorithms typically rely on the assumption that the environment dynamics and value function can be expressed in terms of a Markovian state representation. However, when state information is only partially observable, how can an agent learn such a state representation, and how can it detect when it has found one? We introduce a metric that can accomplish both objectives, without requiring access to—or knowledge of—an underlying, unobservable state space. Our metric, the λ-discrepancy, is the difference between two distinct temporal difference (TD) value estimates, each computed using TD(λ) with a different value of λ. Since TD(λ=0) makes an implicit Markov assumption and TD(λ=1) does not, a discrepancy between these estimates is a potential indicator of a non-Markovian state representation. Indeed, we prove that the λ-discrepancy is exactly zero for all Markov decision processes and almost always non-zero for a broad class of partially observable environments. We also demonstrate empirically that, once detected, minimizing the λ-discrepancy can help with learning a memory function to mitigate the corresponding partial observability. We then train a reinforcement learning agent that simultaneously constructs two recurrent value networks with different λ parameters and minimizes the difference between them as an auxiliary loss. The approach scales to challenging partially observable domains, where the resulting agent frequently performs significantly better (and never performs worse) than a baseline recurrent agent with only a single value network.

1 Introduction

The dominant modeling frameworks for reinforcement learning [Sutton and Barto, 2018] define environments in terms of an underlying Markovian state representation. This modeling choice, called the *Markov assumption*, is nearly ubiquitous in reinforcement learning, because it allows environment dynamics, rewards, value functions, and policies all to be expressed as functions that are independent of the past given the most recent state. In principle, an environment can be modeled as either a Markov decision process (MDP) [Puterman, 1994], or its *partially observable* counterpart, a POMDP

[*]These authors contributed equally and are ordered alphabetically. Please send any correspondence to `camallen@berkeley.edu`, `aaron_kirtland@brown.edu`, and `ruoyutao@brown.edu`.

[†]Some of this work was completed while at Brown University.

[‡]Work completed while outside of Amazon.

ICML 2024 Workshop on Foundations of Reinforcement Learning and Control.

[Kaelbling et al., 1998], as long as the underlying state representation contains enough information to satisfy the Markov assumption. The POMDP framework is more general, but POMDPs are typically much harder to solve than MDPs [Zhang et al., 2012], so it is important to know when it is appropriate to use the simpler MDP framework.

Ideally, a system designer can ensure that their reinforcement learning agent is configured to use the appropriate problem model. If an environment is partially observable, the designer may manually add relevant decision-making features to the agent's state representation, either by concatenating observations together [Mnih et al., 2015] or by using other types of feature engineering [Bellemare et al., 2020, Galataud, 2021, Tao et al., 2023]. Alternatively, the designer can manually specify a set of possible environment states over which the agent can maintain a Markovian belief distribution [Kaelbling et al., 1998]. The challenge is that it is not always obvious when the designer has supplied sufficient information to satisfy the Markov assumption. These approaches require the person deploying the agent to know details about the very problem the agent is supposed to help them solve.

An alternative approach—that we explore in this paper—is to let the agent *learn* a good state representation for the problem it is solving. The conventional deep-learning-era wisdom is that the best problem representations come from training a system to solve a task, rather than from human designers. When faced with a potentially partially observable environment, if we provide the agent with a large enough recurrent neural network (RNN), it can perhaps discover an internal state representation "end-to-end" that maximizes reward via gradient descent [Bakker, 2001, Hausknecht and Stone, 2015, Ni et al., 2022a, Dong et al., 2022]. Indeed, end-to-end RNN architectures work well for many problems and are both general-purpose and scalable. However, these techniques only implicitly address the problem of learning a Markovian state representation; in this paper, we show that we can achieve much better learning performance when we explicitly tackle the problem of partial observability.

In our approach, the agent learns a state representation by directly minimizing a metric that assesses whether the environment is fully or partially observable. We call our metric the λ-*discrepancy*, and define it as the difference between two distinct value functions estimated using temporal difference learning (TD) [Sutton, 1988]. Specifically, the TD(λ) method defines a smooth trade-off between one-step TD (i.e., $\lambda = 0$), which makes an implicit Markov assumption, and Monte Carlo (MC) estimation ($\lambda = 1$), which does not, and intermediate λ values interpolate between these extremes. By comparing value estimates for two distinct values of λ, we can check that the agent's observations support Markovian value prediction, and augment them with memory if we find they are incomplete.

Our main contributions[4] are as follows:

1. We introduce and formally define the λ-discrepancy and prove that it is exactly zero for MDPs and almost always non-zero for a broad class of POMDPs that we characterize. This analysis tells us that our metric reliably *detects* partial observability.

2. We then consider a tabular, proof-of-concept experiment that adjusts the parameters of a memory function to minimize the λ-discrepancy via gradient descent. For this experiment, we compute the λ-discrepancy in closed form. Our results demonstrate that minimizing the λ-discrepancy is a viable path to *mitigating* partial observability.

3. Finally, we integrate our approach into a deep reinforcement learning agent and evaluate on a set of large and challenging POMDP benchmarks. We find that minimizing the λ-discrepancy between two learnt value functions, as an auxiliary loss alongside traditional reinforcement learning, is often significantly more effective (and never worse) than training the same agent with just a single value function.

Overall, we find that the λ-discrepancy is a simple yet powerful metric for detecting and mitigating partial observability. The metric is also practical, since it can be computed directly from value functions, which are ubiquitous in reinforcement learning. Furthermore, such value functions need only consider observable parts of the environment, so the λ-discrepancy remains applicable even without the common assumption that the agent knows the full set of possible POMDP states.

[4] Code: `https://github.com/brownirl/lambda_discrepancy`
Videos: `https://lambda-discrepancy.github.io`

2 Background

We consider two frameworks for modeling sequential decision processes: MDPs and POMDPs. The MDP framework [Puterman, 1994] consists of a state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ mapping to a distribution over states, discount factor $\gamma \in [0, 1]$, and initial state distribution $p_0 \in \Delta\mathcal{S}$. The agent's goal is to find a policy $\pi_\mathcal{S} : \mathcal{S} \to \Delta\mathcal{A}$ that selects actions to maximize *return*, g_t, the discounted sum of future rewards starting from timestep t: $g_t^{\pi_\mathcal{S}} = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, where r_i is the reward at timestep i. We denote the expectation of these returns as *value functions* $V_{\pi_\mathcal{S}}(s) = \mathbb{E}_{\pi_\mathcal{S}}[g_t \mid s_t = s]$ and $Q_{\pi_\mathcal{S}}(s, a) = \mathbb{E}_{\pi_\mathcal{S}}[g_t \mid s_t = s, a_t = a]$.

The POMDP framework [Kaelbling et al., 1998] additionally includes a set of observations Ω and an observation function $\Phi : \mathcal{S} \to \Delta\Omega$ that describes the probability $\Phi(\omega|s)$ of seeing observation ω in latent state s. POMDPs are a more general model of the world, since they contain MDPs as a special case: namely, when observations have a one-to-one correspondence with states. Similarly, POMDPs where states correspond to disjoint sets of observations are called *block MDPs* [Du et al., 2019]. However, such examples are rare; in typical POMDPs, a single observation ω does not contain enough information to fully resolve the state s. While agents need not *fully* resolve the underlying state to behave optimally, they must retain at least enough information across timesteps that the optimal policy becomes clear.

We are interested in the learning setting, where the agent has no knowledge of the underlying state s nor even the set of possible states $\mathcal{S}$ (let alone the transition, reward, and observation functions). It receives an observation $\omega_t \in \Omega$ at each timestep and must find a way to maximize expected return. One way to do this is to construct a *state representation*, perhaps using some form of memory, on which it can condition its behavior. A state representation $z \in Z$ is *Markovian* if at any timestep t, the representation z_t and action a_t together are a sufficient statistic for predicting the reward r_t and next representation z_{t+1}, instead of requiring the agent's whole history:

$$\Pr(z_{t+1}, r_t | z_t, a_t) = \Pr(z_{t+1}, r_t | z_t, a_t, \ldots, z_0, a_0). \tag{1}$$

States and observations are equivalent in MDPs, and this property is satisfied for both by definition, but in POMDPs it typically only holds for the underlying, unobserved state s—not the observations.

Markovian state representations have several desirable implications. First, if the Markov property holds then so does the Bellman equation: $V_{\pi_\mathcal{S}}(s) = \sum_{a \in \mathcal{A}} \pi_\mathcal{S}(a \mid s)\big(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s, a) V_{\pi_\mathcal{S}}(s')\big)$. The Bellman equation allows agents to estimate the value of a policy, from experiences, and without knowing T or R, via a recurrence relation over one-step returns,

$$V_{\pi_\mathcal{S}}^{(i+1)}(s) = \mathbb{E}_{\pi_\mathcal{S}}\left[r_t + \gamma V_{\pi_\mathcal{S}}^{(i)}(s_{t+1}) \mid s_t = s\right], \tag{2}$$

which converges to the unique fixed point $V_{\pi_\mathcal{S}}$. A second implication is that the transition and reward functions, and consequently the value functions $V_{\pi_\mathcal{S}}$ and $Q_{\pi_\mathcal{S}}$, have fixed-sized inputs and are therefore easy to parameterize, learn, and reuse. Finally, it follows from the Markov property that the optimal policy $\pi_\mathcal{S}^*$ can be expressed deterministically and does not require memory [Puterman, 1994].

We can unroll the Bellman equation over multiple timesteps to obtain a similar estimator that uses n-step returns: $V_{\pi_\mathcal{S}}(s) = \mathbb{E}_{\pi_\mathcal{S}}[g_{t:t+n} \mid s_t = s]$, where $g_{t:t+n} := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n V_{\pi_\mathcal{S}}(s_{t+n})$, with $g_{t:t+n} := g_t$ if the episode terminates before $t + n$ has been reached. The same process works for weighted combinations of such returns, including the exponential average:

$$V_{\pi_\mathcal{S}}^{\lambda}(s) = \mathbb{E}_{\pi_\mathcal{S}}\left[(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} g_{t:t+n} \;\middle|\; s_t = s\right], \tag{3}$$

with $V_{\pi_\mathcal{S}}^{\lambda=1}(s) = \mathbb{E}_{\pi_\mathcal{S}}[g_t \mid s_t = s]$ as a special case. Equation (3) defines the TD(λ) value function as an expectation over the so-called λ-return [Sutton, 1988]. Given an MDP and a fixed policy, the recurrence relations for all TD(λ) value functions share the same fixed point for any $\lambda \in [0, 1]$. If the Markov property does not hold, different λ may have different TD(λ) fixed points. In this work, we seek to characterize this phenomenon and leverage it for detecting and mitigating partial observability.

3 Detecting Partial Observability

Before we introduce our partial observability metric, let us first consider the T-maze example of Figure 1 under our two candidate decision-making frameworks: MDPs and POMDPs. In the T-maze

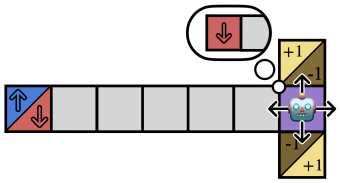

Figure 1: T-maze decision process. The agent must remember the initial observation to earn the maximum reward (+1).

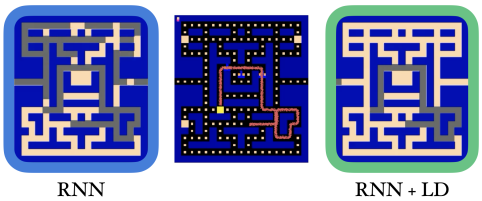

Figure 2: PacMan memory visualization. The agent moves within the maze (center), and we reconstruct the dot locations from the agent's memory. RNNs (left) benefit from the λ-discrepancy auxiliary loss (LD, right).

environment, the agent can only observe the color of its current square, and must remember the color of the starting square (sampled uniformly from **BLUE**/**RED**) to select the action at the junction that leads to the positive reward.

To model the T-maze as a POMDP, the state space must include the agent's position and the goal location. The transition function moves the agent deterministically in one of four directions according to the selected action. The goal location is sampled at the start of an episode, and defines both the reward function and the observation for the initial square. The observation function uniquely identifies these starting states, but all corridor states are aliased together (they map to the same observation), as are the two junction states.

We can convert our POMDP model into an MDP model by using observations Ω as the MDP's "states". This requires new transition and reward functions, T_Ω and R_Ω, which we define as: $T_\Omega(\omega' \mid \omega, a) := \sum_{s,s' \in \mathcal{S}} \Phi(\omega' \mid s') T_\mathcal{S}(s' \mid s, a) \Pr(s \mid \omega)$ and $R_\Omega(\omega, a) := \sum_{s \in \mathcal{S}} R_\mathcal{S}(s, a) \Pr(s \mid \omega)$, where $\Pr(s \mid \omega)$ is policy-dependent and describes how each hidden state $s_i \in \mathcal{S}$ contributes to the overall environment behavior when we see observation ω. While there are several sensible choices of weighting scheme for this type of state aggregation (uniform over states, relative frequency, etc.), only one of these choices coincides with the distribution $\Pr(s \mid \omega)$. That particular weighting scheme is the one that averages the time-dependent $\Pr(s_t \mid \omega_t)$ over all timesteps, weighted by visitation probability under the agent's policy, and discounted by γ; we explain how to construct it in Appendix A. We call the MDP defined in this way the *effective MDP model* for a given POMDP.

The effective MDP model marginalizes over histories (and POMDP hidden states). For example, the model predicts that going UP from the junction will reach the goal exactly half the time, because the transition dynamics marginalize over (equally likely) starting observations. Note that the POMDP model does no such averaging: if the agent initially observes **BLUE**, then going UP from the junction will *always* reach the goal, but DOWN never will. Thus, we see that, for the T-maze, there is a mismatch between the POMDP model and the effective MDP model. In other words, the POMDP's hidden states $\mathcal{S}$ are a Markovian representation, but its observations Ω are not, despite the fact that the effective MDP model treats them as Markovian "states".

In principle, an agent could measure partial observability by simultaneously modeling its environment as both a POMDP and an MDP, and comparing the models' predictions. Unfortunately, since the agent lacks any information about the unobserved state space $\mathcal{S}$, the POMDP model would require variable-length history inputs, and would come with computational and memory costs that grow exponentially with history length. Instead, we propose a model-free approach, using value functions defined only over observations, to approximate this comparison in a tractable way.

3.1 Value Function Estimation under Partial Observability

The Bellman equation and its sample-based recurrence relations (2) and (3) are defined for Markovian states. If we apply them to the observations of a POMDP, we are actually working with the effective MDP model of that POMDP, instead of the POMDP itself. To see this, consider one-step TD ($\lambda = 0$), where we use the same recurrence relation (2) but now our expectation is sampling from the POMDP:

$$\begin{aligned} V_\Omega^{\lambda=0}(\omega) &= \sum_{s \in \mathcal{S}} \Pr(s \mid \omega) \sum_{a \in \mathcal{A}} \pi(a \mid \omega) \Big(R_S(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{\omega' \in \Omega} \Phi(\omega' \mid s') T_\mathcal{S}(s' \mid a, s) V_\Omega^{\lambda=0}(\omega') \Big) \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid \omega) \Big(R_\Omega(\omega, a) + \gamma \sum_{\omega' \in \Omega} T_\Omega(\omega' \mid a, \omega) V_\Omega^{\lambda=0}(\omega') \Big), \end{aligned} \tag{4}$$

where we have suppressed V_Ω's dependence on the observation-based policy π for ease of notation.

