MISMATCHED NO MORE: JOINT MODEL-POLICY OPTIMIZATION FOR MODEL-BASED RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Many model-based reinforcement learning (RL) methods follow a similar template: fit a model to previously observed data, and then use data from that model for RL or planning. However, models that achieve better training performance (e.g., lower MSE) are not necessarily better for control: an RL agent may seek out the small fraction of states where an accurate model makes mistakes, or it might act in ways that do not expose the errors of an inaccurate model. As noted in prior work, there is an objective mismatch: models are useful if they yield good policies, but they are trained to maximize their accuracy, rather than the performance of the policies that result from them. In this work we propose a model-learning objective that directly optimizes a model to be useful for model-based RL. This objective, which depends on samples from the learned model, is a (global) lower bound on the expected return in the real environment. We jointly optimize the policy and model using this one objective, thus mending the objective mismatch in prior work. The resulting algorithm (MnM) is conceptually similar to a GAN: a classifier distinguishes between real and fake transitions, the model is updated to produce transitions that look realistic, and the policy is updated to avoid states where the model predictions are unrealistic. Our theory justifies the intuition that the best dynamics for learning a good policy are not necessarily the correct dynamics.

1 INTRODUCTION

Much of the appeal of model-based RL is that model learning is a simple and scalable supervised learning problem. Unfortunately, the accuracy of the learned model does not directly correlate with whether the model-based RL algorithm will receive high reward [15, 23]. For example, a model might make small mistakes in critical states that cause a policy to take suboptimal actions. Alternatively, a model with large errors may yield a policy that attains high return if the model errors occur in states that the policy never visits.

The underlying problem is that dynamics models are trained differently from how they are used. Typical model-based methods train a model using data sampled from the *real* dynamics (e.g., using maximum likelihood), but apply these models by using data sampled from the *learned* dynamics [9, 10, 19, 20, 40]. Prior work has identified this *objective mismatch* issue [15, 23, 24, 36, 44]: the model is trained using one objective, but the policy is trained using a different objective. Designing an effective objective for model training that is guaranteed to improve the expected reward remains an open problem. This paper aims to answer the following question: *How should we train a dynamics model so that it produces high-return policies when used for model-based RL?*

In this paper, we propose a model-based RL algorithm where the model and policy are *jointly optimized* with respect to the same objective. Importantly, this objective is a lower bound on the expected return under the true environment dynamics. This bound becomes tight under some assumptions, thereby guaranteeing that policies learned by interacting with the learned model will not necessarily be worse than policies learned by interacting with the real environment. Structurally, our algorithm resembles a generative adversarial network (a GAN), where the model is trained using a discriminator that distinguishes between real and fake transitions. Crucially, this same discriminator is also included in the objective for the policy, and both the model and policy are jointly trained to maximize reward and minimize discriminator accuracy. Thus, the model and policy cooperate to produce realistic and high-reward trajectories against an adversary that attempts to classify whether

the trajectory is realistic. Our method stands in contrast to standard model-based RL methods, where it is more common to pit the model "against" the policy [5, 26, 30]. By cooperating with the policy, the model initially learns "optimistic" dynamics that facilitate exploration. Although this design choice may appear counterintuitive, we show that it can be reduced to a lower bound on the expected reward. We illustrate the method via a diagram in Figure 1.

The main contribution of this work is an algorithm, Mismatched no More (MnM) for model-based RL that provably maximizes a lower bound on expected reward. Importantly, this bound is non-vacuous, becoming tight at optimality under some assumptions. This algorithm has the unique property of jointly optimizing the policy and model using the same objective. Across a range of tasks, we demonstrate that our method is competitive with prior state-of-the-art methods on benchmark tasks; on certain hard exploration tasks, our method outperforms prior methods based on maximum likelihood.

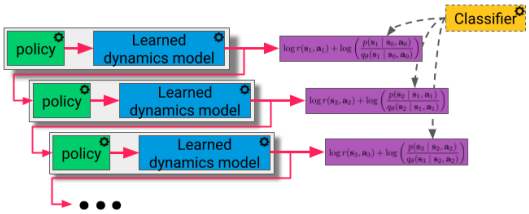

Figure 1: **Mismatched No More:** Our method jointly optimizes the policy and dynamics model to visit states with high reward and take transitions that look realistic. A separate classifier estimates whether transitions are realistic or not.

2 Related Work

Model-based RL methods typically fit a dynamics model to observed transitions, and then apply an RL method to that learned model. Most of these methods use maximum likelihood to fit the dynamics model, and then use RL to maximize the expected return under samples from that model [9, 10, 19, 20, 40]. The observation that this maximum likelihood objective is not aligned with the RL objective has been noted in prior work [15, 23, 24, 36, 44]. One strategy for mitigating this problem is to modify the model training to improve model accuracy under multi-step rollouts [3, 4, 15, 21, 36, 39]. A second, orthogonal strategy is to make the policy act conservatively with respect to the learned model, regularizing the policy to avoid transitions where the model is likely to be inaccurate [22, 33, 41, 43]. Like Yu et al. [41], our method modifies the reward function for RL to include an additional term that is estimated by a discriminator. A third strategy is to directly optimize the model such that it produces good policies [2, 27, 34]. While our aim, learning models that produce good policies, is the same as these prior methods, our approach will not require differentiating through unrolled model updates or optimization procedures.

Prior methods have analyzed theoretical properties of model-based RL methods. Some prior works cast MBRL as a two-player game between a model-player and policy player [5, 26, 29, 31]. However, whereas the formulation in these prior works have the model and policy compete against one another, our formulation will result in a cooperative game wherein the model and policy players cooperate in optimizing the *same* objective (a lower bound on the expected return). Asadi et al. [4] study smoothness of learned dynamics models, arguing that smoothness mitigates compounding errors. Our method will combat compounding errors by learning an explicit classifier to detect these errors. Similar to our work, Luo et al. [24] construct a lower bound on the expected reward objective using a term that involves the discrepancy between the learned model and the true environment dynamics. Algorithmically, an important difference with our method is that we learn a classifier to identify these discrepancies instead of relying on mean-squared error. Theoretically, our lower bound holds globally, not just for local regions of the state space [24]. Our method builds on prior work in domain adaptation for RL [13, 38] that penalizes a policy for visiting states where a (fixed) dynamics model is inaccurate. Unlike these methods, our approach will also optimize the dynamics model to become more accurate. Whereas our method updates the model and policy to be self-consistent, some exploration methods do exactly the opposite, rewarding the policy for visiting states where the model has high error [7, 28, 35]. Importantly, our method does not forgo exploration, but rather learns a dynamics model that itself facilitates effective exploration.

The most similar prior work is VMBPO [8]. The most important difference between our method and VMBPO is that our method maximizes a lower bound on expected return, whereas VMBPO maximizes a different, risk-seeking objective, which is an *upper* bound on expected return. This different objective can be expressed as the expected return plus the variance of the return, so VMBPO

has the undesirable property of preferring policies that receive slightly lower return if the variance of the return is much larger (see Appendix B.1).