We see from Equation (4) that the value function TD computes for a POMDP is the fixed point of the Bellman operator for the effective MDP model.[5]

By contrast, the Monte Carlo value estimator ($\lambda = 1$) does not exploit the Bellman equation at all; it simply averages over returns: $V_\pi^{\lambda=1}(s) = \mathbb{E}_\pi[g_t \mid s_t = s]$. Translating to POMDPs merely requires one additional expectation to convert from states to observations:

$$V_\Omega^{\lambda=1}(\omega) = \mathbb{E}_\pi\left[g_t \mid \omega_t = \omega\right] = \sum_{s\in\mathcal{S}} \Pr(s \mid \omega)\, \mathbb{E}_\pi\left[g_t \mid s_t = s\right] = \sum_{s\in\mathcal{S}} \Pr(s \mid \omega) V_\mathcal{S}^{\lambda=1}(s). \tag{5}$$

This means MC ($\lambda = 1$) effectively takes the hidden-state value function $V_\mathcal{S}$ of the POMDP and projects it into observation space, whereas TD ($\lambda = 0$) directly computes the value function for the projected model as an MDP. Interpolating λ between 0 and 1 smoothly varies the value estimate's reliance on the Markov assumption.

For generality, we derive an expression (see Appendix A) for Q_π^λ-values in terms of a given λ parameter (expressed in tensor notation for compactness) to reveal how λ trades off between the projected state value function and the value function of the projected model:

$$Q_\pi^\lambda = W K_\pi^\lambda {:} R^{\mathcal{SA}}, \text{where } K_\pi^\lambda = \Big(I - \gamma T\left(\lambda \Pi^\mathcal{S} + (1-\lambda)\Phi W^\Pi\right)\Big)^{-1}, \tag{6}$$

where the tensor product $:$ contracts two indices instead of one,[6] with tensors defined as follows:

Q_π^λ ($\Omega \times \mathcal{A}$) is a matrix of Q-values;
W ($\Omega \times \mathcal{S}$) contains the state-blending weights given by $\Pr(s \mid \omega)$ for observation ω;
I ($\mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A}$) is an identity tensor with $I_{sas'a'} = \mathbb{1}[s = s']\mathbb{1}[a = a']$;
T ($\mathcal{S} \times \mathcal{A} \times \mathcal{S}$) contains the hidden-state transition probabilities $T(s' \mid a, s)$;
$\Pi^\mathcal{S}$ ($\mathcal{S} \times \mathcal{S} \times \mathcal{A}$) contains the effective policy over hidden states (see Appendix A);
Φ ($\mathcal{S} \times \Omega$) is the observation function, containing probabilities $\Phi(\omega \mid s)$;
W^Π ($\Omega \times \mathcal{S} \times \mathcal{A}$) combines W with $\Pi^\mathcal{S}$ to obtain probabilities $\Pr(s, a \mid \omega)$;
$R^{\mathcal{SA}}$ ($\mathcal{S} \times \mathcal{A}$) contains the hidden-state rewards $R(s, a)$.

Intuitively, this equation says TD(λ) interpolates between the transition matrix ($T\Pi^\mathcal{S}$) and the *projected* transition matrix ($T\Phi W^\Pi$) and computes the observation-space value function for the result.

3.2 λ-Discrepancy

We have shown that, under partial observability, there may be a difference between Q_π^λ value functions for two different λ parameters due to the implicit Markov assumption in TD(λ). We call this difference the *λ-discrepancy*, and we propose to use it as a measure of partial observability.

Definition 1 *For a given POMDP model $\mathcal{P}$ and policy π, the* λ-discrepancy $\Lambda_{\mathcal{P},\pi}^{\lambda_1,\lambda_2}$ *is the weighted norm of the difference between the Q-functions estimated by TD(λ) for $\lambda \in \{\lambda_1, \lambda_2\}$:*

$$\Lambda_{\mathcal{P},\pi}^{\lambda_1,\lambda_2} := \left\|Q_\pi^{\lambda_1} - Q_\pi^{\lambda_2}\right\| = \left\|W\Big(K_\pi^{\lambda_1} - K_\pi^{\lambda_2}\Big){:}R^{\mathcal{SA}}\right\|.$$

The choice of norm can be arbitrary, as can the norm weighting scheme, as long as it assigns positive weight to all reachable observation-action pairs. We discuss choices of weighted norm in Appendix E.2. For brevity, we suppress the $\mathcal{P}$ subscript when the POMDP model is clear from context.

A useful property (that we will prove in Theorem 2) is that if the POMDP has Markovian observations, the λ-discrepancy is exactly zero for all policies. However, for it to be a useful measure of parital observability, we must also show that the λ-discrepancy is reliably non-zero when observations are non-Markovian. For this, we have the following theorem.

[5] This equivalence justifies our choice of $\Pr(s|\omega)$ when defining T_Ω and R_Ω for the effective MDP model, since states appear in precisely this proportion when we generate experiences in the POMDP.

[6] For 3-dimensional tensors A and B, $(AB)_{ijlm} = \sum_k A_{ijk}B_{klm}$, and $(A{:}B)_{il} = \sum_{jk} A_{ijk}B_{jkl}$.

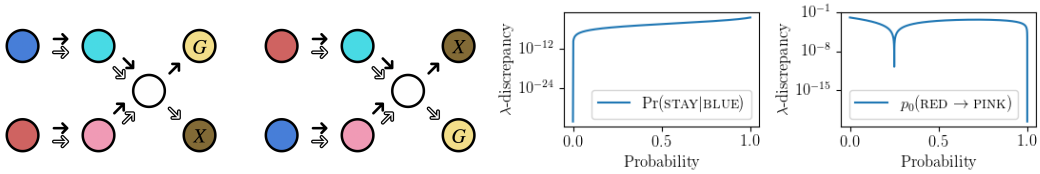

Figure 3: (Left) The Parity Check environment, a POMDP with zero λ-discrepancy for every policy. (Right) Minor modifications to the transition dynamics (left) or initial state distribution (right) result in non-zero λ-discrepancy for almost all policies.

Theorem 1 *Given a POMDP model $\mathcal{P}$ and distinct $\lambda \neq \lambda'$, if there exists a policy $\pi : \Omega \to \Delta\mathcal{A}$ such that $\Lambda^{\lambda,\lambda'}_{\mathcal{P},\pi} \neq 0$, then $\Lambda^{\lambda,\lambda'}_{\mathcal{P},\pi} \neq 0$ for all policies except at most a set of measure zero.*

Proof sketch: We formulate the λ-discrepancy of a given policy as the norm of an analytic function, then use the fact that analytic functions are zero everywhere or almost nowhere, along with the fact that norms preserve this property. The full proof is given in Appendix B.

Intuitively, Theorem 1 says that the λ-discrepancy can detect non-Markovian observations. If it is possible to reveal that a POMDP's observation-space value function does not match that of its effective MDP model, then almost all policies will do so. The theorem further suggests that even if a particular policy has zero λ-discrepancy, small perturbations to that policy will almost surely detect partial observability if it is present.

3.3 What conditions cause the λ-discrepancy to be zero?

There are two cases in which the λ-discrepancy is zero for all policies, which we can see by inspecting Definition 1. Because norms are positive definite, it suffices to consider the expression inside the norm. The only ways for this expression to equal zero are either when the difference term $\left(K^{\lambda_1}_\pi - K^{\lambda_2}_\pi\right)$ is zero (which we will show implies Markovian observations), or it is non-zero but is projected away by the outer terms W and/or $R^{\mathcal{SA}}$. We first consider when the two inner terms—which are the only terms that depend on λ—are equal, i.e. $K^{\lambda_1}_\pi = K^{\lambda_2}_\pi$.

Theorem 2 *For any POMDP $\mathcal{P}$ and any $\lambda_1 \neq \lambda_2$, $K^{\lambda_1}_\pi = K^{\lambda_2}_\pi$ if and only if $\mathcal{P}$ is a block MDP.*

Proof sketch: Using Eq. (6), $K^{\lambda_1}_\pi = K^{\lambda_2}_\pi$ can be simplified to $\Pi^{\mathcal{S}} = \Phi W^{\Pi}$, which is satisfied if and only if $\Phi W = I^{\mathcal{S}}$, i.e. each observation is generated by exactly one hidden state. Proof in Appendix C.

Now let us consider the case where the difference between K^λ_π is projected away by the outer terms W and $R^{\mathcal{SA}}$. In Appendix D, we expand Equation (6) as a power series and consider when $Q^\lambda_\pi - Q^{\lambda'}_\pi = 0$. It is instructive to let $\lambda = 0$ and $\lambda' = 1$, since this simplifies the math and allows us to group terms in the power series by their γ^n coefficients. This leads to the following condition, which, if it holds for all time horizons k, ensures $\Lambda^{0,1}_{\mathcal{P},\pi} = 0$:

$$W\left(T\Pi^{\mathcal{S}}\right)^k : R^{\mathcal{SA}} = W\left(T\Phi W^{\Pi}\right)^k : R^{\mathcal{SA}}. \tag{7}$$

There are several ways to satisfy Equation (7). Let us start with the innermost terms and work our way outwards: when $\Pi^{\mathcal{S}} = \Phi W^{\Pi}$, there is no state aliasing and Theorem 2 applies. Assuming Theorem 2 does not apply, there may still be some uninteresting cases where the state transitions ($T\Pi^{\mathcal{S}}$) and the projected state transitions ($T\Phi W^{\Pi}$) are identical (e.g. if the transition probabilities are the same for any states with aliased observations). If the transition probabilities are not identical, then they will also differ when rolled out for k steps. Equation (7) requires that when these k-step rollout dynamics differ, there are no differences in reward prediction at any time horizon, either because of how states are averaged together (W), or because rewards are constant for all states we might confuse. The following example illustrates one way this can occur.

Parity Check Environment. A helpful example for understanding the limitations of the λ-discrepancy is the Parity Check environment of Figure 3. This POMDP has four equally likely starting states, each associated with a pair of colors that the agent will see during the first two timesteps. At the subsequent (white) junction state, the agent must decide based on these colors whether to go

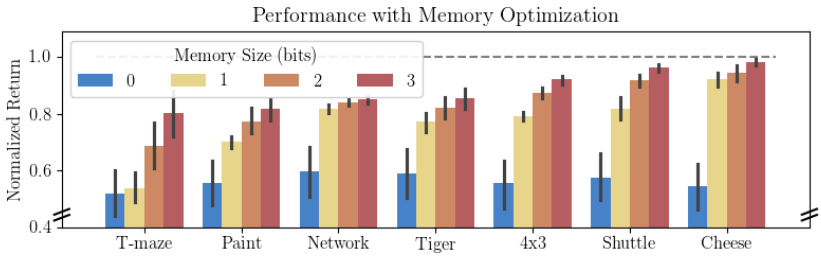

Figure 4: Memory optimization increases normalized return of subsequent policy gradient learning. Performance is calculated as the expected start-state value, and is normalized between a random policy ($y = 0$) and the optimal belief state policy ($y = 1$) found with a POMDP solver [Cassandra, 2003]. Error bars are 95% confidence intervals over 30 seeds.

UP (black arrow) or DOWN (white arrow). The rewards are defined such that UP is optimal if and only if the color family matches (i.e. **RED** $\rightarrow$ **PINK** or **BLUE** $\rightarrow$ **CYAN**).

Note that states do not all have unique colors, so Theorem 2 does not apply. However, the expected return for any observation and time horizon is zero, so Equation (7) is satisfied, and the λ-discrepancy is zero for all policies and choices of λ, λ', by symmetry. However, this symmetry disappears if we modify the example even slightly, such as by changing the start state probabilities or by introducing a "stay-in-place" action at any one color. These modifications lead to the λ-discrepancies in Figure 3, which offer some reassurance that such edge cases are the exception and not the rule.

Taken together, Theorems 1 and 2 suggest that the λ-discrepancy could be a useful measure for detecting partial observability. In the next section, we demonstrate the efficacy of using the λ-discrepancy to learn memory functions that mitigate partial observability.

4 Memory Learning with the λ-Discrepancy

We have established that the λ-discrepancy can identify that we need memory, but it cannot yet tell us what to remember. For that, we must replace the POMDP $\mathcal{P}$ in Definition 1 with a memory-augmented version. In general, an agent's memory can be any mapping from a variable length history $(\omega_0, a_0, ..., a_{t-1}, r_{t-1}, \omega_t)$ to an internal memory state m within a set of possible memory states M. For practical reasons, we restrict our attention to recurrent memory functions that update an internal representation incrementally from fixed-size inputs.

We define a *memory function* $\mu : \Omega \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{M}$ as a mapping from an observation, action, and memory state, (ω, a, m), to a next memory state $m' = \mu(\omega, a, m)$. Given a POMDP $\mathcal{P}$, a memory function μ induces a *memory-augmented* POMDP $\mathcal{P}^\mu$, with extended states $\mathcal{S}_\mathcal{M} = \mathcal{S} \times \mathcal{M}$, actions $\mathcal{A}_\mathcal{M} = \mathcal{A} \times \mathcal{M}$, and observations $\Omega_\mathcal{M} = \Omega \times \mathcal{M}$. The augmented transition dynamics $T_\mathcal{M}$ preserve the original transition dynamics T for states $\mathcal{S}$ while allowing the agent full observability and control over memory states $\mathcal{M}$. Memory functions naturally lead to memory-augmented policies $\pi_\mu : \Omega_\mathcal{M} \rightarrow \Delta\mathcal{A}_\mathcal{M}$ and value functions $V_{\pi,\mu} : \Omega_\mathcal{M} \rightarrow \mathbb{R}$ and $Q_{\pi,\mu} : \Omega_\mathcal{M} \times \mathcal{A}_\mathcal{M} \rightarrow \mathbb{R}$ that reflect the expected return under such policies. For details, see Appendix E.1.

The λ-discrepancy (Definition 1) applies equally well to memory-augmented POMDPs, and can thus be used to determine whether a particular memory function μ induces a POMDP $\mathcal{P}^\mu$ with a Markovian observation space $\Omega_\mathcal{M}$. We can also use it as a training objective for *learning* such a memory function. We conduct a proof-of-concept experiment on several classic POMDPs for which we can obtain closed-form gradients of the λ-discrepancy with respect to a parametrized memory function. In each domain, we randomly generate a set of stochastic policies, select the one with maximal λ-discrepancy $\Lambda^{0,1}_\mathcal{P}$, and adjust the parameters of a memory function μ to minimize $\Lambda^{0,1}_{\mathcal{P}^\mu}$ via gradient descent. Figure 4 shows the improvement in policy gradient performance due to the resulting memory function for increasing memory sizes. The details of this experiment are provided in Appendix F.

We see that the λ-discrepancy can help mitigate partial observability, provided that the agent can optimize it in closed-form. The question that remains is: can we somehow integrate λ-discrepancy minimization into a sample-based learning algorithm and scale this up to more challenging problems?