3 A UNIFIED OBJECTIVE FOR MODEL-BASED RL

Notation. We focus on the Markov decision process with states s_t, actions a_t, initial state distribution $p_0(s_0)$, positive reward function $r(s_t, a_t) > 0$, and dynamics $p(s_{t+1} \mid s_t, a_t)$. Our aim is to learn a control policy $\pi_\theta(a_t \mid s_t)$ with parameters θ that maximizes the expected discounted return:

$$\max_\theta \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

During training, the agent receives transitions $(s_t, a_t, r_t = r(s_t, a_t), s_{t+1} \sim p(s_{t+1} \mid s_t, a_t))$ collected from the environment. These samples can either come from an offline dataset, or be collected interactively from the policy being learned. We will use these transitions to train a dynamics function $q_\theta(s_{t+1} \mid s_t, a_t)$, and use transitions sampled from this learned model to train the policy. To simplify notation, we will define a trajectory τ to be the sequence of states and actions visited in an episode: $\tau \triangleq (s_0, a_0, s_1, a_1, \cdots)$. We then define $R(\tau) \triangleq \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ as the discounted return of a trajectory. Finally, we define two distributions over trajectories. First, $p^\pi(\tau)$ is the distribution over trajectories when policy π_θ interacts with dynamics $p(s' \mid s, a)$; $q^\pi(\tau)$ is the distribution over trajectories when policy π_θ interacts with the learned dynamics $q_\theta(s_{t+1} \mid s_t, a_t)$:

$$p^\pi(\tau) = p_0(s_0) \prod_{t=0}^{\infty} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t), \quad q^\pi(\tau) = p_0(s_0) \prod_{t=0}^{\infty} q_\theta(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t).$$

Desiderata. Our aim is to design an objective $\mathcal{L}(\theta)$ that can be used to jointly optimize both the policy $\pi_\theta(a_t \mid s_t)$ and the dynamics model, $q(s_{t+1} \mid s_t, a_t)$. Since both the model and policy will be optimized using the same objective, our method will avoid the objective mismatch problem from prior work. This objective should be a lower bound on the expected return which becomes tight at optimality, such that improving the model is guaranteed to improve the policy

An objective for model-based RL. We now introduce an objective that achieves these aims. Our objective will be the policy's reward when interacting with the learned model, but using a different reward function. The new reward function, $\tilde{r}$, augments the task reward with an additional term that measures the difference between the learned model and the real environment. We define our objective

$$\mathcal{L}(\theta) \triangleq \mathbb{E}_{q^{\pi_\theta}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t \tilde{r}(s_t, a_t, s_{t+1}) \right], \tag{2}$$

where the modified reward function is defined as

$$\tilde{r}(s_t, a_t, s_{t+1}) \triangleq (1-\gamma) \log r(s_t, a_t) + \log \left(\frac{p(s_{t+1} \mid s_t, a_t)}{q(s_{t+1} \mid s_t, a_t)} \right) - (1-\gamma) \log(1-\gamma). \tag{3}$$

This objective should be understood as maximizing an *augmented* reward under the *learned* dynamics. The augmented reward function $\tilde{r}$ penalizes the policy for taking transitions that are unlikely under the true dynamics model, similar to prior work [13, 41]. Later, we will show that we can estimate this augmented reward *without knowing the true environment dynamics* by using a GAN-like classifier. We will optimize this lower bound with respect to both the policy $\pi_\theta(a_t \mid s_t)$ and the dynamics model $q_\theta(s_{t+1} \mid s_t, a_t)$. For the policy, this optimization entails performing RL to maximize the modified reward using samples from the learned model; the only difference from prior work is the modification to the reward function. Training the dynamics model using this objective is very different from standard maximum likelihood training, and instead resembles a GAN. The model is optimized to sample trajectories that both have high reward (i.e., $\log r$ is large) and are similar to real dynamics (i.e., $\log \frac{p}{q}$ is large). This objective differs from VMBPO [8] by taking the log of the original reward functions; our experiments demonstrate that excluding this component invalidates our lower bound and results in learning suboptimal policies. Our objective satisfies a crucial property: it is a lower bound on the (log) expected return in the real environment:

Theorem 3.1. *The following bound holds for **any** dynamics $q(s_{t+1} \mid s_t, a_t)$ and policy $\pi(a_t \mid s_t)$:*

$$\log \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \geq \mathcal{L}(\theta).$$

The proof is presented in Appendix B.3. Of course, maximizing the $\log(\cdot)$ of the expected return is equivalent to maximizing the expected return. Note that, unlike prior work [24], this bound holds for all dynamics functions and policies, not just those in a small region around previous dynamics functions and policies. To the best of our knowledge, this is the first such *global* lower bound for model-based RL. The lower bound raises a natural question: how tight is this lower bound? Showing that a bound is tight is crucial for making the claim that better dynamics models will produce better policies. In our experiments (e.g., Fig. 4), we show that this bound is quite tight in practice, and optimizing this lower bound yields policies that achieve high reward across a wide range of tasks.

Tightening the lower bound. We now introduce a modification to our lower bound that does make the bound tight. This new lower bound will be more complex than the one introduced above and we have not yet successfully designed a non-trivial algorithm for maximizing it. Nonetheless, we believe that presenting the bound may prove useful for the design of future model-based RL algorithms.

We will use $\mathcal{L}_\gamma(\theta)$ to denote this new lower bound. In addition to the policy and dynamics, this bound will also depend on a time-varying discount, $\gamma_\theta(t)$, in place of the typical γ^t term. Similar learned discount factors have been studied in previous work on model-free RL [32]. We define this objective as follows:

$$\mathcal{L}_\gamma(\theta) \triangleq \mathbb{E}_{q^{\pi_\theta}(\tau)} \left[\sum_{t=0}^{\infty} \gamma_\theta(t) \tilde{r}_\gamma(s_t, a_t, s_{t+1}) \right], \tag{4}$$

where the augmented reward is now defined as

$$\tilde{r}_\gamma(s_t, a_t, s_{t+1}) \triangleq \log r(s_t, a_t) + \frac{\Gamma_\theta(t)}{\gamma_\theta(t)} \log \left(\frac{p(s_{t+1} \mid s_t, a_t)}{q_\theta(s_{t+1} \mid s_t, a_t)} \right) + \log \left(\frac{\gamma^t}{\gamma_\theta(t)} \right)$$

and $\Gamma_\theta(t) = \sum_{t'=0}^{t} \gamma_\theta(t')$ is the CDF of the learned discount function. This new lower bound, which differs from our main lower bound by the learnable discount factor, does provide a tight bound on the expected return objective:

Lemma 3.2. *Let an arbitrary policy $\pi(a_t \mid s_t)$ be given. The objective $\mathcal{L}_\gamma(\theta)$ is also a lower bound on the expected return objective, $\log \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \geq \mathcal{L}_\gamma(\theta)$, and this bound becomes tight at optimality:*

$$\log \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] = \max_{q_\theta(\tau), q_\theta(t)} \mathbb{E}_{q^{\pi_\theta}(\tau)} \left[\sum_{t=0}^{\infty} \gamma_\theta(t) \tilde{r}_\gamma(s_t, a_t, s_{t+1}) \right].$$

The proof is presented in Appendix 3.2. One important limitation of this result is that the learned dynamics that maximize this lower bound to make the bound tight may be non-Markovian. While our lower bound remains valid for any dynamics model, including Markovian models, restricting a model-based RL algorithm to Markovian dynamics models precludes the algorithm from making the bound tight. Intriguingly, this analysis suggests that using non-Markovian models such as RNNs and transformers may accelerate learning on Markovian tasks. We leave an empirical investigation of the more complex lower bound $\mathcal{L}_\gamma$ and non-Markovian models to future work.

The optimal dynamics are optimistic. In stochastic environments, the dynamics model that optimizes the lower bound $\mathcal{L}_\gamma(\theta)$ is not (in general) equal to the true environment dynamics. Rather, it is biased towards sampling trajectories with high return. Ignoring parametrization constraints, the dynamics model that optimizes our lower bound is $q^*(\tau) = \frac{p(\tau)R(\tau)}{\int p(\tau')R(\tau')d\tau'}$ (proof in Appendix B.4.). We hypothesize that the optimism in the dynamics model will accelerate policy optimization, an hypothesis test in Sec. 5.1. Importantly, the augmented reward compensates the the optimism in the dynamics model, such that our objective is a valid lower bound.

4 MISMATCHED NO MORE

The previous section presented a single (global) lower bound ($\mathcal{L}$ from Eq. 2) for jointly optimizing the policy and the dynamics model. In this section, we develop a practical algorithm for optimizing this lower bound. We call our method MISMATCHED NO MORE (MnM) because the policy and model optimize the same objective, thereby resolving the objective mismatch problem noted in prior work. The main challenge in optimizing this bound is that the augmented reward function depends on the transition probabilities of the real environment, $p(s_{t+1} \mid s_t, a_t)$, which are unknown. We first estimate this function by learning a classifier (Sec. 4.1), and then describe the precise update rules for the policy, dynamics model, and classifier (Sec. 4.2).

4.1 ESTIMATING THE AUGMENTED REWARD FUNCTION

We now describe how we estimate the augmented reward function, which depends on the transition probabilities of the real environment. Instead of estimating the transition probabilities of the real environment directly, we learn a classifier that distinguishes real transitions from fake transitions. This approach is similar to GANs [17] and similar to prior work in RL [13, 41]. We use $C_\phi(s_t, a_t, s_{t+1}) \in [0, 1]$ to denote the classifier. We can use the classifier's predictions to estimate the augmented reward function:

$$\tilde{r}(s_t, a_t, s_{t+1}) = \log r(s_t, a_t) + \underbrace{\log \frac{C_\phi(s_t, a_t, s_{t+1})}{1 - C_\phi(s_t, a_t, s_{t+1})}}_{\approx \log p(s_{t+1}|s_t,a_t) - \log q_\theta(s_{t+1}|s_t,a_t)} . \tag{5}$$

We now present our complete method, which trains three components: a classifier, a policy, and a dynamics model. Our method alternates between *(1)* updating the policy (by performing RL using model experience with augmented rewards) and *(2)* updating the dynamics model and classifier (using a GAN-like objective). In describing the loss functions below, we use the superscripts $(\cdot)^{\text{real}}$ and $(\cdot)^{\text{model}}$ to denote transitions that have been sampled from the true environment dynamics or the learned dynamics function. To reduce clutter, we omit the superscripts when unambiguous.

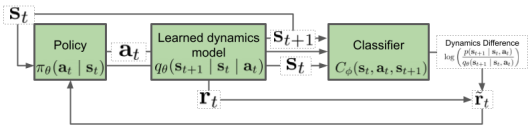

Figure 2: MnM learns a policy, dynamics model, and classifier. The classifier distinguishes real transitions from model transitions, and is used both to train the dynamics model and to augment the reward function. The policy maximizes the augmented reward using transitions sampled from the model.

4.2 UPDATING THE MODEL, POLICY, AND CLASSIFIER

Updating the policy. To optimize the policy, we will maximize the augmented reward on transitions sampled from the learned dynamics model. This optimization can be done using any RL algorithm, including on-policy and off-policy methods. In this section, we describe the off-policy, actor-critic method we used in our experiments.

We define the Q function as sum of *augmented* rewards under the learned dynamics model:

$$Q(s_t, a_t) \triangleq \mathbb{E}_{\substack{\pi(a_t|s_t), \\ q_\theta(s_{t+1}|s_t,a_t)}} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} \tilde{r}(s_{t'}, a_{t'}) \mid s_t = s_t, a_t = a_t \right] . \tag{6}$$

We approximate the Q function using a neural network $Q_\psi(s_t, a_t)$ with parameters ϕ. To estimate the corresponding value function, we use a 1-sample approximation: $V_\psi(s_t) = Q_\psi(s_t, a_t \sim \pi_\theta(a_t \mid s_t))$. We train the Q function using the TD loss on transitions sampled from the *learned* dynamics model:

$$\mathcal{L}_Q(s_t, a_t, r_t, s_{t+1}^{\text{model}}; \psi) = \left(Q_\psi(s_t, a_t) - \lfloor y_t \rfloor_{\text{sg}}\right)^2 , \tag{7}$$

where $\lfloor \cdot \rfloor_{\text{sg}}$ is the stop-gradient operator and $y_t = \tilde{r}(s_t, a_t, s_{t+1}^{\text{model}}) + \gamma \mathbb{E}_{\pi(a_{t+1}|s_{t+1}^{\text{model}})} \left[Q_\psi(s_{t+1}^{\text{model}}, a_{t+1}) \right]$. The augmented reward, $\tilde{r}$, is estimated using the learned classifier. The policy is trained to maximize the expected future (augmented) return, as estimated by the Q function:

$$\max_\theta \mathcal{L}_\pi(s_t; \theta) \triangleq \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[\tilde{Q}_\phi(s_t, a_t) \right] . \tag{8}$$

Algorithm 1 **Mismatched no More (MnM)** is an algorithm for model-based RL. The method alternates between training the policy on experience from the learned dynamics model with augmented rewards and updating the model+classifier using a GAN-like loss. Updates are gradient steps with the Adam optimizer.

1: **while** not converged **do**
2: Sample experience from learned model, and modify rewards using classifier (Eq. 5).
3: Update policy and Q function using model experience and augmented rewards (Eq.s 8 and 7).
4: Update model and classifier using GAN-like loss (Eq.s 9 and 10).
5: (Infrequently) Sample experience from real model.
6: **return** policy $\pi_\theta(a_t \mid s_t)$.

In our implementation, we regularize the policy by adding an additional entropy regularizer. Following prior work [16], we maintain two Q functions and two target Q functions, use the minimum of the two target Q functions to compute the TD target, and do not backpropagate gradients through the TD targets y_t. For both loss functions, we sample the transitions uniformly from a replay buffer of experience collected from the learned dynamics model. See Appendix C for details.

Updating the classifier. We train the classifier to distinguish real versus model transitions using the standard cross entropy loss:

$$\max_\phi \mathcal{L}_C(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{real}}, s_{t+1}^{\text{model}}; \phi) \triangleq \log C_\phi(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{real}}) + \log\left(1 - C_\phi(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{model}})\right). \quad (9)$$

Note that the real transition $(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{real}})$ and model transition $(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{model}})$ have the same initial state and initial action.

Updating the dynamics model. To optimize the dynamics model, we rewrite the lower bound in terms of a single transition:

$$\mathcal{L}_q(s_t^{\text{real}}, a_t^{\text{real}}; \theta) = \mathbb{E}_{s_t^{\text{model}} \sim q(s_{t+1} \mid s_t^{\text{real}}, a_t^{\text{real}})}\Bigg[V_\psi(s_{t+1}^{\text{model}}) + \underbrace{\log \frac{C_\phi(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{model}})}{1 - C_\phi(s_t^{\text{real}}, a_t^{\text{real}}, s_{t+1}^{\text{model}})}}_{\approx \log p(s_{t+1}^{\text{model}} \mid s_t^{\text{real}}, a_t^{\text{real}}) - \log q_\theta(s_{t+1}^{\text{model}} \mid s_t^{\text{real}}, a_t^{\text{real}})}\Bigg]. \quad (10)$$

The approximation above reflects approximation error in learning the optimal classifier. This approximation is standard in prior work on GANs [17] and adversarial inference [11, 12]. The procedure for optimizing the dynamics model and the classifier resembles a GAN [17]: the classifier is optimized to distinguish real transitions from model transitions, and the model is updated to fool the classifier. However, *our method is not equivalent to simply replacing a maximum likelihood model with a GAN model*. Indeed, such an approach would not optimize a lower bound on expected return. Rather, our model objective adds an additional value term and our policy objective adds an additional classifier term. These changes enable the model and policy to optimize the same objective, which is a lower bound on expected return.

Algorithm summary. We summarize the method in Alg. 1. Implementing MnM on top of a standard model-based RL algorithm is straightforward. First, create an additional classifier network. Second, instead of using the maximum likelihood objective to train the model, use the GAN-like objective in Eq. 10 to update both the model and the classifier. Third, add the classifier's logits to the predicted reward function (Eq. 5) before applying the standard RL updates. Following prior work [20], we learn an additional model to predict the true environment reward.

5 Experiments

We present two sets of experiments. Our first set of experiments studies the importance of different components of MnM and builds intuition for what the dynamics model learns. Second, we study how MnM compares with prior model-based RL algorithms on challenging, robotic control tasks.

5.1 Understanding the Lower Bound and the Learned Dynamics

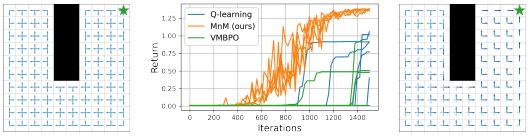

Figure 3: **MnM learns dynamics that accelerate RL.** *(Left)* We apply MnM to a navigation task with stochastic dynamics (blue arrows); the agent receives a sparse reward for reaching the goal (⋆). *(Center)* MnM solves this task more quickly than Q-learning and VMBPO. *(Right)* The dynamics learned by MnM are different from the real dynamics, biasing the noise to point towards the goal.