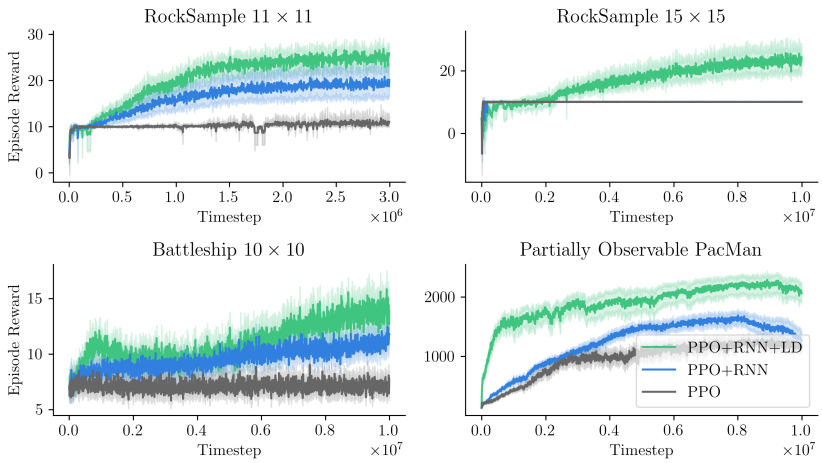

Figure 5: The λ-discrepancy auxiliary objective (LD) improves performance over recurrent (RNN) and memoryless PPO. Learning curves shown are the mean and 95% confidence interval over 30 runs.

5 A Scalable, Online Learning Objective

So far, we have shown that the λ-discrepancy can detect partial observability in theory and can mitigate it under certain idealized conditions. Now we demonstrate how to integrate our metric into sample-based deep reinforcement learning to solve problems requiring large, complex memory functions.

5.1 Combining the λ-Discrepancy with PPO

To minimize the λ-discrepancy, we augment a recurrent version of the proximal policy optimization (PPO) algorithm [Schulman et al., 2017] with an auxiliary loss. We use recurrent PPO as our base algorithm due to its strong performance in many POMDPs [Ni et al., 2022b], and since the λ-discrepancy is a natural extension of generalized advantage estimation [Schulman et al., 2015], which is used in PPO. In this algorithm, a recurrent neural network [Amari, 1972] (specifically a gated recurrent unit, or GRU [Cho et al., 2014]) is used as the memory function μ that returns a latent state representation given previous latent state and an observation. This latent representation is used as input to an actor network to return a distribution over actions, as well as a critic. The critic is usually a value function network which learns a truncated TD(λ) value estimate for its advantage estimate.

To estimate the λ-discrepancy, we learn two TD(λ) value function networks with different λ, parameterized by $\theta_{V,1}$ and $\theta_{V,2}$ respectively, and minimize their mean squared difference as an auxiliary loss:

$$L_\Lambda(\theta) = \mathbb{E}_\pi \left[\left(V^{\lambda_1}_{\theta_{V,1}}(z_t) - V^{\lambda_2}_{\theta_{V,2}}(z_t) \right)^2 \right], \tag{8}$$

where $z_t = \mu_{\theta_{\text{RNN}}}(\omega_t, z_{t-1})$ is the latent state output by the GRU, and θ represents all parameters. We train this neural network end-to-end with the standard PPO actor-critic losses and backpropagation through time [Mozer, 1995]. Note that any algorithm that uses value estimates could potentially leverage the λ-discrepancy. Full details of the algorithm are provided in Appendix G.1.

5.2 Large Partially Observable Environments

We evaluate our approach on a suite of four hard partially observable environments that require complex memory functions. *Battleship* [Silver and Veness, 2010] requires reasoning about unknown ship positions and remembering previous shots. Partially observable *PacMan* [Silver and Veness, 2010] requires localization within the map while tracking dot status and avoiding ghosts with only short-range sensors. *RockSample (11, 11)* and *RockSample (15, 15)* [Smith and Simmons, 2004] have stochastic sensors and require remembering which rocks have been sampled. While these environments were originally used to evaluate partially observable planning algorithms in large-scale POMDPs [Silver and Veness, 2010], we use them to test our sample-based learning algorithms due to the complexity of their required memory functions. See Appendix G.2 for more details.

5.3 Experiments

We show experimental results with regular PPO, recurrent PPO, and our λ-discrepancy-augmented recurrent PPO in Figure 5. In Figure 2, we also visualize the agent's memory for the partially observable PacMan environment by reconstructing the dot locations from the RNN latent state to show where the agent "thinks" it has been (see Appendix G.5). We performed a hyperparameter sweep for each method and report learning curves for undiscounted return (see Appendix G.3 for discounted learning curves and Appendix G.4 for additional experimental details).

The λ-discrepancy objective leads to significantly better final performance and learning rate versus recurrent and memoryless PPO in all environments. In RockSample $(15, 15)$, the baseline agents quickly learn to exit immediately (for +10 reward), but never improve on this. By contrast, the λ-discrepancy objective leads to better performance, as the agent learns the missing features that allow it to express better policies. We also run experiments on the classic POMDPs from Section 4, but due to the size of these problems, both baseline and our proposed approach performed almost optimally. We show these results in Appendix G.6.

Besides the improvements to performance, the best selected hyperparameters from our sweep support the theory developed in Section 3. The sweep selected two λs with a large difference between λ_1 and λ_2, with either of the λs close to either 0 or 1. This reflects our theory that small or large λs control the weakness or strength of the model's Markov assumption, and that the difference of the two is a good indicator for partial observability. In addition to this, our hyperparameter sweep also includes a PPO variant with two different TD(λ) value functions but without the λ-discrepancy auxiliary loss described in Section 5.1. These agents were never selected in the sweep, and using the loss in Equation 8 seems to only help with performance in tested environments.

6 Related Work

There is an interesting connection between state abstraction [Li et al., 2006], which selectively removes information from state, and partial observability mitigation, which aims to recover state from incomplete observations. Allen et al. [2021] investigated the state abstraction perspective and characterized the properties under which abstract state representations of MDPs either do or do not preserve the Markov property. Several other approaches characterize and measure partial observability and POMDPs. While POMDPs have been shown to be computationally intractable in general [Papadimitriou and Tsitsiklis, 1987], various works have studied complexity measures [Zhang et al., 2012] and defined subclasses with tractable solutions [Littman, 1993, Liu et al., 2022].

The most common strategies for mitigating partial observability are memory-based approaches that summarize history. Early approaches relied on discrete representations of history, including tree representations [McCallum, 1996] or finite-state controllers [Meuleau et al., 1999]. Modern approaches mostly use RNNs [Amari, 1972] trained via backpropagation through time (BPTT) [Mozer, 1995] to tackle non-Markovian decision processes [Schmidhuber, 1990]. Various approaches use recurrent function approximation to learn better state representations. One successful approach is learning a recurrent value function [Lin and Mitchell, 1993, Bakker, 2001, Hausknecht and Stone, 2015] that uses TD error as a learning signal for memory. Policy gradient methods, including PPO (which we compare to), have also been used with recurrent learning to mitigate partial observability [Wierstra et al., 2007, Heess et al., 2015]. Model-based methods can learn a recurrent dynamics model [Hafner et al., 2020] to facilitate planning alongside reinforcement learning. These approaches learn their representations implicitly to improve prediction error, rather than explicitly to mitigate partial observability.

7 Conclusion

We introduce the λ-discrepancy: an observable and differentiable measure of non-Markovianity suitable for mitigating partial observability. The λ-discrepancy is the difference between two distinct value functions estimated using TD(λ), for two different λ values. We characterize the λ-discrepancy and prove that it reliably distinguishes MDPs from POMDPs. We then use it as a memory learning objective and demonstrate that minimizing it in closed-form helps learn useful memory functions in small-scale POMDPs. Finally, we propose a deep reinforcement learning algorithm which leverages the λ-discrepancy as an auxiliary loss, and show that it significantly improves the performance of a baseline recurrent PPO agent on a set of large and challenging partially observable tasks.

Author Contributions

CA, AK and RYT led the project. CA, OG, GK, MLL and SL came up with the initial idea. CA, OG, MLL, SL, and DS conducted the first conceptual investigations and proof-of-concept experiments. AK led the theoretical work, with support from CA, SL, RP, and RYT. RYT led the algorithm development, with support from CA, SL, and GK. CA and RYT led the implementation and experiments, with support from SL, NP, and DS. RP discovered the class of parity check examples and showed that they have zero λ-discrepancy. CA led the writing, with support from AK and RYT. OG, GK, MLL, SL, and RP advised on the project and provided regular feedback.

Acknowledgments and Disclosure of Funding

We acknowledge and thank Saket Tiwari, Anita de Mello Koch, Sam Musker, Brad Knox, Stuart Russell, and our colleagues at Brown University and UC Berkeley for their valuable advice and discussions towards completing this work. We also thank our reviewers from NeurIPS'23 and ICLR'24 for comments on earlier drafts.

This work was generously supported under NSF grant 1955361 and CAREER grant 1844960 to George Konidaris, NSF fellowships to Aaron Kirtland and Sam Lobel, ONR grant N00014-22-1-2592, a gift from Open Philanthropy to the Center for Human-Compatible AI at Berkeley, and an AI2050 Senior Fellowship for Stuart Russell from the Schmidt Fund for Strategic Innovation.

References

Cameron Allen, Neev Parikh, Omer Gottesman, and George Konidaris. Learning Markov state abstractions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 8229–8241, 2021.

S.-I. Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on Computers*, C-21(11):1197–1206, 1972.

Bram Bakker. Reinforcement learning with long short-term memory. *Advances in Neural Information Processing Systems*, 14, 2001.

M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang. Autonomous Navigation of Stratospheric Balloons using Reinforcement Learning. *Nature*, pages 77–82, 2020.

Clément Bonnet, Daniel Luo, Donal John Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Petrus Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Neil Waters, Mohamed Ali Ali Mimouni, Ulrich Armel Mbou Sob, Ruan John de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In *The Twelfth International Conference on Learning Representations*, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Anthony Cassandra. The POMDP page. https://www.pomdp.org/, 2003.

Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, pages 1023–1028, 1994.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, volume 1992, pages 183–188. Citeseer, 1992.

Shi Dong, Benjamin Van Roy, and Zhengyuan Zhou. Simple agent, complex environment: Efficient reinforcement learning with agent states. *Journal of Machine Learning Research*, 23(255):1–54, 2022.

Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pages 1665–1674. PMLR, 2019.

Antoine Galataud. HVAC processes control: Can AI help?, Jan 2021.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable MDPs. In *Proceedings of the 2015 American Association for Artificial Intelligence (AAAI) Conference*, 2015.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. In *the Deep Reinforcement Learning Workshop at NeurIPS*, 2015.

Herbert Jaeger. Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach. *GMD-Forschungszentrum Informationstechnik, 2002.*, 5, 01 2002.

L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Steven G Krantz and Harold R Parks. *A primer of real analytic functions*. Springer Science & Business Media, 2002.

Nicholas Kushmerick, Steve Hanks, and Daniel S Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

L. Li, T.J. Walsh, and M.L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

Long-Ji Lin and Tom M Mitchell. Reinforcement learning with hidden states. *From animals to animats*, 2:271–280, 1993.

Michael L. Littman. An optimization-based categorization of reinforcement learning environments. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. The MIT Press, 04 1993. ISBN 9780262287159.

Qinghua Liu, Alan Chung, Csaba Szepesvari, and Chi Jin. When is partially observable reinforcement learning not scary? In *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pages 5175–5220. PMLR, July 2022.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

Andrew Kachites McCallum. *Reinforcement learning with selective perception and hidden state*. University of Rochester, 1996.

Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 427–436, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

Boris Samuilovich Mityagin. The zero set of a real analytic function. *Matematicheskie Zametki*, 107 (3):473–475, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937. PMLR, June 2016.

Michael Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3, 01 1995.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16691–16723. PMLR, 17–23 Jul 2022a.

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

Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1088–1094, San Francisco, CA, USA, 1995.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318. PMLR, 17–19 Jun 2013.

M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

Jürgen Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann, 1990.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Neural Information Processing Systems*, 2010.

Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, page 520–527. AUAI Press, 2004.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1): 9–44, 1988.

R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

Ruo Yu Tao, Adam White, and Marlos C. Machado. Agent-state construction with auxiliary inputs. *Transactions on Machine Learning Research*, 2023.

Daan Wierstra, Alexander Foerster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *Artificial Neural Networks – ICANN 2007*, pages 697–706, Berlin, Heidelberg, 2007.

Zongzhang Zhang, Michael Littman, and Xiaoping Chen. Covering number as a complexity measure for POMDP planning and learning. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

A TD(λ) Fixed Point

Here we derive the fixed point of the TD(λ) action-value update rule in a POMDP, following the Markov version by Sutton [1988]. First, define the expected return given initial observation ω_0 and initial action a_0 as

$$
\begin{aligned}
&\mathbb{E}_\pi(G^n|\omega_0, a_0) \\
&\quad = \sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{r_0} \Pr(r_0|s_0, a_0) r_0 \\
&\qquad + \gamma \sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{s_1} \Pr(s_1|s_0, a_0) \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \Pr(\omega_1|s_1) \Pr(a_1|\omega_1) \sum\nolimits_{r_1} \Pr(r_1|s_1, a_1) r_1 \\
&\qquad + \gamma^2 \sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{s_1} \Pr(s_1|s_0, a_0) \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \sum\nolimits_{s_2} \Pr(\omega_1|s_1) \Pr(a_1|\omega_1) \Pr(s_2|s_1, a_1) \\
&\qquad\qquad * \sum\nolimits_{\omega_2} \sum\nolimits_{a_2} \Pr(\omega_2|s_2) \Pr(a_2|\omega_2) \sum\nolimits_{r_2} \Pr(r_2|s_2, a_2) r_2 \\
&\qquad + \dots
\end{aligned}
$$

We can define the n-step bootstrapped update rule from this given a value matrix Q_π by replacing part of the term with coefficient γ^n with a Q_π value, e.g. for the $n = 2$ case, we get

$$
\begin{aligned}
Q_\pi^{(i+1)}(\omega_0, a_0) = &\sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{r_0} \Pr(r_0|s_0, a_0) r_0 \\
&+ \gamma \sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{s_1} \Pr(s_1|s_0, a_0) \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \Pr(\omega_1|s_1) \Pr(a_1|\omega_1) \\
&\qquad\qquad * \sum\nolimits_{r_1} \Pr(r_1|s_1, a_1) r_1 \\
&+ \gamma^2 \sum\nolimits_{s_0} \Pr(s_0|\omega_0) \sum\nolimits_{s_1} \Pr(s_1|s_0, a_0) \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \Pr(\omega_1|s_1) \Pr(a_1|\omega_1) \\
&\qquad * \sum\nolimits_{s_2} \Pr(s_2|s_1, a_1) \sum\nolimits_{\omega_2} \sum\nolimits_{a_2} \Pr(\omega_2|s_2) \Pr(a_2|\omega_2) Q_\pi^{(i)}(\omega_2, a_2).
\end{aligned}
$$