In this section, we study the seemingly contradictory attributes of our method: optimistic dynamics and a pessimistic reward function. While MnM learns a dynamics model is that optimistic (see Sec. 3), our method actually optimizes a *lower bound* because the augmented reward function counterbalances the effect of this optimism. The augmented reward function contains two crucial components, *(1)* the logarithmic transformation of the reward function and *(2)* the classifier term. Although most model-based RL methods omit both components, the benefits of the classifier term are rather intuitive: this term rewards the agent for visiting states where the model is accurate. However, the benefits of the logarithmic transformation are harder to recognize. For analysis, we will compare MnM to VMBPO [8], which also learns optimistic dynamics and also includes the classifier term but omits the logarithmic transformation.

Our theory suggests that MnM should work best in settings with stochastic dynamics, as the learned dynamics model should tilt the true dynamics to make the stochasticity more favorable for solving the task. Our first experiment studies this capability in an idealized setting. We use a 10x10 gridworld with highly stochastic dynamics, and the agent receives only a sparse reward upon reaching a goal state (Fig. 3 *(Right)*). The stochasticity and sparse rewards make this task challenging, while the tabular setting allows us to study the performance of MnM without introducing function approximation error. Note that the learned value function depends on samples from the MnM dynamics model, and the MnM dynamics model depends on the learned value function. Thus, it is not immediately clear this cyclic dependency would accelerate learning or create unstable learning dynamics. The results, shown in Fig. 3 *(Center)*, show that MnM learns this task much faster than both Q-learning and VMBPO. For fair comparison, both methods receive the same amount of data, perform the same number of updates, and are evaluated using the real environment dynamics. In line with our theory, the dynamics learned by MnM (Fig. 3 *(Right)*) not equal to the true environment dynamics, but rather alter the environment stochasticity to tend to lead the agent towards the goal. This visualization gives us a clue for why MnM learns faster than Q-learning: the learned dynamics make the exploration problem easier, increasing the probability of collecting high-reward experience. While VMBPO also learns optimistic dynamics for this task, it is different from MnM because it does not perform a log-transformation of the reward function, a difference that seems to have a large effect on this task.

We now study the lower bound optimized by MnM. For our objective to be useful, it is import not only that the model and policy optimize the same objective, but also that this objective be a lower bound and that this lower bound be tight. To confirm that our lower bound is actually a lower bound, and to measure how tight this bound is, we implement a version of MnM using value iteration. This implementation allows us to compute the bound exactly without incurring function approximation error and sampling error. The results shown in Fig. 4 *(Left)* show that the MnM lower bound is a valid lower bound. Moreoever, the gap between the lower bound and the (log) expected return decreases from about 1.0 to 0.001. Because the dynamics model learned by MnM is optimistic, it is not surprising that the (log) expected return under the learned model overestimates the (log) expected

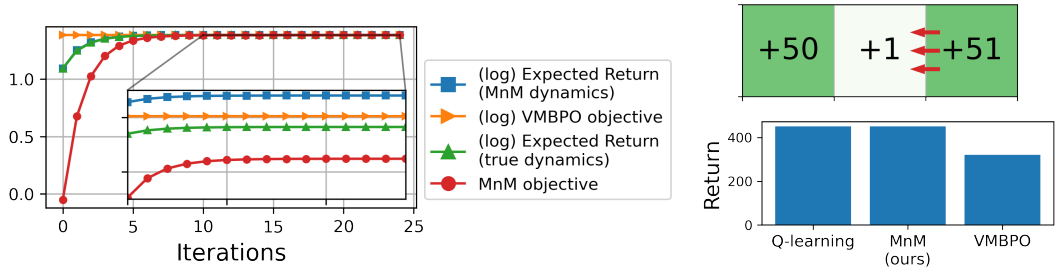

Figure 4: **Lower Bounds and Stochastic Dynamics.** *(Left)* As predicted by our theory, the MnM objective (red) is a lower bound on the log of the expected return objective (blue); the final gap is 0.001. Both VMBPO and an ablation of MnM that does not augment the reward function overestimate the reward. *(Right)* On a simple gridworld with a stochastic transition in one state (red arrows), MnM converges to the reward-maximizing policy while VMBPO learns a strategy with lower rewards and higher variance (as predicted by theory).

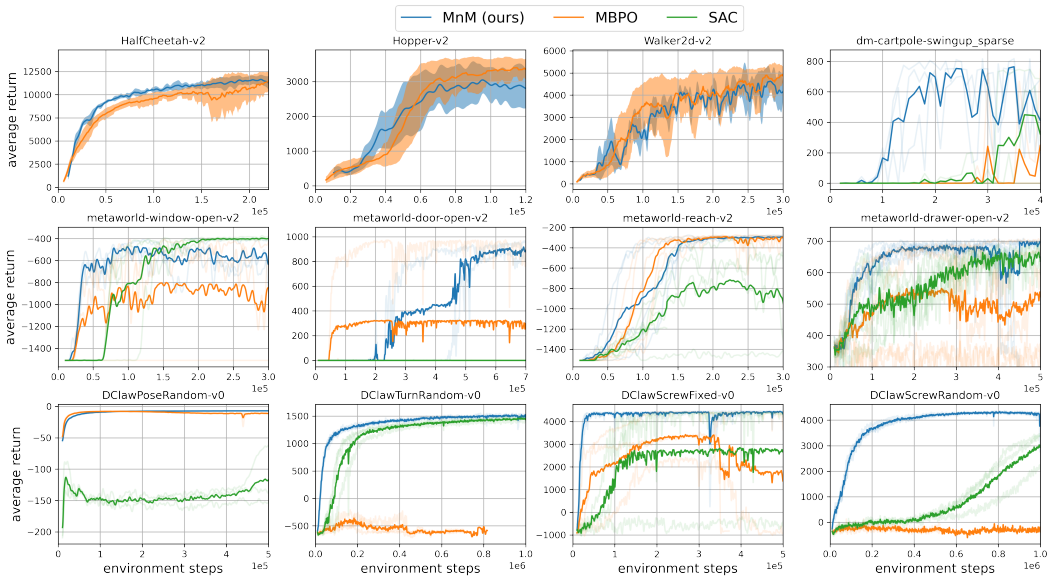

Figure 5: **Comparison on Robotics Tasks**: We compare MnM to MBBO and model-free RL ("RL on real") on simulated control tasks. On the benchmark locomotion tasks (top left), MnM performs comparably with MBPO. On many of the other tasks with sparse rewards that pose an exploration challenge, MnM significantly outperforms both MBPO and the model-free baseline. These experiments suggest that maximizing a well defined bound on expected return, as done by our method, can lead to improved performance on difficult tasks.

return under the true dynamics; our lower avoids this overestimation by using the augmented reward function. Finally, we observe that the (log) VMBPO objective, which omits the log transformation, also overestimates the expected return.

While MnM maximizes a lower bound on the expected return, VMBPO maximizes a risk-seeking objective. We thus expect that VMBPO will prefer policies with slightly lower returns if those policies have much higher variance. This risk-seeking behavior is not evident in benchmark tasks because these tasks have nearly deterministic dynamics and reward functions. We therefore craft a simple environment to study risk seeking behavior. In this task, shown in Fig. 4 *(Right)*, the agent can move left and receive a reward of +50 or move right and receive a slightly higher reward (+51). However, "wind" in the right state knocks the agent out of this state with probability 50%. Thus, the optimal policy chooses to go left. MnM learns the correct policy for this task, indicating that the augmented reward function does successfully counter the optimism in the learned dynamics. In contrast, VMBPO learns a policy that incorrectly chooses to go right, and thus receives lower cumulative return.

5.2 Comparisons On Robotics Tasks

Our second set of experiments evaluate MnM on continuous-control robotic tasks of increasing difficulty. The MuJoCo locomotion tasks studied in prior work model-based RL [20] generally do not present a substantial exploration challenge, as they have dense rewards. Therefore, we do not expect MnM to outperform prior methods. However, we additionally include more difficult sparse reward benchmark domains from prior work on model-free RL [1, 37, 42], which present a more substantial exploration challenge, leading us to expect more noticeable performance gains. We illustrate a subset of the experimental environments in Appendix Fig. 9, and include full implementation details in Appendix C.2. One detail of note is that we omit the reward augmentation from the MnM for these experiments, finding that it hinders exploration and leads to lower returns. These experiments will compare against state-of-the-art RL algorithms such as MBPO [20]. MnM is structurally similar to MBPO, allowing for an apples-to-apples comparison. See Appendix A for more experiments.

Our first comparison looks at three locomotion tasks from OpenAI Gym [6]. Despite being relatively simple control problems with dense rewards, these tasks have become the standard benchmark for model-based RL algorithms. The results, shown in Fig. 5 (top left) show that MnM performs roughly on par with MBPO.

Tasks with sparse rewards, complex contact dynamics, and those requiring hard exploration often present a challenge for model-based RL algorithms, which are liable to exploit inaccuracies in the learned dynamics model. Our next experiment evaluates MnM on control tasks used in prior work that demonstrate these properties. These tasks include a sparse version of the cartpole task from the DM Control benchmark [37], four manipulation tasks from the Metaworld benchmark [42], and four dexterous manipulation tasks from the ROBEL benchmark [1]. See Appendix‘C for experiment details. The results shown in Fig. 5 indicate that the complex environment dynamics of these tasks can cause prior model-based algorithms (MBPO) to perform worse than model-free algorithms (SAC), both in terms of asymptotic performance and sample complexity. Nonetheless, we observe that MnM frequently outperforms all prior methods and, more importantly, that it *consistently* does well across all tasks.

Finally, we visualize the dynamics learned by MnM on two of these robotic control tasks in Fig. 6. On the `Pusher-v2` task, the MnM dynamics cause the puck to move towards the robot arm even before the arm has come in contact with the puck. While this movement is not physically realistic, it does make the exploration problem easier; in line with our theory, the learned policy does perform well on the original dynamics. On the `HalfChetah-v2` task, the MnM dynamics increase the probability that the agent remain upright after tripping. This change to the dynamics may make it easier for the agent to learn how to run. Since these tasks have nearly-deterministic dynamics, why should MnM learn optimistic dynamics? We attribute the small (but noticeable) optimism in the MnM dynamics to the fact that our use of function approximators makes otherwise-deterministic dynamics look stochastic. For example, if the function approximator learns features that ignore some joint angle, then the dynamics of that joint angle will appear to be stochastic. Indeed, we expect that this implicit stochasticity caused by function approximation to be increasingly important for real-world tasks, where the complexity of the real dynamics often dwarfs the capacity of the learned dynamics model.

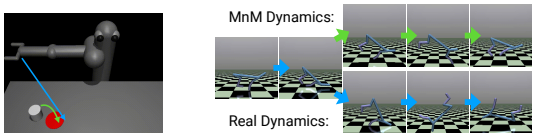

Figure 6: **Optimistic Dynamics**: *(Left)* On the `Pusher-v2` task, the MnM dynamics model makes the puck move towards the puck move towards the gripper before being grasped. *(Right)* On the `HalfCheetah-v2` task, the MnM dynamics model helps the agent stay upright after tripping.

6 Conclusion

The main contribution of this paper is an approach to model-based RL where the policy and dynamics model are jointly optimized using the same objective. Unlike prior work [8], our objective is a lower bound on the standard expected return objective. Our approach not only tells users how to train their dynamics model, but also guarantees to them that updating their model (using our objective) will result in a better policy. We believe that this *joint optimization* will ease and accelerate the design of future model-based RL algorithms.

We suspect that the greatest benefits will arise in domains with high-dimensional tasks, tasks with significant exploration challenges, and finally long-horizon tasks, all of which are particularly vulnerable to model exploitation. about applications of MnM to image-based settings, as a simple adjustment in our proof can be made to incorporate latent space models. We are especially optimistic about approaches which combine MnM with powerful but sample-complex methods. Because MnM is capable of producing realistic long-horizon model trajectories, requiring large quantities of on-policy experience is no longer detrimental, opening up a promising gateway for future work. A second direction for future work is to extend MnM to example-based control tasks, where the user specifies tasks using data (e.g., goal images) rather than reward functions. To encourage future work in this direction, we derive a similar lower bound for this setting in Appendix B.6.

Limitations. The main limitations of our method are complexity and stability. Compared with methods based on maximum likelihood, our method requires learning an additional classifier and balancing the capacity of that classifier against the capacity of the dynamics model. Despite the limitation, we believe that our lower bound may be a promising tool for designing new algorithms, as it allows researchers to directly optimize their dynamics models for producing good policies.

REFERENCES

[1] Ahn, M., Zhu, H., Hartikainen, K., Ponte, H., Gupta, A., Levine, S., and Kumar, V. (2020). Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on Robot Learning*, pages 1300–1313. PMLR.

[2] Amos, B., Rodriguez, I., Sacks, J., Boots, B., and Kolter, Z. (2018). Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*.

[3] Asadi, K., Misra, D., Kim, S., and Littman, M. L. (2019). Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*.

[4] Asadi, K., Misra, D., and Littman, M. (2018). Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 264–273. PMLR.

[5] Bagnell, J. A., Ng, A. Y., and Schneider, J. G. (2001). Solving uncertain markov decision processes.

[6] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[7] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. In *International Conference on Learning Representations*.

[8] Chow, Y., Cui, B., Ryu, M., and Ghavamzadeh, M. (2020). Variational model-based policy optimization. *arXiv preprint arXiv:2006.05443*.

[9] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

[10] Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472.

[11] Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

[12] Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

[13] Eysenbach, B., Chaudhari, S., Asawa, S., Levine, S., and Salakhutdinov, R. (2020a). Off-dynamics reinforcement learning: Training for transfer with domain classifiers. In *International Conference on Learning Representations*.

[14] Eysenbach, B., Salakhutdinov, R., and Levine, S. (2020b). C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*.

[15] Farahmand, A.-m., Barreto, A., and Nikovski, D. (2017). Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494.

[16] Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. *International Conference on Machine Learning (ICML)*.

[17] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NIPS)*.

[18] Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. (2018). TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`. [Online; accessed 25-June-2019].

[19] Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

[20] Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12519–12530.

[21] Joseph, J., Geramifard, A., Roberts, J. W., How, J. P., and Roy, N. (2013). Reinforcement learning with misspecified model classes. In *2013 IEEE International Conference on Robotics and Automation*, pages 939–946. IEEE.

[22] Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel : Model-based offline reinforcement learning. *ArXiv*, abs/2005.05951.

[23] Lambert, N., Amos, B., Yadan, O., and Calandra, R. (2020). Objective mismatch in model-based reinforcement learning. In *Learning for Dynamics and Control*, pages 761–770. PMLR.

[24] Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. (2019). Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *ICLR (Poster)*.

[25] Mihatsch, O. and Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine learning*, 49(2):267–290.

[26] Nilim, A. and El Ghaoui, L. (2003). Robustness in markov decision problems with uncertain transition matrices. In *NIPS*, pages 839–846. Citeseer.

[27] Okada, M., Rigazio, L., and Aoshima, T. (2017). Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*.

[28] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. In *International Conference on Machine Learning (ICML)*, pages 488–489. IEEE.

[29] Rajeswaran, A., Mordatch, I., and Kumar, V. (2020). A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, pages 7953–7963. PMLR.

[30] Ross, S. and Bagnell, D. (2012). Agnostic system identification for model-based reinforcement learning. In *ICML*.

[31] Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

[32] Rudner, T. G., Pong, V. H., McAllister, R., Gal, Y., and Levine, S. (2021). Outcome-driven reinforcement learning via variational inference. *arXiv preprint arXiv:2104.10190*.

[33] Sorg, J., Singh, S. P., and Lewis, R. L. (2010). Internal rewards mitigate agent boundedness. In *ICML*.

[34] Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR.

[35] Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

[36] Talvitie, E. (2014). Model regularization for stable sample rollouts. In *UAI*, pages 780–789.

[37] Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

[38] Vemula, A., Oza, Y., Bagnell, J. A., and Likhachev, M. (2020). Planning and execution using inaccurate models with provable guarantees. *arXiv preprint arXiv:2003.04394*.

[39] Venkatraman, A., Capobianco, R., Pinto, L., Hebert, M., Nardi, D., and Bagnell, J. A. (2016). Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics*, pages 703–713. Springer.

[40] Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J. M., Boots, B., and Theodorou, E. A. (2017). Information Theoretic MPC for Model-Based Reinforcement Learning. In *International Conference on Robotics and Automation (ICRA)*.

[41] Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. (2021). Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*.