Translating these expressions into matrix notation, we have

$$
\begin{aligned}
\mathbb{E}_\pi(G^n|\omega_0, a_0) = &\sum\nolimits_{s_0} W_{\omega_0, s_0} R_{s_0, a_0} \\
&+ \gamma \sum\nolimits_{s_0} W_{\omega_0, s_0} \sum\nolimits_{s_1} T_{s_0, a_0, s_1} \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \Phi_{s_1, \omega_1} \pi_{\omega_1, a_1} R_{s_1, a_1} \\
&+ \gamma^2 \sum\nolimits_{s_0} W_{\omega_0, s_0} \sum\nolimits_{s_1} T_{s_0, a_0, s_1} \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \sum\nolimits_{s_2} \Phi_{s_1, \omega_1} \pi_{\omega_1, a_1} T_{s_1, a_1, s_2} \\
&\qquad * \sum\nolimits_{\omega_2} \sum\nolimits_{a_2} \Phi_{s_2, \omega_2} \pi_{\omega_2, a_2} R_{s_2, a_2} \\
&+ \dots,
\end{aligned}
$$

where the terms W, T, R, and Φ are as in Equation 6, and π is the $\Omega \to \Delta\mathcal{A}$ policy written as an $\Omega \times \mathcal{A}$ tensor with entries in $[0, 1]$. In particular, $W_{\omega,s} = \Pr(s|\omega)$, which averages $\Pr(s_t|\omega_t)$ over all timesteps, weighted by visitation probability and discounted by γ. This is a well-defined stationary quantity, and it can be computed as follows. First solve the system $Cx = b$ to find the discounted state occupancy counts $x = c(s)$, where $b = p_0$ is the initial state distribution over $\mathcal{S}$, and $C = (I - \gamma(T^\pi)^\top)$ accounts for the policy-dependent state-to-state transition dynamics T^π defined by $T^\pi_{s,s'} = \sum_\omega \sum_a \Pr(\omega|s) \Pr(a|\omega) \Pr(s'|s, a)$. Then $\Pr(s|\omega) \propto c(s) * \Phi(\omega|s)$, so we can just multiply these terms together and renormalize. For the 2-step bootstrapped update rule above, we have:

$$
\begin{aligned}
Q_\pi^{(i+1)}(\omega_0, a_0) = &\sum\nolimits_{s_0} W_{\omega_0, s_0} R_{s_0, a_0} \\
&+ \gamma \sum\nolimits_{s_0} W_{\omega_0, s_0} \sum\nolimits_{s_1} T_{s_0, a_0, s_1} \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \Phi_{s_1, \omega_1} \pi_{\omega_1, a_1} R_{s_1, a_1} \\
&+ \gamma^2 \sum\nolimits_{s_0} W_{\omega_0, s_0} \sum\nolimits_{s_1} T_{s_0, a_0, s_1} \sum\nolimits_{\omega_1} \sum\nolimits_{a_1} \sum\nolimits_{s_2} \Phi_{s_1, \omega_1} \pi_{\omega_1, a_1} T_{s_1, a_1, s_2} \\
&\qquad * \sum\nolimits_{\omega_2} \Phi_{s_2, \omega_2} \sum\nolimits_{a_2} \Pi_{\omega_2, a_2} Q_\pi^{(i)}(\omega_2, a_2).
\end{aligned}
$$

Let Π be an $\Omega\times\Omega\times\mathcal{A}$ representation of the $\Omega\times\mathcal{A}$ policy π with $\Pi_{\omega,\omega',a} = \mathbb{1}[\omega = \omega']\pi_{\omega,a}$. Likewise, let $\Pi^{\mathcal{S}}$ be the effective policy over latent states, an $\mathcal{S}\times\mathcal{S}\times\mathcal{A}$ representation of the matrix $\Phi\pi$, i.e. $\Pi^{\mathcal{S}}_{s,s',a} = \mathbb{1}[s = s'](\Phi\pi)_{s,a}$. Given a matrix of action-values $Q^{(i)}_\pi$, the n-step update rule is:

$$Q^{(i+1)}_\pi = Q_{\pi,n}(Q^{(i)}_\pi) := W\left(\sum\nolimits_{k=0}^{n-1}(\gamma T\Pi^{\mathcal{S}})^k{:}R^{\mathcal{SA}} + \gamma(\gamma T\Pi^{\mathcal{S}})^{n-1}{:}T\Phi\Pi{:}Q^{(i)}_\pi\right),$$

where R^{SA} is the matrix of reward values as described in Equation 6, and we use $:$ to denote double tensor contraction, while all other tensor products contract a single index.

We also have the standard definition of the TD(λ) update rule as:

$$Q^{(i+1)}_\pi = (1-\lambda)\sum\nolimits_{n=1}^{\infty}\lambda^{n-1}Q_n(Q^{(i)}_\pi).$$

We are concerned with the fixed point of this update rule, which we refer to as Q^λ_π:

$$Q^\lambda_\pi = (1-\lambda)\sum\nolimits_{n=1}^{\infty}\lambda^{n-1}W\left(\sum\nolimits_{k=0}^{n-1}(\gamma T\Pi^{\mathcal{S}})^k{:}R^{\mathcal{SA}} + \gamma(\gamma T\Pi^{\mathcal{S}})^{n-1}{:}T\Phi\Pi{:}Q^\lambda_\pi\right).$$

Separating this into a reward part with factor $R^{\mathcal{SA}}$ and a value part with factor Q^λ_π, we find that the value part is

$$\begin{aligned}&(1-\lambda)W\left(\sum\nolimits_{n=1}^{\infty}(\lambda\gamma T\Pi^{\mathcal{S}})^{n-1}\right){:}\gamma T\Phi\Pi{:}Q^\lambda_\pi\\ &\quad= (1-\lambda)W(I-\lambda\gamma T\Pi^{\mathcal{S}})^{-1}{:}\gamma T\Phi\Pi{:}Q^\lambda_\pi,\end{aligned}$$

and for the reward part, we have the coefficients of $R^{\mathcal{SA}}$ in the table below for values of n and k

	$k=0$	1	2	...
$n=1$	1			...
2	λ	$\lambda\gamma T\Pi^{\mathcal{S}}$		...
3	λ^2	$\lambda^2\gamma T\Pi^{\mathcal{S}}$	$\lambda^2(\gamma T\Pi^{\mathcal{S}})^2$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

where each term is multiplied by $(1-\lambda)W$ in front. We can then see by summing over rows before columns that the reward part is:

$$\begin{aligned}&(1-\lambda)W\sum\nolimits_{k=0}^{\infty}\frac{1}{1-\lambda}(\lambda\gamma T\Pi^{\mathcal{S}})^k{:}R^{\mathcal{SA}}\\ &\quad= W(I-\lambda\gamma T\Pi^{\mathcal{S}})^{-1}{:}R^{\mathcal{SA}}.\end{aligned}$$

So we rewrite Q^λ_π as follows:

$$Q^\lambda_\pi = W\left(\left(I^{\mathcal{SA}} - \lambda\gamma T\Pi^{\mathcal{S}}\right)^{-1}{:}\left(R^{\mathcal{SA}} + (1-\lambda)\gamma T\Phi\Pi{:}Q^\lambda_\pi\right)\right).$$

Now let $W^{\mathcal{A}} = W\otimes I^{\mathcal{A}}$, which is $\Omega\times\mathcal{A}\times\mathcal{S}\times\mathcal{A}$, and $W^\Pi = \Pi{:}W^{\mathcal{A}}$, which is $\Omega\times\mathcal{S}\times\mathcal{A}$. Here, $\otimes$ means the Kronecker product. This essentially repeats the W matrix $|\mathcal{A}|$ times to incorporate actions into the tensor. Note that for any $\mathcal{S}\times\mathcal{A}$ tensor G, $(W^{\mathcal{A}}{:}G) = WG$. This is because $(W^{\mathcal{A}}{:}G)_{\omega a} = \sum_{s,a'}W^{\mathcal{A}}_{\omega asa'}G_{sa'}$, and the only nonzero terms in this sum are those such that $a = a'$. For these indices, $W^{\mathcal{A}}_{\omega asa'} = W_{\omega s}$, so $\sum_{s,a'}W^{\mathcal{A}}_{\omega asa'}G_{sa'} = \sum_s W_{\omega s}G_{sa} = (WG)_{\omega a}$.

Also, let $F = \left(I^{\mathcal{SA}} - \lambda\gamma T\Pi^{\mathcal{S}}\right)^{-1}$, which is $\mathcal{S}\times\mathcal{A}\times\mathcal{S}\times\mathcal{A}$. Then we find:

$$\begin{aligned}Q^\lambda_\pi &= W^{\mathcal{A}}{:}\left(\left(I-\lambda\gamma T\Pi^{\mathcal{S}}\right)^{-1}{:}\left(R^{\mathcal{SA}} + (1-\lambda)\gamma T\Phi\Pi{:}Q^\lambda_\pi\right)\right)\\ &= W^{\mathcal{A}}{:}\left(F{:}\left(R^{\mathcal{SA}} + (1-\lambda)\gamma T\Phi\Pi{:}Q^\lambda_\pi\right)\right)\\ &= W^{\mathcal{A}}{:}F{:}R^{\mathcal{SA}} + W^{\mathcal{A}}{:}F{:}(1-\lambda)\gamma T\Phi\Pi{:}Q^\lambda_\pi.\end{aligned}$$

At this point, we can subtract the second term on the right hand side from both sides, factor out Q_π^λ on the right, and multiply by $\left(I^{\Omega\mathcal{A}} - (1-\lambda)\gamma W^{\mathcal{A}}{:}F{:}T\Phi\Pi\right)^{-1}$ on the left of both sides to obtain:

$$\begin{aligned}
Q_\pi^\lambda &= \left(I^{\Omega\mathcal{A}} - (1-\lambda)\gamma W^{\mathcal{A}}{:}F{:}T\Phi\Pi\right)^{-1} W^{\mathcal{A}}{:}F{:}R^{\mathcal{SA}} \\
&= W^{\mathcal{A}}{:}\left(I^{\mathcal{SA}} - (1-\lambda)\gamma F{:}T\Phi\Pi{:}W^{\mathcal{A}}\right)^{-1}{:}F{:}R^{\mathcal{SA}} \\
&= W\left(I^{\mathcal{SA}} - (1-\lambda)\gamma F{:}T\Phi W^{\Pi}\right)^{-1}{:}F{:}R^{\mathcal{SA}} \\
&= W\Big(F + (1-\lambda)\gamma F{:}T\Phi W^{\Pi}{:}F + \dots \\
&\qquad + (1-\lambda)^k\gamma^k (F{:}T\Phi W^{\Pi})^k{:}F + \dots\Big){:}R^{\mathcal{SA}},
\end{aligned}$$

where the last equality follows from expanding the geometric series. Now we use the identity $(A-B)^{-1} = \sum_{k=0}^{\infty}(A^{-1}B)^k A^{-1}$ to find:

$$\begin{aligned}
Q_\pi^\lambda &= W\left(F^{-1} - (1-\lambda)\gamma T\Phi W^{\Pi}\right)^{-1}{:}R^{\mathcal{SA}} \\
&= W\left(I - \gamma T\left(\lambda\Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right)\right)^{-1}{:}R^{\mathcal{SA}}.
\end{aligned}$$

To recap our previous definitions, W is an $\Omega{\times}\mathcal{S}$ tensor, I is an $\mathcal{S}{\times}\mathcal{A}{\times}\mathcal{S}{\times}\mathcal{A}$ tensor, T is an $\mathcal{S}{\times}\mathcal{A}{\times}\mathcal{S}$ tensor, $\Pi^{\mathcal{S}}$ is an $\mathcal{S}{\times}\mathcal{S}{\times}\mathcal{A}$ tensor, Φ is an $\mathcal{S}{\times}\Omega$ tensor, W^{Π} is an $\Omega{\times}\mathcal{S}{\times}\mathcal{A}$ tensor, and $R^{\mathcal{SA}}$ is an $\mathcal{S}{\times}\mathcal{A}$ tensor.

Lastly, we briefly note that one can get the V_π^λ values by replacing W in the above equation with W^{Π} and using a double contraction:

$$V_\pi^\lambda = W^{\Pi}{:}\left(I - \gamma T\left(\lambda\Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right)\right)^{-1}{:}R^{\mathcal{SA}}.$$

We can confirm that $V_\pi^\lambda = \sum_a \pi_{\omega,a} Q_{\omega,a}$, by rewriting the expression on the right as follows:

$$\begin{aligned}
\sum_a \pi_{\omega,a} Q_{\omega,a} &= \sum_a \pi_{\omega,a} \sum_{s,a'} W^{\mathcal{A}}_{\omega,a,s,a'} B_{s,a'} \\
&= \sum_a \pi_{\omega,a} \sum_s W^{\mathcal{A}}_{\omega,a,s,a} B_{s,a} \\
&= \sum_a \pi_{\omega,a} \sum_s W_{\omega,s} B_{s,a} \\
&= \sum_{s,a} \underbrace{\pi_{\omega,a} W_{\omega,s}}_{W^{\Pi}_{\omega,s,a}} B_{s,a} \\
&= V_\pi^\lambda,
\end{aligned}$$

where $B = \left(I - \gamma T\left(\lambda\Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right)\right)^{-1}{:}R^{\mathcal{SA}}$ is an $\mathcal{S}\times\mathcal{A}$ tensor.

B Proof of Theorem 1 (Almost All)

In this section we prove Theorem 1, that there is either a λ-discrepancy for almost all policies or for no policies. Fix λ and λ'. Recall that we define the λ-discrepancy as follows:

$$\Lambda_\pi := \left\| Q_\pi^\lambda - Q_\pi^{\lambda'} \right\|_{2,\pi} = \left\| \left(W \left(K_\pi^\lambda - K_\pi^{\lambda'} \right) : R^{\mathcal{SA}} \right) \cdot w_\pi \right\|_2,$$

where $K_\pi^\lambda = \left(I - \gamma T \left(\lambda \Pi^{\mathcal{S}} + (1-\lambda) \Phi W^\Pi \right) \right)^{-1}$ and w_π is a weight vector of probabilities dependent on π defined as $w_\pi(\omega, a) = (1, \pi(a|\omega))$. Let Y be the largest open set in the space of stochastic $\Omega \times \mathcal{A}$ matrices, considered as a subset of $\mathbb{R}^{|\Omega|(|\mathcal{A}|-1)}$. Now consider the λ-discrepancy as a function of the policy π. In other words, we define

$$\begin{aligned} \Lambda : Y &\to \mathbb{R}, \\ \pi &\mapsto Q_\pi^\lambda - Q_\pi^{\lambda'}. \end{aligned}$$

Let X be an open subset of $\mathbb{R}^n$. We say that a function $f : X \to \mathbb{R}$ is real analytic on X if for all $x \in X$, f can be written as a convergent power series in some neighborhood of x. For this proof, we will utilize the following facts:

1. the composition of analytic functions is analytic [Krantz and Parks, 2002],
2. the quotient of two analytic functions is analytic where the denominator is nonzero,
3. a real analytic function on a domain X is either identically zero or only zero on a set of measure zero [Mityagin, 2020].

We will also use the fact that for an invertible matrix A, each entry A_{ij}^{-1} is analytic in the entries of A where the entries of A yield a nonzero determinant. We can prove this by first writing $A^{-1} = \det(A)^{-1} \operatorname{adj}(A) = \det(A)^{-1} \operatorname{cof}(A)^\top$ where $\operatorname{adj} A$ is the adjugate of A and $\operatorname{cof}(A)$ is the cofactor matrix of A. Each entry of the cofactor matrix is a cofactor that is polynomial in the entries of A, and is therefore analytic in them. Therefore, each entry of A^{-1} is the quotient of two analytic functions and is therefore analytic except where $\det A = 0$.

Next, we will show that Λ is an analytic function. Note that the variable terms in the equation are W, W^Π, $R^{\mathcal{SA}}$, T, $\Pi^{\mathcal{S}}$, and Φ. Of these, $R^{\mathcal{SA}}$, T, and Φ are constant with respect to π. $\Pi^{\mathcal{S}}_{ilj} = \sum_k \mathbb{1}[i = l] \Phi_{ik} \pi_{kj}$, so each entry of $\Pi^{\mathcal{S}}$ is analytic on Y in the entries of π. Likewise, $P_{ij} = \sum_{k,a} \Phi_{ik} \pi_{ka} T_{iaj}$ is analytic on Y. Therefore, the state-occupancy counts $c = p_0 + \gamma P^\top p_0 + \gamma^2 (P^\top)^2 p_0 + \cdots = (I - \gamma P^\top)^{-1} p_0$, where p_0 contains the initial state probabilities, are the composition of analytic functions and thus analytic on Y. $W_{\omega s} = \frac{\Phi_{s\omega} c_s}{\sum_{s'} \Phi_{s'\omega} c_{s'}}$ is analytic on Y for the same reason, and the denominator of $W_{\omega s}$, $\sum_{s'} \Phi_{s'\omega} c_{s'}$, is nonzero for all observations able to be observed with nonzero probability. Finally, Λ is then a composition of analytic functions on Y and thus analytic itself.

To handle the norm weighting, we note that w_π is analytic in π as $w_\pi = (1, \pi(a|\omega))$, and the dot product of w_π with Λ is also analytic. Now, we use the fact mentioned above that the zero set of a nontrivial analytic function is of measure zero. Therefore, the zero set of $\Lambda \cdot w_\pi$ is either zero for all policies or zero only on a set of measure zero. To finish, we note that because norms are positive definite, $\Lambda_\pi = 0$ if and only if $\Lambda \cdot w_\pi = 0$, so this result extends to the normed λ-discrepancy as well.

C Proof of Theorem 2 (Block MDP)

In this section, we prove Theorem 2 concerning when the system is a block MDP. Recall that in Equation 6 we define $K_\pi^\lambda = \left(I - \gamma T\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right)\right)^{-1}$. Suppose $K_\pi^\lambda = K_\pi^{\lambda'}$. Then $\gamma T\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right) = \gamma T\left(\lambda' \Pi^{\mathcal{S}} + (1-\lambda')\Phi W^{\Pi}\right)$ as matrix inverses are unique. We can rewrite this as $(\lambda - \lambda')\Pi^{\mathcal{S}} - (\lambda - \lambda')\Phi W^{\Pi} = (\lambda - \lambda')(\Pi^{\mathcal{S}} - \Phi W^{\Pi}) = 0$. This implies that either $\lambda = \lambda'$ or $\Pi^{\mathcal{S}} = \Phi W^{\Pi}$.

Recall the definition of $\Pi^{\mathcal{S}}$ and W^{Π} in Section A. Expanding the equation $\Pi^{\mathcal{S}} = \Phi W^{\Pi}$ using these definitions, we have that

$$\Pi^{\mathcal{S}}_{s,s',a} = \mathbb{1}[s = s'](\Phi\pi)_{s,a} = \mathbb{1}[s = s']\sum\nolimits_{\omega} \Pr(\omega|s)\Pr(a|\omega);$$
$$(\Phi W^{\Pi})_{s,s',a} = \sum_{\omega} \Phi_{s,\omega} W^{\Pi}_{\omega,s',a} = \sum_{\omega} \Pr(\omega|s)\Pr(a|\omega)\Pr(s'|\omega).$$

Then by setting the rightmost expression in each equation equal and simplifying with the indicator function, we have that for all i, j, k,

$$\sum\nolimits_{\omega} \Pr(\omega|s_i)\Pr(a_k|\omega)\Pr(s_j|\omega) = \begin{cases} \sum_{\omega} \Pr(\omega|s_i)\Pr(a_k|\omega) & i = j \\ 0 & i \neq j \end{cases}.$$

We will first consider the case where $i \neq j$. We have that for all $i \neq j$ and all k, $\sum_{\omega} \Pr(\omega|s_i)\Pr(a_k|\omega)\Pr(s_j|\omega) = 0$. Because each term in the sum is nonnegative, this is equivalent to the statement that for all $i \neq j$, all k, and all ω, $\Pr(\omega|s_i)\Pr(a_k|\omega)\Pr(s_j|\omega) = 0$. Now note that for all observations ω, there exists some k' such that $\Pr(a_{k'}|\omega) > 0$. Therefore, we have that for all $i \neq j$ and all ω, there exists a k' such that $\Pr(a_{k'}|\omega) > 0$. This implies that for all $i \neq j$ and all ω, $\Pr(\omega|s_i)\Pr(a_{k'}|\omega)\Pr(s_j|\omega) = 0$ and thus $\Pr(\omega|s_i)\Pr(s_j|\omega) = 0$. This means that if state s_i produces an observation ω, then ω cannot be produced by any other reachable state $s_j \neq s_i$, where two states are said to be *reachable* if there exists a sequence of actions sampled from the policy that enable the agent to reach state s_i from s_j with nonzero probability. In other words, each observation uniquely identifies the hidden state, and the POMDP is a block MDP with corresponding Markovian observations.

Next, we consider the case where $i = j$. We have that for all i, k, $\sum_{\omega}\Pr(\omega|s_i)\Pr(a_k|\omega)(1 - \Pr(s_i|\omega)) = 0$. Because each term is nonnegative, this is equivalent to $\Pr(\omega|s_i)\Pr(a_k|\omega)(\Pr(s_i|\omega) - 1) = 0$. Because we again have that for all observations there exists an action $a_{k'}$ with nonzero probability, this means we can choose $k = k'$ to find $\Pr(\omega|s_i) = 0$ or $\Pr(s_i|\omega) = 1$ for all ω, s_i. This means that either the state s_i does not produce an observation ω, or the observation ω uniquely determines which state the agent is in. For all ω and $i = j$, either $\Pr(\omega|s_i) = 0$ or $\Pr(s_j|\omega) = 1$, so if we restrict our focus to the set $\mathcal{O}(s_i) := \{\omega \in \Omega : \Pr(\omega|s_i) > 0\}$, we see that $\sum_{\omega \in \mathcal{O}(s_i)} \Pr(\omega|s_i)\Pr(s_j|\omega) = \sum_{\omega \in \mathcal{O}(s_i)} \Pr(\omega|s_i) \cdot 1 = 1$. For the remaining $\omega \notin \mathcal{O}(s_i)$, $\Pr(\omega|s_i) = 0$.

To recap, the first case tells us that for all ω and $i \neq j$, $\sum_{\omega \in \Omega} \Pr(\omega|s_i)\Pr(s_j|\omega) = 0$. The second case tells us that for all ω and $i = j$, $\sum_{\omega \in \Omega} \Pr(\omega|s_i)\Pr(s_j|\omega) = 1$. Combining both these cases, we see that $\sum_{\omega} \Pr(\omega|s_i)\Pr(s_j|\omega) = \mathbb{1}[s_i = s_j]$, which we can write more succinctly as: $\Phi W = I^{\mathcal{S}}$. We call ΦW the state confusion matrix. For block MDPs, observations cause no confusion about which state the agent is in.

Lastly, by going backwards through the proof, we see that the converse is also true. If the system is a block MDP, then $\Pi^{\mathcal{S}} = \Phi W^{\Pi}$ and so $K_\pi^\lambda = K_\pi^{\lambda'}$.

D Details Leading to Equation 7

In this section, we provide more detail on how we derived Equation 7, a condition for the λ-discrepancy to vanish.

Let $K_\pi^\lambda = \left(I - \gamma T\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right)\right)^{-1}$. $Q_\pi^\lambda = Q_\pi^{\lambda'}$ iff $W(K_\pi^\lambda - K_\pi^{\lambda'}) : R^{\mathcal{SA}} = 0$. Expanding K_π^λ and $K_\pi^{\lambda'}$ into power series, we have

$$\begin{aligned}
0 = &(WR^{\mathcal{SA}} + \gamma WT\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right) : R^{\mathcal{SA}} + \\
&\quad \gamma^2 WT\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right) T\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right) : R^{\mathcal{SA}} + \ldots) \\
&- (WR^{\mathcal{SA}} + \gamma WT\left(\lambda' \Pi^{\mathcal{S}} + (1-\lambda')\Phi W^{\Pi}\right) : R^{\mathcal{SA}} + \\
&\quad \gamma^2 WT\left(\lambda' \Pi^{\mathcal{S}} + (1-\lambda')\Phi W^{\Pi}\right) T\left(\lambda' \Pi^{\mathcal{S}} + (1-\lambda')\Phi W^{\Pi}\right) : R^{\mathcal{SA}} + \ldots).
\end{aligned}$$

We observe that it is potentially possible to get cases of zero λ-discrepancy from pairs of terms in this equation cancelling out. Concretely, this means:

$$\begin{aligned}
WR^{\mathcal{SA}} - WR^{\mathcal{SA}} &= 0 \\
WT\left(\lambda \Pi^{\mathcal{S}} + (1-\lambda)\Phi W^{\Pi}\right) : R^{\mathcal{SA}} = WT\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : R^{\mathcal{SA}} &= 0 \\
WT\Pi^{\mathcal{S}} : T\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : T\Phi W^{\Pi} : R^{\mathcal{SA}} &= 0 \\
&\vdots
\end{aligned}$$

This can occur in one particularly nice way if $\lambda = 0$ and $\lambda' = 1$. In this case, there is no λ-discrepancy precisely when

$$\begin{aligned}
0 = &\left(WR^{\mathcal{SA}} + \gamma WT\Pi^{\mathcal{S}} : R^{\mathcal{SA}} + \gamma^2 WT\Pi^{\mathcal{S}} : T\Pi^{\mathcal{S}} : R^{\mathcal{SA}} + \ldots\right) \\
&- \left(WR^{\mathcal{SA}} + \gamma WT\Phi W^{\Pi} : R^{\mathcal{SA}} + \gamma^2 WT\Phi W^{\Pi} : T\Phi W^{\Pi} : R^{\mathcal{SA}} + \ldots\right) \\
&= \gamma\left(WT\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : R^{\mathcal{SA}}\right) \\
&+ \gamma^2\left(WT\Pi^{\mathcal{S}} : T\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : T\Phi W^{\Pi} : R^{\mathcal{SA}}\right) \\
&+ \ldots
\end{aligned} \tag{9}$$

This occurs when each pair of n-step return terms cancel, or

$$\begin{aligned}
WT\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : R^{\mathcal{SA}} &= 0 \\
WT\Pi^{\mathcal{S}} : T\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : T\Phi W^{\Pi} : R^{\mathcal{SA}} &= 0 \\
&\vdots
\end{aligned}$$

Thus we see that when Equation 7 holds for all initial observations ω^0 and actions a^0, and all horizons k, all pairs of terms in the power series will cancel, and the λ-discrepancy will be zero.

An interesting fact is the converse also holds if $\Lambda_{\mathcal{P},\pi}^{0,1} = 0$ for all discount factors γ.

The only way that LD = 0 for all γ is for each pair of terms in the power series to cancel. Consider what happens if γ is not fixed with the POMDP, but is instead allowed to vary. In that case, we have that the function f from γ to the λ-discrepancy is an analytic function of γ:

$$\begin{aligned}
f : \gamma \mapsto &\gamma\left(WT\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : R^{\mathcal{SA}}\right) \\
&+ \gamma^2\left(WT\Pi^{\mathcal{S}} : T\Pi^{\mathcal{S}} : R^{\mathcal{SA}} - WT\Phi W^{\Pi} : T\Phi W^{\Pi} : R^{\mathcal{SA}}\right) \\
&+ \ldots
\end{aligned} \tag{10}$$

Therefore, following an approach similar to Appendix B, we know that either f is everywhere 0, or it is nonzero almost everywhere. This means that either

1. The λ-discrepancy is 0 for all values of γ.
2. The λ-discrepancy is nonzero for almost all values of γ.

If f is identically 0, then each coefficient of γ in the power series expansion of the λ-discrepancy must be 0, which implies Equation 7. Therefore, the condition is equivalent to the λ-discrepancy being 0 in this case.

If f is nonzero almost everywhere, then the λ-discrepancy can only be zero on a set of γ of measure 0. Therefore, for almost all γ, the condition holds if and only if the λ-discrepancy is actually 0.

In other words, if $\Lambda^{0,1}_{\mathcal{P},\pi} = 0$, either the state transitions ($T\Pi^{\mathcal{S}}$) and the projected state transitions ($T\Phi W^{\Pi}$) predict the same rewards at all time horizons, or they produce different predictions that precisely cancel for the specific discount factor γ that the environment happens to use. That is to say, Λ would be non-zero if the environment had almost any other discount factor.

E Memory Augmentation and Norm Details

In this section, we clarify theoretical details in augmenting a POMDP with memory and the λ-discrepancy norm.

E.1 Memory-Augmented POMDP

As referenced in Section 4, here we will explain how to define a memory-augmented POMDP from a base POMDP $(\mathcal{S}, \mathcal{A}, T, R, \Omega, \Phi, \gamma)$. Given a set of memory states $\mathcal{M}$, we will augment the POMDP as follows:

$$
\begin{aligned}
\mathcal{S}_\mathcal{M} &:= \mathcal{S} \times \mathcal{M} \\
\mathcal{A}_\mathcal{M} &:= \mathcal{A} \times \mathcal{M} \\
\Omega_\mathcal{M} &:= \Omega \times \mathcal{M} \\
T_\mathcal{M} &: \mathcal{S}_\mathcal{M} \times \mathcal{A}_\mathcal{M} \to \Delta\mathcal{S}_\mathcal{M}; && (s_0, m_0, a_1, m_1) \mapsto T(\cdot|s_0, a_1) \times \delta_{m_1}(\cdot) \\
R^{\mathcal{S}_\mathcal{M}\mathcal{A}_\mathcal{M}} &: \mathcal{S}_\mathcal{M} \times \mathcal{A}_\mathcal{M} \to \mathbb{R}; && (s_0, m_0, a_1, m_1) \mapsto R(s_0, a_1) \\
\Phi_\mathcal{M} &: \mathcal{S}_\mathcal{M} \to \Delta\Omega_\mathcal{M}; && (s_0, m_0) \mapsto \Phi(\cdot \mid s_0) \times \delta_{m_0}(\cdot) \\
\gamma_\mathcal{M} &:= \gamma,
\end{aligned}
$$

where $\delta_{x_0}(\cdot)$ is the discrete point distribution centered at x_0, and $\times$ between two distributions denotes the product distribution. To demonstrate what this means, the notation above is equivalent to seeing $T_\mathcal{M}$ instead as a function mapping to $[0, 1]$ and defining $T_\mathcal{M} : \mathcal{S}_\mathcal{M} \times \mathcal{A}_\mathcal{M} \times \mathcal{S}_\mathcal{M} \to [0, 1], (s_0, m_0, a_1, m_1, s_2, m_2) \mapsto T(s_2|s_0, a_1)\mathbb{1}[m_1 = m_2]$, where a memory-augmented action (a_1, m_1) means to take action a_1 and set the memory state to m_1.