[42] Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020a). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR.

[43] Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. (2020b). Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*.

[44] Ziebart, B. D. (2010). Modeling purposeful adaptive behavior with the principle of maximum causal entropy.

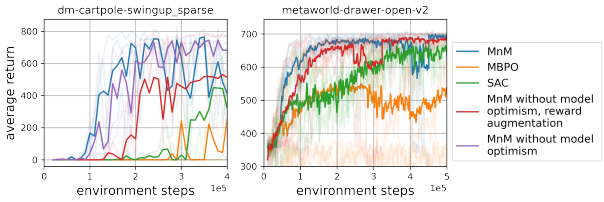

Figure 7: **Ablations Experiments**: Compared with MBPO (orange line), MnM uses a GAN-like model (red line) with a model optimism term and modifies the reward function.

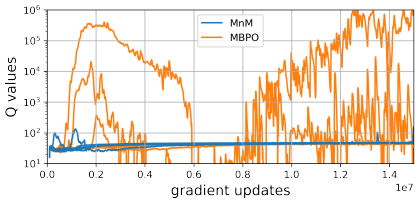

Figure 8: **Model exploitation**: On the `drawer-open` task where both MnM and MBPO learn policies that receive (undiscounted) returns of about 700, MBPO overestimates the returns by many of magnitude.

A Additional Experiments

This section describes additional experiments run on the robotic control tasks. We run an ablation experiment to study the importance of a few key design decisions. We compare MnM with ablations that omit the reward augmentation and the model optimism term and plot results Fig. 7. These results indicate that most of the benefit of MnM comes from using a GAN-like model. Because the dynamics of these tasks are nearly deterministic, it is not surprising that the optimistic dynamics and the reward augmentation have only a small effect.

To investigate the benefits of MnM over prior model-based methods, we logged the Q values throughout training and visualized them for the `metaworld-drawer-open-v2` task in Fig. 8. For fair comparison, we use Q values corresponding to just the task reward, omitting the augmented reward used by MnM. Fig. 8 shows that MnM yields Q values that are not only more accurate, but also more stable than MBPO. This figure suggests that algorithms such as MBPO exploit inaccuracies in the learned model.

B Proofs and Additional Analysis

B.1 VMBPO Maximizes an Upper Bound on Return

While MnM aims to maximize the (log) of the expected return, VMBPO aims to maximize the expected *exponentiated* return:

$$\text{MnM:} \quad \log \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \qquad \text{VMBPO:} \quad \log \mathbb{E}_\pi \left[e^{\eta \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)} \right],$$

where $\eta > 0$ is a temperature term used by VMBPO. Note that maximizing the log of the expected return, as done by MnM, is equivalent to maximizing the expected return, as the function $\log(\cdot)$ is monotone increasing. However, maximizing the log of the expected *exponentiated* return, as done by VMBPO, is not equivalent to maximizing the expected return. Rather, it corresponds to maximizing a sum of the expected return an the *variance* of the return [25, Page 272]:

$$\frac{1}{\eta} \log \mathbb{E}_\pi \left[e^{\eta \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)} \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \frac{\eta}{2} \mathrm{Var}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] + \mathcal{O}(\eta^2).$$

Thus, in environments with stochastic dynamics or rewards (e.g., the didactic example in Fig. **??**), VMBPO will prefer to receive lower returns if the variance of the returns is much higher. We note that the expected *exponentiated* return is an *upper* bound on the expected return:

$$\log \mathbb{E}_\pi \left[e^{\eta \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)} \right] \geq \eta \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right].$$

This statement is a direct application of Jensen's inequality. The bound holds with a strict inequality in almost all MDPs. The one exception is trivial MDPs where all trajectories have exactly the same return. Of course, even a random policy is optimal for these trivial MDPs.

B.2 HELPER LEMMAS

We start by introducing a simple identity that will help handle discount factors in our analysis.

Lemma B.1. *Define $p(H) = \text{GEOM}(1-\gamma)$ as the geometric distribution. Let discount factor $\gamma \in (0,1)$ and random variable x_t be given. Then the following identity holds:*

$$\mathbb{E}_{p(H)}\left[\sum_{t=0}^{H} x_t\right] = \sum_{t=0}^{\infty} \gamma^t x_t.$$

The proof involves substituting the definition of the Geometric distribution and then rearranging terms.

Proof.

$$\begin{aligned}
\mathbb{E}_{p(H)}\left[\sum_{t=0}^{H} x_t\right] &= (1-\gamma)\sum_{H=0}^{\infty}\gamma^H \sum_{t=0}^{H} x_t && (11)\\
&= (1-\gamma)\left(x_0 + \gamma(x_0+x_1) + \gamma^2(x_0+x_1+x_2) + \cdots\right) && (12)\\
&= (1-\gamma)\left(x_0(1+\gamma+\gamma^2+\cdots) + x_1(\gamma+\gamma^2+\cdots) + \cdots\right) && (13)\\
&= (1-\gamma)\left(x_0\frac{1}{1-\gamma} + x_1\frac{\gamma}{1-\gamma} + x_2\frac{\gamma^2}{1-\gamma} + \cdots\right) && (14)\\
&= \sum_{t=0}^{\infty}\gamma^t x_t. && (15)
\end{aligned}$$

□

The second helper lemma describes how the discounted expected return objective can be written as the expected *terminal* reward of a mixture of finite-length episodes.

Lemma B.2. *Define $p(H) = \text{GEOM}(1-\gamma)$ as the geometric distribution, and $p(\tau \mid H)$ as a distribution over length-H episodes. We can then write the expected discounted return objective as follows:*

$$\begin{aligned}
\mathbb{E}_{p(\tau|H=\infty)}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] &= \frac{1}{1-\gamma}\mathbb{E}_{p(H)}\left[\mathbb{E}_{p(\tau|H=H)}\left[r(s_H,a_H)\right]\right] && (16)\\
&= \frac{1}{1-\gamma}\iint p(H)p(\tau \mid H=H)r(s_H,a_H)d\tau dH. && (17)
\end{aligned}$$

Proof. The first identity follows from the definition of the geometric distribution. The second identity writes the expectations as integrals, which will make future analysis clearer. □

B.3 PROOF OF LEMMA 3.1

Proof.

$$\log \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] \overset{(a)}{=} \log \frac{1}{1-\gamma}\iint p(H)p(\tau \mid H=H)r(s_H,a_H)d\tau dH$$

$$= \log \iint p(H)\frac{p(\tau \mid H=H)}{q_\theta(\tau \mid H=H)}q_\theta(\tau \mid H=H)r(s_H,a_H)d\tau dH - \log(1-\gamma)$$

$$\overset{(b)}{\geq} \int p(H)\left(\log \int \frac{p(\tau \mid H=H)}{q_\theta(\tau \mid H=H)}q_\theta(\tau \mid H=H)r(s_H,a_H)d\tau\right)dH - \log(1-\gamma)$$

$$\overset{(c)}{\geq} \iint p(H)q_\theta(\tau \mid H=H)\left(\log p(\tau \mid H=H) - \log q_\theta(\tau \mid H=H) + \log r(s_H,a_H)\right)d\tau dH - \log(1-\gamma)$$

$$\overset{(d)}{=} \iint p(H)q_\theta(\tau \mid H=H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) + \cancel{\pi_\theta(a_t \mid s_t)} - \log q_\theta(s_{t+1} \mid s_t,a_t) - \cancel{\log \pi_\theta(a_t \mid s_t)}\right) + \log r(s_H,a_H)\right)d\tau dH - \log(1-\gamma)$$

$$\overset{(e)}{=} \iint p(H)q_\theta(\tau \mid H=\infty)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log r(s_H,a_H)\right)d\tau dH - \log(1-\gamma)$$

$$\overset{(f)}{=} \int q_\theta(\tau)\int p(H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log r(s_H,a_H)\right)dH d\tau - \log(1-\gamma)$$

$$\overset{(g)}{=} \int q_\theta(\tau)\mathbb{E}_{p(H)}\left[\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log r(s_H,a_H)\right]d\tau - \log(1-\gamma)$$

$$\overset{(h)}{=} \int q_\theta(\tau)\sum_{t=0}^{\infty}\gamma^t\left(\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t) + (1-\gamma)\log r(s_H,a_H)\right)d\tau - \log(1-\gamma)$$

$$\overset{(i)}{=} \mathbb{E}_{q_\theta(\tau)}\left[\sum_{t=0}^{\infty}\gamma^t\left(\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t) + (1-\gamma)\log r(s_H,a_H) - (1-\gamma)\log(1-\gamma)\right)\right].$$