This augmentation scheme uses the memory states $\mathcal{M}$ in three ways: as augmentations of states, actions, and observations. The state augmentation concatenates the environment state $\mathcal{S}$ with the agent's internal memory state $\mathcal{M}$. Meanwhile, the action augmentation $\mathcal{A}_\mathcal{M}$ provides the agent with a means of managing its internal memory state. Together, these augmentations allow writing the augmented transition dynamics $T_\mathcal{M}$, which are defined so as to preserve the underlying state-transition dynamics T while allowing the agent full control to select its desired next memory state. The observation augmentation $\Omega_\mathcal{M}$ provides the agent with additional context with which to make policy decisions, and the observation function $\Phi_\mathcal{M}$ preserves the original behavior of the observation function Φ while giving the agent complete information about the internal memory state.

We define an augmented policy $\pi_\mathcal{M}$ and its tensor counterparts $\Pi_\mathcal{M}$ and $\Pi^{\mathcal{S}_\mathcal{M}}$ over observations and states, respectively, as follows:

$$
\begin{aligned}
\pi_\mathcal{M} &: \Omega_\mathcal{M} \to \Delta\mathcal{A}_\mathcal{M}; \\
\Pi_\mathcal{M} &\text{ is of size } (\Omega_\mathcal{M} \times \Omega_\mathcal{M} \times \mathcal{A}_\mathcal{M}); \\
\Pi^{\mathcal{S}_\mathcal{M}} &\text{ is of size } (\mathcal{S}_\mathcal{M} \times \mathcal{S}_\mathcal{M} \times \mathcal{A}_\mathcal{M});
\end{aligned}
$$

where

$$
\begin{aligned}
\Pi_\mathcal{M}(\omega_0, m_0, \omega_1, m_1, a_2, m_2) &= \mathbb{1}[(\omega_0, m_0) = (\omega_1, m_1)]\pi_\mathcal{M}(a_2, m_2|\omega_1, m_1); \\
\Pi^{\mathcal{S}_\mathcal{M}}(s_0, m_0, s_1, m_1, a_2, m_2) &= \mathbb{1}[(s_0, m_0) = (s_1, m_1)] \sum_{(\omega,m)\in\Omega_\mathcal{M}} \Phi_\mathcal{M}(\omega, m|s_0, m_0)\pi_\mathcal{M}(a_2, m_2|\omega, m).
\end{aligned}
$$

We extend the $\Pr(s|\omega)$ tensors to $\mathcal{S}_\mathcal{M}$ and $\Omega_\mathcal{M}$ as follows:

$$
\begin{aligned}
W_\mathcal{M} &\text{ is of size } (\Omega_\mathcal{M} \times \mathcal{S}_\mathcal{M}); \\
W^\Pi_\mathcal{M} &\text{ is of size} (\Omega_\mathcal{M} \times \mathcal{S}_\mathcal{M} \times \mathcal{A}_\mathcal{M});
\end{aligned}
$$

where

$$
\begin{aligned}
W_\mathcal{M}(\omega_0, m_0, s_1, m_1) &= \mathbb{1}[m_0 = m_1] \Pr(s_1|\omega_0); \\
W^\Pi_\mathcal{M}(\omega_0, m_0, s_1, m_1, a_2, m_2) &= \mathbb{1}[m_0 = m_1] \Pr(s_1|\omega_0)\pi_\mathcal{M}(a_2, m_2|\omega_0, m_0).
\end{aligned}
$$

One appealing aspect of the above augmentation process is that it makes the agent's control over its memory function explicit, via "memory-augmented actions" $\mathcal{A}_\mathcal{M} = \mathcal{A} \times \mathcal{M}$. However, it is perhaps a bit unusual to combine the agent's policy and memory together. An equivalent and perhaps more intuitive formulation decomposes this action-memory policy into a separate action policy $\pi_\mu : \Omega \times \mathcal{M} \to \Delta\mathcal{A}$ and a memory function $\mu : \Omega \times \mathcal{M} \times \mathcal{A} \to \Delta\mathcal{M}$:

$$\pi_\mathcal{M}(a, m' \mid \omega, m) = \pi_\mu(a \mid \omega, m)\mu(m' \mid \omega, m, a).$$

Note that we define memory functions as distributions of next memory states for generality.

There are two ways to view this augmentation procedure. In the first view, outlined above, the agent selects both an action a and a next memory state m', and the environment $T_\mathcal{M}$ responds by sampling the next state $s' \sim T(\cdot|s, a)$ while deterministically setting the next memory state to m'. In the second (equivalent) view, the agent selects only action a, with the agent's memory function behavior folded into the transition dynamics T_μ. This view leads to the following POMDP quantities:

$$\begin{aligned} T_\mu &: \mathcal{S}_\mathcal{M} \times \mathcal{A} \to \Delta\mathcal{S}_\mathcal{M}; & (s_0, m_0, a_0) &\mapsto T(\cdot \mid s_0, a_0)\mu_\mathcal{S}(\cdot \mid s_0, m_0, a_0), \\ R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}} &: \mathcal{S}_\mathcal{M} \times \mathcal{A} \to \mathbb{R}; & (s_0, m_0, a_0) &\mapsto R(s_0, a_0), \end{aligned}$$

where $\mu_\mathcal{S}(\cdot|s_0, m_0, a_0) := \sum_{\omega\in\Omega} \Phi(\omega|s_0)\mu(\cdot|\omega_0, a_0, m_0)$ is the effective memory function for states.

The μ-specific policy and weight tensors are defined as follows:

$$\begin{aligned} \Pi_\mu &\text{ is of size } (\Omega_\mathcal{M} \times \Omega_\mathcal{M} \times \mathcal{A}); \\ \Pi_\mu^{\mathcal{S}_\mathcal{M}} &\text{ is of size } (\mathcal{S}_\mathcal{M} \times \mathcal{S}_\mathcal{M} \times \mathcal{A}); \\ W_\mu^\Pi &\text{ is of size} (\Omega_\mathcal{M} \times \mathcal{S}_\mathcal{M} \times \mathcal{A}); \end{aligned}$$

where

$$\begin{aligned} \Pi_\mu(\omega_0, m_0, \omega_1, m_1, a_2) &= \mathbb{1}[(\omega_0, m_0) = (\omega_1, m_1)]\pi_\mu(a_2|\omega_1, m_1); \\ \Pi_\mu^{\mathcal{S}_\mathcal{M}}(s_0, m_0, s_1, m_1, a_2) &= \mathbb{1}[(s_0, m_0) = (s_1, m_1)] \sum_{(\omega,m)\in\Omega_\mathcal{M}} \Phi_\mathcal{M}(\omega, m|s_0, m_0)\pi_\mu(a_2|\omega, m); \\ W_\mu^\Pi(\omega_0, m_0, s_1, m_1, a_2) &= \mathbb{1}[m_0 = m_1] \Pr(s_1|\omega_0)\pi_\mu(a_2|\omega_0, m_0). \end{aligned}$$

To recap, the first augmentation scheme transforms the POMDP $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \Omega, T, R^{\mathcal{S}\mathcal{A}}, \Phi, p_{\mathcal{M}_0}, \gamma)$ into $\mathcal{P}_\mathcal{M} = (\mathcal{S}_\mathcal{M}, \mathcal{A}_\mathcal{M}, \Omega_\mathcal{M}, T_\mathcal{M}, R^{\mathcal{S}_\mathcal{M}\mathcal{A}_\mathcal{M}}, \Phi_\mathcal{M}, p_{\mathcal{M}_0}, \gamma)$. The second view forms an equivalent POMDP $\mathcal{P}_\mu = (\mathcal{S}_\mathcal{M}, \mathcal{A}, \Omega_\mathcal{M}, T_\mu, R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}}, \Phi_\mathcal{M}, \gamma)$, which folds the agent's memory function μ into the transition dynamics.

Since both of these are valid POMDPs and we already have a general expression for the TD(λ) value function of a POMDP, we can immediately write down the corresponding value functions for each of these types of augmentation. For $\mathcal{P}_\mathcal{M}$, we have:

$$Q^\lambda_{\pi_\mathcal{M}} = W_\mathcal{M}\Big(I^{\mathcal{S}_\mathcal{M}\mathcal{A}_\mathcal{M}} - \gamma T_\mathcal{M}\big(\lambda\Pi^{\mathcal{S}_\mathcal{M}} + (1-\lambda)\Phi_\mathcal{M} W^\Pi_\mathcal{M}\big)\Big)^{-1} : R^{\mathcal{S}_\mathcal{M}\mathcal{A}_\mathcal{M}}; \tag{11}$$

and for $\mathcal{P}_\mu$, we have:

$$Q^\lambda_{\pi_\mu} = W_\mathcal{M}\Big(I^{\mathcal{S}_\mathcal{M}\mathcal{A}} - \gamma T_\mu\big(\lambda\Pi^{\mathcal{S}_\mathcal{M}}_\mu + (1-\lambda)\Phi_\mathcal{M} W^\Pi_\mu\big)\Big)^{-1} : R^{\mathcal{S}_\mathcal{M}\mathcal{A}}_\mu. \tag{12}$$

We provide pseudocode for taking this memory-Cartesian product of a given POMDP in Appendix F.3, Algorithm 3. The pseudocode uses the view that aligns with Equation (12), since that view matches our implementation.

E.2 λ-Discrepancy Norm Weighting

The λ-discrepancy as introduced in Definition 1 contains a weighted norm over the observations and actions of the decision process. There are many choices of norm and weighting scheme. We use an L^2 norm to highlight the connection to mean-squared action-value error. For the weighting scheme, we weight actions according to the policy for each observation, and we weight observations uniformly. More precisely, the weighting assigns the (ω, a) entry the weight $(1, \pi(a|\omega))$. We also considered weighting observations according to their discounted visitation frequency, but found that this led to worse performance during memory optimization.

F Environments and Experimental Details for Closed-Form Memory Optimization

We now describe the proof-of-concept experiments from Section 4 involving closed-form memory optimization on a range of small-scale classic POMDPs. The environments are: T-maze [Bakker, 2001], the Tiger problem [Cassandra et al., 1994], Paint [Kushmerick et al., 1995], Cheese Maze, Network, Shuttle [Chrisman, 1992], and the partially observable version of the 4×3 maze [Parr and Russell, 1995].

We first begin by detailing the small-scale POMDPs we use. Then we describe the algorithm used to calculate closed-form λ-discrepancy gradients, as well as procedures for policy improvement after learning a memory function. Apart from the T-maze, all other POMDPs used in Section 4 were taken from pre-defined POMDP definitions [Cassandra, 2003].

We made one slight modification to the Tiger environment that preserves the original environment behavior but adapts the domain specification to match our formalism such that observations are only a function of state. The original Tiger domain used a hand-coded initial belief distribution that was uniform over the two states `L`/`R`, and did not emit an observation until after the first action was selected. Thereafter, the observation function was action-dependent, with state-action pair $(\texttt{L}, \texttt{listen})$ emitting observations `left` and `right` with probability 0.85 and 0.15 respectively, and other actions $(\texttt{L}, *)$ emitting uniform observations and returning to the initial belief distribution. Since our agent does not have access to the set of states, it cannot use an initial belief distribution. To achieve the same behavior, we modified the domain by splitting each state `L`/`R` into an initial state $\texttt{L}_1/\texttt{R}_1$ that always emits an `initial` observation, and a post-listening state $\texttt{L}_2/\texttt{R}_2$ that uses the $0.85/0.15$ probabilities. We visualize these changes in Figure 6. This type of modification is always possible for finite POMDPs and does not change the underlying dynamics.

F.1 T-maze Details

We use T-maze with corridor length 5 as an instructive example (see Figure 1). The environment has 18 underlying MDP states: one initial state for each reward configuration (reward is either up or down), five for each corridor, one for each junction, and finally one for each terminal state.[7] There are 5 observations in this environment - one for each of the initial states, a corridor observation shared by all corridor states, a junction observation shared by both junction states, and a terminal observation. The action space is defined by movement in the cardinal directions. If the agent tries to move into a wall, it remains in the current state. From the junction state, the agent receives a reward of $+4$ for going north, and -0.1 for going south in the first reward configuration. The rewards are flipped for the second configuration. The environment has a discount rate of $\gamma = 0.9$.

[7]Technically, we group the four terminal states into a single state for conciseness, which is functionally equivalent.

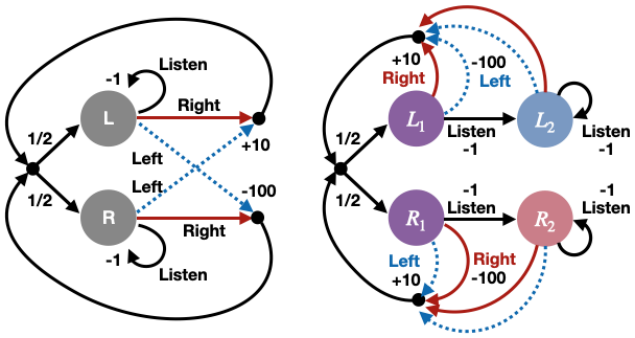

Figure 6: Visualizations of the Tiger POMDP. In the original version (left) the observation function was action-dependent, whereas in our modified version (right) observations only depend on state. The state color for the domain on the right represents the distinct state-dependent observation functions: purple states use the `initial` observation, while the other states are biased towards either `left` (blue) or `right` (red) observations with probability 0.85.

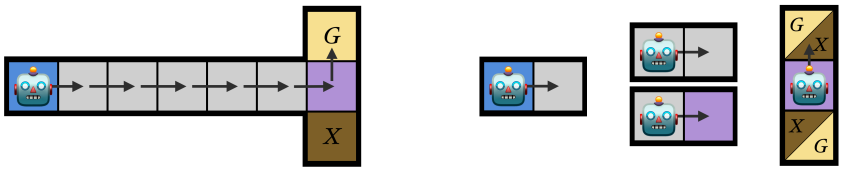

Figure 7: Visualizations of value functions computed using MC (left) and TD (right). MC averages over entire trajectories, so it can associate the blue observation with the upward goal. By contrast, TD computes value by bootstrapping; its value estimates for subsequent observations ignore any prior history.

This environment makes it easy to see the differences between MC and TD approaches to value function estimation. We visualize these differences in Figure 7. MC computes the average value for each observation by averaging the return over all trajectories starting from that observation. By contrast, TD averages over the 1-step observation transition dynamics and rewards, and bootstraps off the value of the next observation. For a policy that always goes directly down the corridor and north at the junction, this leads to an average (undiscounted) return for the blue observation of $+4$ with MC and $(4 - 0.1)/2 = 1.95$ with TD.

F.2 Analytical Memory-Learning and Policy Improvement Algorithm

Algorithm 1 describes our memory optimization procedure, which reduces the λ-discrepancy to learn a memory function, then learns an optimal memory-augmented policy.

Algorithm 1 Memory Optimization with Value Improvement

Input: Randomly initialized policy parameters θ_π, where $\Pi = \mathrm{softmax}(\theta_\pi)$, randomly initialized memory parameters θ_μ, POMDP parameters $\mathcal{P} := (\mathcal{S}, \mathcal{A}, T, R^{\mathcal{SA}}, \Omega, \Phi, p_0, \gamma)$, number of memory improvement steps $n_{\text{steps},\mu}$, number of policy iteration steps $n_{\text{steps},\pi}$, learning rate $\alpha \in [0, 1]$, and number of initial random policies n.
Output: Optimized memory-augmented policy parameters θ_{π_μ} and memory parameters θ_μ.