For *(a)*, we applied Lemma B.2. For *(b)*, we applied Jensen's inequality. For *(c)*, we applied Jensen's inequality again. For *(d)*, we substituted the definitions of $p_\theta(\tau \mid H)$ and $q_\theta(\tau \mid H)$. For *(e)*, we noted that the term inside the summation only depends on the first H steps of the trajectory, so collecting longer trajectories will not change the result. This allows us to rewrite the integral as an expectation using a single infinite-length trajectory. For *(f)*, we recalled the definition $q_\theta(\tau) = q_\theta(\tau = H = \infty)$ and swap the order of integration. For *(g)*, we express the inner integral over $p(H)$ as an expectation. For *(h)*, we applied the identity from Lemma B.1. For *(i)*, we moved the constant $\log(1-\gamma)$ back inside the integral and rewrote the integral as an expectation. We have thus obtained the desired result. □

B.4 PROOF OF LEMMA 3.2

Before presenting the proof of Lemma 3.1 itself, we show how we derived the lower bound in this more general case. While this step is not required for the proof, we include it because it sheds light on how similar lower bounds might be derived for other problems. We define $q(H)$ to be a learned distribution over horizons H. We then proceed, following many of the same steps as for the proof of Lemma 3.1.

$$\log \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right] \overset{(a)}{=} \log \iint \frac{p(\tau,H)}{q_\theta(\tau,H)}q_\theta(\tau,H)r(s_H,a_H)d\tau dH - \log(1-\gamma)$$

$$\overset{(b)}{\geq} \iint q_\theta(\tau,H)\left(\log p(\tau,H) - \log q_\theta(\tau,H) + \log r(s_H,a_H)d\tau\right)dH - \log(1-\gamma) \tag{18}$$

$$\overset{(c)}{=} \int \sum_{H=0}^{\infty} q(H)q_\theta(\tau \mid H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log p(H) - \log q(H) + \log r(s_H,a_H)d\tau\right) - \log(1-\gamma)$$

$$\overset{(d)}{=} \int q_\theta(\tau \mid H=\infty)\sum_{H=0}^{\infty} q(H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log p(H) - \log q(H) + \log r(s_H,a_H)d\tau\right) - \log(1-\gamma)$$

$$\overset{(e)}{=} \int q_\theta(\tau)\sum_{H=0}^{\infty} q(H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \log p(H) - \log q(H) + \log r(s_H,a_H)d\tau\right) - \log(1-\gamma)$$

$$\overset{(f)}{=} \mathbb{E}_{q_\theta(\tau)}\left[\sum_{H=0}^{\infty} q(H)\left(\left(\sum_{t=0}^{H}\log p(s_{t+1} \mid s_t,a_t) - \log q_\theta(s_{t+1} \mid s_t,a_t)\right) + \cancel{\log(1-\gamma)} + H\log\gamma - \log q(H) + \log r(s_H,a_H)\right)\right] - \cancel{\log(1-\gamma)}$$

$$\overset{(g)}{=} \mathbb{E}_{q_\theta(\tau)}\left[\sum_{H=0}^{\infty}\left(\sum_{t=0}^{H} q(t)\right)\left(\log p(s_{H+1} \mid s_H,a_H) - \log q_\theta(s_{H+1} \mid s_H,a_H)\right) + q(H)\left(H\log\gamma - \log q(H) + \log r(s_H,a_H)\right)\right].$$

For *(a)*, we applied Lemma B.2 and multiplied the integrand by $\frac{q_\theta(\tau|H=H)q(H)}{q_\theta(\tau|H=H)q(H)} = 1$. For *(b)*, we applied Jensen's inequality. For *(c)*, we factored $p(\tau, H) = p(\tau, H)p(H)$ and $q_\theta(\tau, H) = q(\tau \mid H)q(H)$. Note that under the joint distribution $p(\tau, H)$, the horizon $H \sim p(H) = \textsc{Geom}(1-\gamma)$ is independent of the trajectory, τ. For *(d)*, we rewrote the expectation as an expectation over a single infinite-length trajectory and simplified the summand. For *(e)*, we recall the definition $q_\theta(\tau) = q_\theta(\tau = H = \infty)$. For *(f)*, we rewrote the integral as an expectation and wrote out the definition of the geometric distribution, $p(H)$. For *(g)*, we regrouped the difference of dynamics terms. Note that the summation $\sum_{t=0}^{H} q(t)$ is the CDF of the learned discount $q(t)$. Thus, we have obtained the desired result.

We now prove Lemma 3.2, showing that Eq. 4 becomes tight at optimality.

Proof.

$$\begin{aligned}\mathcal{L}_\gamma(\theta) &\overset{(a)}{=} \iint q_\theta(\tau, H)\left(\log p(\tau, H) - \log q_\theta(\tau, H) + \log r(s_H, a_H) d\tau\right) dH - \log(1-\gamma) \\ &\overset{(b)}{=} \iint q_\theta(\tau) q(H \mid \tau)\left(\log p(\tau) + \log p(H) - \log q_\theta(\tau) - \log q(H \mid \tau) + \log r(s_H, a_H) d\tau\right) dH - \log(1-\gamma)\end{aligned} \tag{19}$$

For *(a)*, we undo some of the simplifications above, going back to Eq. 18 For *(b)*, we factor $q_\theta(\tau, H) = q_\theta(\tau)q(H \mid \tau)$ and $p(\tau, H) = p(\tau)p(H)$. At this point, we can solve analytically for the optimal discount distribution, $q(H \mid \tau)$:

$$q^*(H \mid \tau) = \frac{p(H)r(s_H, a_H)}{\sum_{H'=0}^{\infty} p(H')r(s_{H'}, a_{H'})} = \frac{p(H)r(s_H, a_H)}{(1-\gamma)R(\tau)} \tag{20}$$

In the second equality, we substitute the definition of $R(\tau)$. We then substitute Eq. 20 into our expression for $\mathcal{L}_\gamma(\theta)$ and simplify the resulting expression.

$$\begin{aligned}\mathcal{L}_\gamma(\theta) &= \iint q_\theta(\tau) q(H \mid \tau)\left(\log p(\tau) + \cancel{\log p(H)} - \log q_\theta(\tau) - \cancel{\log p(H)} - \cancel{\log r(s_H, a_H)} + \cancel{\log(1-\gamma)} + \log R(\tau) + \cancel{\log r(s_H, a_H)} d\tau\right) dH - \cancel{\log(1-\gamma)} \\ &= \iint q_\theta(\tau) q(H \mid \tau)\left(\log p(\tau) - \log q_\theta(\tau) + \log R(\tau) d\tau\right) dH \\ &= \int q_\theta(\tau)\left(\log p(\tau) - \log q_\theta(\tau) + \log R(\tau)\right) d\tau.\end{aligned}$$

In the final line we have removed the integral over H because none of the integrands depend on H. At this point, we can solve analytically for the optimal trajectory distribution, $q_\theta(\tau)$:

$$q^*(\tau) = \frac{p(\tau)R(\tau)}{\int p(\tau')R(\tau')d\tau'}. \tag{21}$$

We then substitute Eq. 21 into our expression for $\mathcal{L}_\gamma(\theta)$, and simplify the resulting expression:

$$\begin{aligned}\mathcal{L}_\gamma(\theta) &= \int q_\theta(\tau)\left(\cancel{\log p(\tau)} - \cancel{\log p(\tau)} - \cancel{\log R(\tau)} + \log \int p(\tau')R(\tau')d\tau' + \cancel{\log R(\tau)}\right) d\tau \\ &= \log \int p(\tau)R(\tau)d\tau = \log \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right].\end{aligned}$$

We have thus shown that the lower bound $\mathcal{L}_\gamma$ becomes tight when we use the optimal distribution over trajectories $q_\theta(\tau)$ and optimal learned discount $q(H \mid \tau)$. □