// Initialize random policies and select policy to fix for memory learning.
$\{\theta_0, \ldots, \theta_{n-1}\} \leftarrow$ `randomly_init_n_policies`(n)
$\theta_\pi \leftarrow$ `select_argmax_lambda_discrepancy`$(\theta_0, \ldots, \theta_{n-1}\})$
// Repeat policy over all memory states.
$\theta_{\pi_\mu} \leftarrow$ `repeat`$(\theta_\pi, |\mathcal{M}|)$
// Improve memory function.
$\theta_\mu \leftarrow$ `memory_improvement`$(\theta_\mu, \theta_{\pi_\mu}, \mathcal{P}, n_{\text{steps},\mu})$
// Augment POMDP with memory function.
$\mathcal{P}_\mu \leftarrow$ `expand_over_memory`$(\mathcal{P}, \theta_\mu)$
// Improve memory-augmented policy over memory-augmented POMDP
$\theta_{\pi_\mu} \leftarrow$ `policy_improvement`$(\theta_{\pi_\mu}, \mathcal{P}_\mu, n_{\text{steps},\pi})$
return $\theta_{\pi_\mu}, \theta_\mu$

In Algorithm 1, the `policy_improvement` function can be any function which improves a parameterized policy. We use an analytical version of the policy gradient [Sutton et al., 2000] algorithm. The `randomly_init_n_policies` function returns n randomly initialized policies, and the `select_argmax_lambda_discrepancy` function picks the one with the largest λ-discrepancy. The `memory_improvement` and `expand_over_memory` functions are defined in Appendices F.2.1 and F.3, respectively.

We have noticed that a larger λ-discrepancy tends to provide a better signal for learning memory functions. Sampling random policies for memory improvement is highly likely to reveal a λ-discrepancy, as suggested by the theory in Section 3.2. For this reason, we consider many random policies ($n = 100$), then use the policy which had maximum λ-discrepancy as the basis for memory optimization.

F.2.1 Memory Improvement Algorithm

In this section we provide pseudocode (Algorithm 2) for the memory-learning algorithm described in Section 4 and used in Algorithm 1. This function takes as input a POMDP $\mathcal{P}$ and memory parameters θ_μ, and minimizes the λ-discrepancy of Definition 1. This minimization is achieved through a gradient descent update computed using the auto-differentiation package JAX [Bradbury et al., 2018].

Algorithm 2 Memory Improvement

Input: Fixed policy parameters θ_{π_μ}, where $\Pi_\mu^{\mathcal{S}_\mathcal{M}}(\mathcal{S}_\mathcal{M} \times \mathcal{S}_\mathcal{M} \times \mathcal{A})$ is the expanded effective policy over state for the policy $\pi_\mu = \mathrm{softmax}(\theta_{\pi_\mu})$ $(\mathcal{S}_\mathcal{M} \times \mathcal{A})$, memory parameters θ_μ, POMDP parameters $\mathcal{P} := (\mathcal{S}, \mathcal{A}, T, R^{\mathcal{SA}}, \Omega, \Phi, p_0, \gamma)$, number of improvement steps $n_{\text{steps},\mu}$, learning rate $\alpha \in [0, 1]$.
Output: Optimized memory parameters θ_μ.

for $i = 0$ **to** $n_{\text{steps},\mu} - 1$ **do**
  // Augment POMDP with memory parameters θ_μ
  $(\mathcal{S}_\mathcal{M}, \mathcal{A}, \Omega_\mathcal{M}, T_\mu, R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}}, \Phi_\mathcal{M}, p_{\mathcal{M}_0}, \gamma) \leftarrow$ `expand_over_memory`$(\mathcal{P}, \theta_\mu)$
  // Compute TD value function (with memory augmentation)
  $Q^0_{\pi_\mu} = W_\mathcal{M}\Big(I - \gamma T_\mu \Phi_\mathcal{M} W_\mu^\Pi\Big)^{-1} : R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}}$
  // Compute MC value function (with memory augmentation)
  $Q^1_{\pi_\mu} = W_\mathcal{M}\Big(I - \gamma T_\mathcal{M} \Pi_\mu^{\mathcal{S}_\mathcal{M}}\Big)^{-1} : R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}}$
  // Calculate the λ-discrepancy
  $\Lambda_{\pi_\mu} = ||Q^0_{\pi_\mu} - Q^1_{\pi_\mu}||_{\pi_\mu,2}$
  // Calculate the gradient of Λ_{π_μ} w.r.t. θ_μ, update memory parameters
  $\theta_\mu \leftarrow$ `update_params`$(\alpha, \theta_\mu, \nabla_{\theta_\mu}\Lambda_{\pi_\mu})$
end for
return θ_μ

Here, `update_params()` is any gradient-descent-like update, such as stochastic gradient descent, Adam, etc. As a note, all parameters θ in these experiments are initialized with a Gaussian distribution, with mean 0 and standard deviation 0.5.

F.3 Memory Cartesian Product

In this section, we define the memory-Cartesian product function, `expand_over_memory()`, used by Algorithms 1 and 2. This function computes the Cartesian product of the POMDP $\mathcal{P}$ and the memory state space $\mathcal{M}$, as described in Appendix E.1.

Algorithm 3 Memory Cartesian Product (`expand_over_memory`)

Input: Memory parameters θ_μ (with corresponding memory function μ), POMDP parameters $\mathcal{P} := (T, R^{\mathcal{SA}}, \Phi, p_0, \gamma)$, number of memory states $|\mathcal{M}|$
// Repeat reward function for each state over each memory $m \in \mathcal{M}$ $(\mathcal{S}_\mathcal{M} \times \mathcal{A})$.
$R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}} \leftarrow$ `repeat_over_states`$(R^{\mathcal{SA}}, |\mathcal{M}|)$
// Calculate transition function memory cross product.
// First, calculate the effective memory function over state $(\mathcal{S} \times \mathcal{A} \times \mathcal{M} \times \mathcal{M})$.
$\mu_\mathcal{S} \leftarrow$ `einsum`$('ij, jklm \rightarrow iklm', \Phi, \mu)$
// Now expand the state transition function to include memory state transitions $(\mathcal{S}_\mathcal{M} \times \mathcal{A} \times \mathcal{S}_\mathcal{M})$.
$T_\mu \leftarrow$ `einsum`$('iljk, lim \rightarrow lijmk', \mu_\mathcal{S}, T)$
// Calculate observation function memory cross product $(\mathcal{S}_\mathcal{M} \times \Omega_\mathcal{M})$. $I_{|\mathcal{M}|}$ is the identity function over $|\mathcal{M}|$.
$\Phi_\mathcal{M} \leftarrow$ `kron`$(\Phi, I_{|\mathcal{M}|})$
// Finally, calculate the initial state distribution $(\mathcal{S}_\mathcal{M})$.
$p_{\mathcal{M}_0} \leftarrow [p_0(s)$ if $m = 0$ else 0 for $s, m \in \mathcal{S}, \mathcal{M}]$
return $\mathcal{P}_\mu = (\mathcal{S}_\mathcal{M}, \mathcal{A}, \Omega_\mathcal{M}, T_\mu, R_\mu^{\mathcal{S}_\mathcal{M}\mathcal{A}}, \Phi_\mathcal{M}, p_{\mathcal{M}_0}, \gamma)$

Note that `einsum` is the Einstein summation, and `kron` is the Kronecker product. The augmented initial state distribution is simply the same distribution as p_0, except with 0 probability mass over all non-zero memory states, since the memory state always initializes to memory state 0.

F.4 Analytical Memory Learning Experiment Details

For all experiments in Section 4, we run memory optimization on the suite of POMDPs with the following hyperparameters. We optimize memory for $n_{\text{steps},\mu} = 20K$ steps and run policy iteration for $n_{\text{steps},\pi} = 10K$ steps. For all gradient-based experiments, we use the Adam optimizer [Kingma and Ba, 2015].

For the belief-state baselines, solutions were calculated using a POMDP solver from the `pomdp-solve` package [Cassandra, 2003]. The performance of the belief-state optimal policy was calculated by taking the dot product between the initial belief state and the maximal alpha vector for that belief state. This returns a metric comparable to the initial state distribution weighted value function norm, which we use as a performance metric for our memory-augmented agents.

The belief-state solution for the 4×3 maze was solved using an epsilon parameter of $\epsilon = 0.01$, due to convergence issues with the environment when utilizing the POMDP solver.

G Scaling the λ-Discrepancy with PPO and RNNs

G.1 Algorithm Details

We build our λ-discrepancy minimizing reinforcement learning algorithm on top of online recurrent PPO. Normally, PPO would have two losses: one for the actor and one for the critic.

$$L_{\text{PPO}}(\theta) = \mathbb{E}_\pi\big[L_{\text{CLIP}}(\theta_{\text{Actor}}, \theta_{\text{RNN}}) - c_V L_V(\theta_{\text{Critic}}, \theta_{\text{RNN}}) + c_{\text{Ent}} H[\pi_\mu(\cdot \mid \omega_t, z_{t-1})]\big]. \quad (13)$$

Here, L_{CLIP} is the clipped surrogate loss for the actor, defined by:

$$L_{\text{CLIP}}(\theta_{\text{Actor}}, \theta_{\text{RNN}}) = \mathbb{E}_\pi\left[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon))\right], \quad (14)$$

where $\rho_t(\theta)$ is the ratio between policy probabilities of the old policy and new policy, $\hat{A}_t$ is the estimated advantage at timestep t, and c_V and L_V are the value coefficient and value function loss, respectively. L_V is the mean-squared TD error, with targets calculated using a truncated version of λ-return [Sutton and Barto, 2018]. Finally, c_{Ent} and $H[\pi_\mu(\cdot \mid \omega_t, z_{t-1})]$ are the entropy coefficient and entropy of the policy π_μ that is parameterized by θ_{RNN} and θ_{Actor}. This term is used for exploration through entropy maximization [Mnih et al., 2016]. For our memoryless baseline, we use a non-recurrent architecture that is a function of only the current observation (and optionally the most recent action).

Our proposed algorithm replaces L_V with $L_{V,\lambda_1,\lambda_2}$, which has three components: two value losses for two separate value heads (each estimating λ-returns for different λs), and our λ-discrepancy auxiliary loss.

$$L_{V,\lambda_1,\lambda_2}(\theta_{\text{Critic}}, \theta_{\text{RNN}}) = \beta L_\Lambda(\theta) + (1-\beta)\left(L_{V,\lambda_1}(\theta_{\text{Critic1}}, \theta_{\text{RNN}}) + L_{V,\lambda_2}(\theta_{\text{Critic2}}, \theta_{\text{RNN}})\right),$$

where $L_\Lambda(\theta)$ is our λ-discrepancy loss, as defined in Equation 8, and $L_{V,\lambda}$ is the λ-return-as-target TD error for a given λ. Here, $\beta \in [0, 1]$ is a hyperparameter trading off between the λ-discrepancy loss and the value losses. Finally, the agent only uses V_{λ_1} as its value estimate for its advantage function.

The computational and memory requirements for PPO with and without the λ-discrepancy auxiliary loss are similar. While the λ-discrepancy loss requires training a second value head, this only adds two additional layers to the critic network and results in a small number of additional parameters (about 12% overhead). As for computation, the addition of this auxiliary loss only adds a small constant factor to the backpropagation of gradients, since we learn everything end-to-end. Overall, the wall-clock time of runs between the baseline algorithm and our algorithm is comparable.

G.2 Environment Details

All four of the environments used for evaluation in Section 5 were all re-implementations of environments used in Silver and Veness [2010], in JAX [Bradbury et al., 2018], allowing for massive hardware acceleration. We now give details of these environments.

G.2.1 Battleship

Partially observable Battleship, a one-player, limited observation variant of the popular two-player board game, was first introduced as a benchmark for partially observable Monte-Carlo planning [Silver and Veness, 2010]. This variant is particularly challenging since the agent has to reason about the unknown positions of the ship and keep track of past shots. The observation space is $\Omega = \{0, 1\}$; at every step, the agent receives a binary observation: 0 if the last shot missed a ship, and 1 if the last shot hit a ship. The state space in this game contains all possible board states, which is astronomically large. The action space is $\mathcal{A} = \{1, \ldots, 10\} \times \{1, \ldots, 10\}$, or all possible grid locations. The agent is only allowed to take valid actions at every step (no position can be selected twice), which is achieved through action masking of the actor. This makes the problem a finite horizon problem, with a horizon at most $10 \times 10 = 100$. The environment terminates when all positions on the grid with a ship are hit. The rewards at every step are -1, with a positive reward of $10 \times 10 = 100$ when all ships are hit/the environment terminates. The discount factor γ is set to 1 here, since we are in the finite horizon setting.

At every environment reset, 4 ships of length $(5, 4, 3, 2)$ are uniformly randomly placed on a 10×10 grid. At every step, to allow for easier learning of all agents, we concatenate the last action to the observation, so that our RNN conditions on the entire history, as opposed to only observations.

G.2.2 Partially Observable PacMan

Partially observable PacMan (a.k.a. *PocMan*), a variant of the popular arcade game, was also introduced to test partially observable Monte-Carlo planning with a simulator [Silver and Veness, 2010]. In this version, the agent can only observe indicators of its surroundings, including walls in its cardinal directions, whether there is a ghost in its line of sight, a power pellet nearby, or food nearby. These highly obscured agent-centric observations require the agent to localize within the map, seek food and power pellets, and avoid ghosts with only local sensor information.

Concretely, the observation space is an 11-dimensional binary vector. The first four values are 1 if there is a wall in each of the four cardinal directions of the agent. The next element is 1 if there is food Manhattan distance 2 or less away from the agent. The next value is 1 if there is a ghost a Manhattan distance 2 or less away. The next four elements are 1 if there is at least one ghost in the line of sight in each of the four cardinal directions. The last element is if the agent currently has a power pellet.

As with Battleship, the state space in PocMan is also very large. The map is a 19×21 sized map defined in the original instantiation of the environment [Silver and Veness, 2010], and the state consists of the agent and ghost locations, as well as the binary status of every dot and power pill. The action space has four actions, corresponding to moving in the four cardinal directions. The agent receives a reward of +200 if it eats a ghost, +10 if it collects a pellet, and +20 if it collects a power pellet. The discount factor is set to $\gamma = 0.95$.

The episode terminates when the agent is killed by a ghost, the agent collects all pellets, or the timer for the environment elapses. At the beginning of each episode, all ghost locations are reset, and the agent is placed in the same fixed starting position. In this setting, we also append the agent's most recent action to its observation.

In our work, we implement this domain by adding the observation function on top of the PacMan environment in the Jumanji reinforcement learning framework [Bonnet et al., 2024].

G.2.3 RockSample

RockSample [Smith and Simmons, 2004], a well-known partially observable benchmark in POMDP planning literature, is a rock-collecting problem simulating a rover scanning potential rock samples and collecting them if they are desirable. RockSample (n, k) is a problem with a grid of size $n \times n$, and k rocks distributed randomly throughout the environment. The two variants we test on have a grid size of 11×11 and 15×15, as well as 11 and 15 rocks in each environment, respectively. When instantiated, each RockSample environment for each seed randomly samples rock positions. At the reset of each environment, each of these rocks are uniformly randomly assigned to be either a good or bad rock. The action space in this environment is $4 + 1 + k = 5 + k$. The first 4 actions correspond to moving in the cardinal directions. Action 5 corresponds to sampling in the current position. The last k actions correspond to checking each of the k rocks. Checking a rock will probabilistically tell the agent the correct parity of the rock, depending on the *half-efficiency distance* d_{hed} and the l_2 distance $d_i(s)$ between the agent's position and the rock being checked, given the current state s:

$$\Pr\left(\text{accurate} \mid s, a = \text{check}_i\right) = \frac{1}{2}\left(1 + 2^{-d_i(s)/d_{\text{hed}}}\right). \tag{15}$$

In this work, we simply set d_{hed} to be the maximum possible distance between any two points in the grid. Traversing to and then sampling a good rock gives a reward of +10. Sampling a bad rock gives a reward of -10. Exiting to the east border of the environment will terminate the environment and result in a reward of +10. After sampling a good rock, sampling the same rock again will result in a reward of -10. This environment has a discount rate of $\gamma = 0.99$

The state space of this environment is all the possible combinations of rock positions, as well as agent positions and rock parities. The agent receives observations of the form of a vector of size $2n + k$ binary values, where the first $2n$ elements are a two-hot representation of the xy-coordinates of the agent. The final k observations are set after choosing a check action, and set a 1 at the position of the checked rock if it appears to be good, and a 0 if it appears to be bad. This observation function is a slight departure from the previous RockSample problem definition [Smith and Simmons, 2004, Silver and Veness, 2010] in two aspects, based on an implementation of RockSample that is used to test RNNs [Tao et al., 2023], as opposed to model-based planning algorithms. The first is that after an agent checks a rock and the agent receives a positive sensor reading, the observation bit corresponding

to this rock is set to 1, and remains 1 until sampled. This makes the memory learning portion of the problem easier, but the agent is still required to remember which rocks it has sampled before, and also deal with the stochasticity of checking. The second change is to not have an explicit negative sensor reading. Instead, after checking a rock, the observation bit corresponding to the sensor reading of the rock remains 0. This makes the problem harder, since the agent has to infer the negative sensor reading, and disambiguate it from a null sensor reading, when the agent takes non-check actions.

G.3 Discounted Return RNN Results

In the main text (Section 5), we follow the common practice of reporting results for undiscounted returns. However, since our optimization objective considers discounted returns, we also include learning curves for discounted returns in Figure 8. For ease of comparison, we also include the undiscounted learning curves (from Figure 5) in Figure 9. Note that Battleship is unchanged, since that domain is finite-horizon and undiscounted.

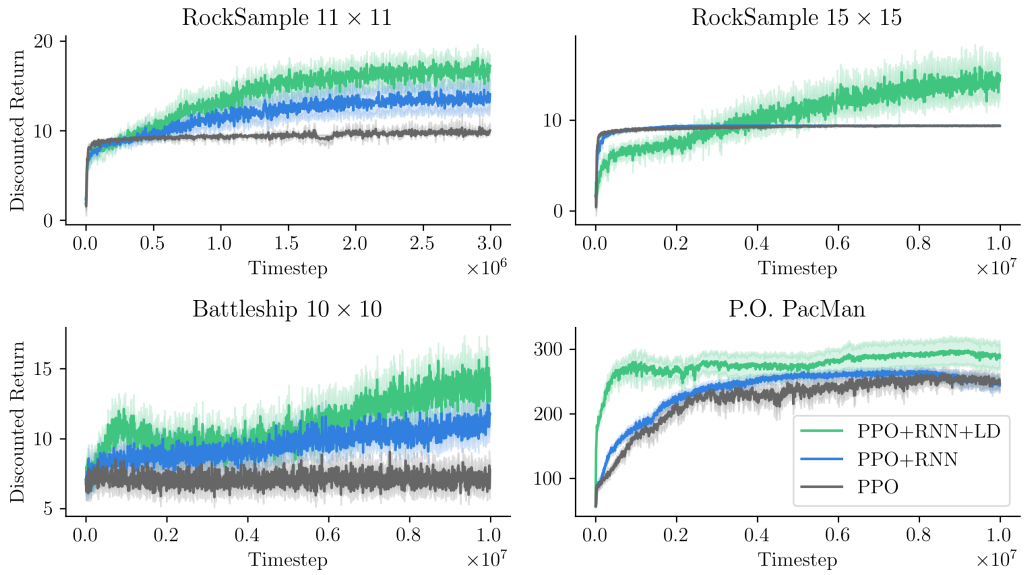

Figure 8: The λ-discrepancy (LD) performance in discounted returns over recurrent (RNN) and memoryless PPO. Learning curves shown are the mean and 95% confidence interval over 30 runs.

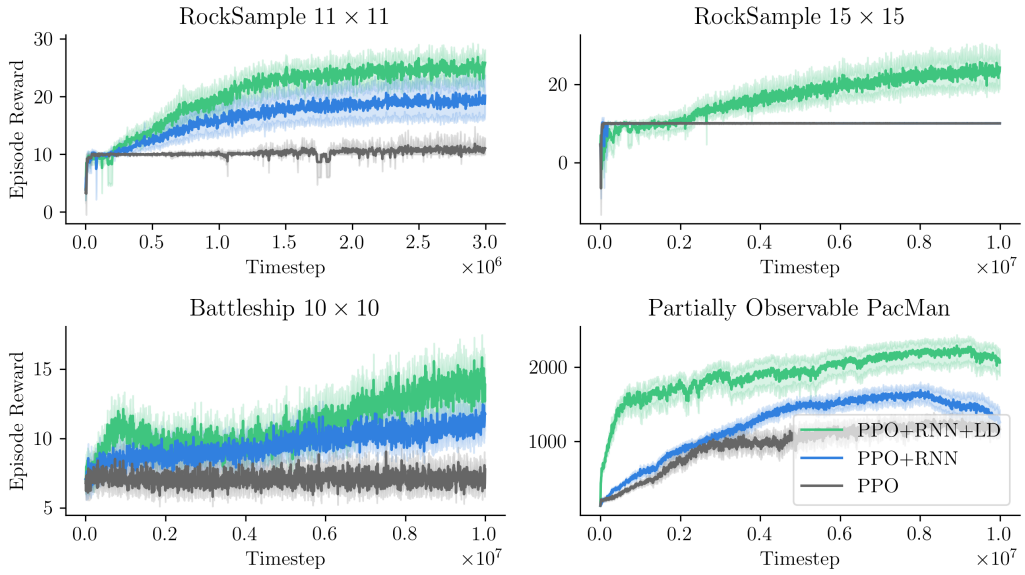

Figure 9: Larger version of Figure 5. The λ-discrepancy (LD) performance (undiscounted returns) over recurrent (RNN) and memoryless PPO. Learning curves shown are the mean and 95% confidence interval over 30 runs.

G.4 Experimental and Hyperparameter Details

Our base PPO algorithm is an online PPO algorithm that trains over a vectorized environment, all parallelized using JAX [Bradbury et al., 2018] and the PureJaxRL batch experimentation library [Lu et al., 2022] with hardware acceleration. The hyperparameter sweep was performed on a cluster of NVIDIA 3090 GPUs, and the best seeds presented were run on one GPU for each algorithm, running for 1 to 12 hours, depending on the domain. Our environment is vectorized over 4 copies, and we use truncated-backpropogation through time [Jaeger, 2002] as our gradient descent algorithm, with a truncation length of 128. Our L_{CLIP} clipping ϵ is set to 0.2. The value loss coefficient is set to $c_V = 0.5$. We also anneal our learning rate over all training steps, and clip gradients when the norm is larger than 0.5 by their global norm [Pascanu et al., 2013].

We now detail the hyperparameters swept across all environments and all algorithms. We do so in Figure 10.

Hyperparameter	
Step size	$[2.5 \times 10^{-3}, 2.5 \times 10^{-4}, 2.5 \times 10^{-5}, 2.5 \times 10^{-6}]$
λ_1	$[0.1, 0.5, 0.7, 0.9, 0.95]$
λ_2 (λ-discrepancy)	$[0.1, 0.5, 0.7, 0.9, 0.95]$
β (λ-discrepancy)	$[0, 0.125, 0.25, 0.5]$

Figure 10: Hyperparameters swept across all algorithms. Rows labelled with λ-discrepancy are hyperparameters swept specific to our algorithm.

We conduct this sweep across all environments for 5 seeds each, and use the highest area under the learning curve (AUC) score in each environment to select the best hyperparameters for each environment. Note, we swept over 10 seeds for recurrent PPO and λ-discrepancy-augmented recurrent PPO for the PocMan environment, due to high variance in returns. After hyperparameter selection, we re-run all algorithms on all environments with the best selected hyperparameters for 30 different seeds to produce the learning curves in Figure 5. We present the best hyperparameters found for both algorithms in Figure 11.

	Step size	λ_1	λ_2	β
Battleship	2.5×10^{-4}	0.1	0.95	0.5
PocMan	2.5×10^{-4}	0.95	0.5	0.5
RockSample $(11, 11)$	2.5×10^{-4}	0.1	0.95	0.5
RockSample $(15, 15)$	2.5×10^{-4}	0.1	0.5	0.25

(a) Best hyperparameters found for λ-discrepancy-augmented recurrent PPO.

	Step size	λ_1
Battleship	2.5×10^{-5}	0.7
PocMan	2.5×10^{-5}	0.7
RockSample $(11, 11)$	2.5×10^{-4}	0.7
RockSample $(15, 15)$	2.5×10^{-5}	0.5

(b) Best hyperparameters found for the recurrent PPO baseline.

	Step size	λ_1
Battleship	2.5×10^{-5}	0.1
PocMan	2.5×10^{-4}	0.7
RockSample $(11, 11)$	2.5×10^{-4}	0.1
RockSample $(15, 15)$	2.5×10^{-5}	0.1

(c) Best hyperparameters found for the memoryless PPO baseline.

Figure 11: Best hyperparameters for each environment and each algorithm. Hyperparameters were found using 5 seeds, and taking the maximum AUC.

Our network architectures are standard multi-layer perceptions (MLPs). Both actor and critic networks are two-layer MLPs with ReLU [Nair and Hinton, 2010] activations between layers. The actor network applies a softmax function to its output logits. Our recurrent neural network is a dense layer, followed by ReLU activation, then the GRU cell, then another dense layer. For Battleship, to better condition on the hit-or-miss observation, we add an additional dense layer after the first dense

layer of the RNN (which has $2\times$ latent size for this environment), that takes as input the outputs of the previous layer concatenated with the hit-or-miss bit (a residual connection). For our memoryless baseline, we replace the recurrent neural network with a 3-layer MLP with ReLU activations between each layer. Put together, our actor-critic network takes in an input observation (and optionally, the previous action) and passes it, as well as the previous latent state, through the GRU for a new latent state. For the memoryless baseline, the 3-layer MLP simply encodes the observation (and potentially action) into the latent state, with no recurrence. It uses this new latent state as inputs to both the actor and critic. All hidden layer latent sizes are the same sizes as the latent sizes in Figure 12.

We now detail environment-specific hyperparameters. We do so in Figure 12.

	Latent size	c_{Ent}
Battleship	512	0.05
PocMan	512	0.05
RockSample $(11, 11)$	128	0.35
RockSample $(15, 15)$	256	0.35

Figure 12: Environment-specific hyperparameters, set across all algorithms. We set the entropy coefficient to a higher value in RockSample because the environment requires more exploration.

The full implementation of algorithms, experiments, and environments are available at https://github.com/brownirl/lambda_discrepancy.

G.5 P.O. PacMan Pellet Probe Visualization Details

We train two RNN hidden state probes in order to generate the memory visualizations in Figure 2. Probes were trained on the hidden state outputs of the RNN + PPO and RNN + PPO + LD agents. Training was done over 2M time steps, where 1M steps were collected from each of these agents. After collection, all trajectories were run through both agents to collect 2M time steps to collect both RNN + PPO hidden states and the RNN + PPO + LD hidden states. With this dataset, the each probe was trained with the corresponding RNN hidden states as inputs, with the pellet occupancy of all potential pellet positions as the target for the prediction. Both probes are 3-layer neural networks with ReLU activations [Nair and Hinton, 2010] between each layer, and a sigmoid function over the final logits to map outputs to 0 and 1. We use a binary cross entropy loss between these predictions and the pellet occupancy targets. The hidden size of the network was 1024, with a step size of 0.0001. At every step of training, a batch of 32 time steps are uniformly randomly sampled from the dataset, and the binary cross entropy loss was minimized. The agent performs 10M steps of training to reach the performance visualized.

G.6 Small-Scale POMDP Experiments

We also run both recurrent PPO and our λ-discrepancy-augmented PPO algorithm on the small-scale POMDPs evaluated in the analytical experiments in Section 4. For these experiments, all agents had a latent state size of 32, with $c_{\text{Ent}} = 0.05$, and the same hyperparameters swept as in Appendix G.4. We show results in Figure 13.

These results imply that the baseline algorithm is already sufficient for solving these tasks, and any additional auxiliary losses cannot help with performance, since performance is already near optimal. We would like to note that, as with the results in Figure 5, adding our λ-discrepancy auxiliary loss never *harms* the performance of PPO.

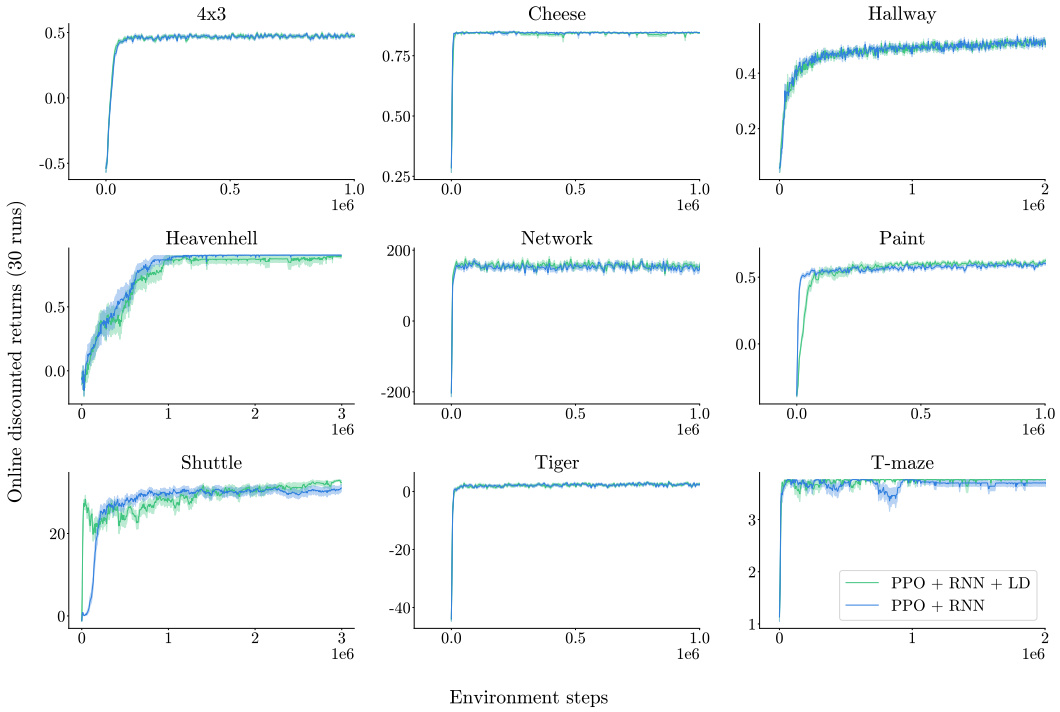

Figure 13: Learning curves for recurrent PPO and λ-discrepancy-augmented PPO on small-scale POMDPs.