B.5 PROOF OF LEMMA B.3

Proof.

$$\log \rho^{\pi_\theta}(s_{t+} = s_g) \overset{(a)}{=} \log \iint p(H)p(\tau \mid H = H)p(s_g \mid s_H, a_H) d\tau dH$$

$$= \log \iint p(H) \frac{p(\tau \mid H = H)}{q_\theta(\tau \mid H = H)} q_\theta(\tau \mid H = H) \frac{p(s_g \mid s_H, a_H)}{q_\theta(s_g \mid s_H, a_H)} q_\theta(s_g \mid s_H, a_H) d\tau dH - \log(1-\gamma)$$

$$\overset{(c)}{\geq} \iint p(H) q_\theta(\tau \mid H = H) \left(\log p(\tau \mid H = H) - \log q_\theta(\tau \mid H = H) + \log p(s_g \mid s_H, a_H) - \log q_\theta(s_g \mid s_H, a_H)\right) d\tau dH - \log(1-\gamma)$$

$$\overset{(d)}{=} \iint p(H) q_\theta(\tau \mid H = \infty) \left(\sum_{t=0}^{H} \log p(s_{t+1} \mid s_t, a_t) - \log q_\theta(s_{t+1} \mid s_t, a_t)\right) + \log p(s_g \mid s_H, a_H) - \log q_\theta(s_g \mid s_H, a_H) d\tau dH - \log(1-\gamma)$$

$$\overset{(d)}{=} \int q_\theta(\tau) \int p(H) \left(\sum_{t=0}^{H} \log p(s_{t+1} \mid s_t, a_t) - \log q_\theta(s_{t+1} \mid s_t, a_t)\right) + \log p(s_g \mid s_H, a_H) - \log q_\theta(s_g \mid s_H, a_H) dH d\tau - \log(1-\gamma)$$

$$\overset{(d)}{=} \int q_\theta(\tau) \sum_{t=0}^{\infty} \gamma^t \left(\log p(s_{t+1} \mid s_t, a_t) - \log q_\theta(s_{t+1} \mid s_t, a_t) + (1-\gamma)(\log p(s_g \mid s_t, a_t) - \log q_\theta(s_g \mid s_t, a_t)\right) d\tau - \log(1-\gamma)$$

$$\overset{(d)}{=} \mathbb{E}_{q_\theta(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t \left(\log p(s_{t+1} \mid s_t, a_t) - \log q_\theta(s_{t+1} \mid s_t, a_t) + (1-\gamma)(\log p(s_g \mid s_t, a_t) - \log q_\theta(s_g \mid s_t, a_t) + (1-\gamma)\log(1-\gamma)\right)\right].$$

□

B.6 A LOWER BOUND FOR GOAL-REACHING TASKS.

Many RL problems can be better formulated as goal-reaching problems, a formulation that does not require defining a reward function. We now introduce a variant of our method for goal-reaching tasks. Using $\rho^\pi(s_{t+})$ to denote the discounted state occupancy measure of policy π, we define the goal-reaching objective as maximizing the probability density of reaching a desired goal s_g:

$$\max_\theta \log \rho^{\pi_\theta}(s_{t+} = s_g). \tag{22}$$

We refer the reader to Eysenbach et al. [14] for a more detailed discussion of this objective. For simplicity, we assume that the goal is fixed, noting that the multi-task setting can be handled by conditioning the policy on the commanded goal. Similar to Lemma 3.1, we can construct a lower bound on the goal-conditioned RL problem:

Lemma B.3. *Let initial state distribution $p_1(s_1)$, real dynamics $p(s_{t+1} \mid s_t, a_t)$, reward function $r(s_t, a_t) > 0$, discount factor $\gamma \in (0, 1)$, and goal g be given. Then the following bound holds for* any *dynamics $q(s_{t+1} \mid s_t, a_t)$ and policy $\pi(a_t \mid s_t)$:*

$$\log p^{\pi_\theta}(s_{t+} = s_g) \geq \mathbb{E}_{q^{\pi_\theta}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \tag{23}$$

where $\tilde{r}_g(s_t, a_t, s_{t+1}) \triangleq (1-\gamma)\log p(s_{t+1} = s_g \mid s_t, a_t) + \log p(s_{t+1} \mid s_t, a_t) - \log q(s_{t+1} \mid s_t, a_t)$.

The proof, presented in Appendix B.3, is similar to the proof of Lemma 3.1. The first term in the reward function, the log probability of reaching the commanded goal one time step in the future, is similar to prior work [32]. The correction term $\log p - \log q$ incentivizes the policy to avoid transitions where the model is inaccurate, and can be estimated using a separate classifier. One important aspect of this goal-reaching problem is that it is entirely data-driven, avoiding the need for any manually-designed reward functions.

C IMPLEMENTATION DETAILS

C.1 ALGORITHMS

Value Iteration (Fig. 4 For the tabular experiments that perform value iteration, we perform Polyak averaging of the policy and learned dynamics model with parameter $\tau = 0.5$. We found that the value iteration version of MnM diverged without this Polyak averaging step. Experiments were stopped when the MnM dynamics model (which depends on the value function) changes by less than 1e-6 across iterations, as measured using an L_0 norm.

	SAC	MBPO	MnM
HalfCheetah-v2	-	40	20
Hopper-v2	-	40	20
Walker2d-v2	-	40	20
dm-cartpole-swingup_sparse	1	1	1
metaworld-window-open-v2	20	20	20
metaworld-door-open-v2	40	40	40
metaworld-reach-v2	20	20	20
metaworld-drawer-open-v2	40	40	40
DClawPoseRandom-v0	20	20	20
DClawTurnRandom-v0	40	40	40
DClawScrewFixed-v0	40	40	40
DClawScrewRandom-v0	40	40	40

Table 1: **Gradient updates per real environment step**: This parameter was separately tuned for each method and each environment.

Q-learning (Fig. 3 For the experiments with Q-learning (both with and without the MnM components), we performed ϵ-greedy exploration with $\epsilon = 0.5$. We used a learning rate of $1e-2$.

SAC for Continuous Control Tasks We used the SAC implementation from TF-Agents [18] with the default hyperparameters.

MBPO for Continuous Control Tasks We implement MBPO on top of the SAC implementation from TF-Agents [18]. Unless otherwise mentioned, we take the default parameters from this implementation. We use an ensemble of 5 dynamics models, each with 4 hidden layers of size 256. The dynamics model predicts the whitened difference between the next state and the current state. That is, to obtain the prediction for the next state, the predictions are scaled by a per-coordinate variance, shifted by a per-coordinate mean, and then added to the current state. These whitened predictions are clipped to have a minimum standard deviation of 1e-5; without this, we found that the MBPO model encounted numerical instabilities. The model is trained using the standard maximum likelihood objective, with all members of the ensemble being trained on the same data. To sample data from this model we perform 1-step rollouts, starting at states visited in the true dynamics. We perform one batch of rollouts in parallel using a batch size of 256. To sample the corresponding action, with probability 50% we take the action that was executed in the true dynamics; with probability 50% we sample an action from the current policy. We found that this modification slightly improves the results of MBPO. We use a batch size of 256. We have two replay buffers: the model replay buffer has size 256e3 and the replay buffer of real experience has size 1e6. At the start of training, we collect 1e4 transitions from the real environment, train the dynamics model on this experience for 1e5 batches, and only then start training the policy. We use a learning rate of 3e-4 for all components. To stabilize learning, we maintain a target dynamics model using an exponential moving average ($\tau = 0.001$), and use this target dynamics model to sample transitions for training. We update the model, policy, and value functions at the same rate we sample experience from the learned model, which is more frequently than we collect experience from the real environment (see Table 1).

MnM for Continuous Control Tasks We implement MnM on top of the SAC implementation from TF-Agents [18]. Unless otherwise mentioned, we take the default parameters from this implementation. Our model architecture is exactly the same as our MBPO implementation, and we follow the same training protocol.

Unlike MBPO, MnM also learns a classifier for distinguishing real versus model transitions. We use a 2 layers neural network with 1024 hidden units in each layer. We found that this large capacity was important for stable learning. We add input noise with $\sigma = 0.1$ while training the classifier. We whiten the inputs to the classifier by subtracting a coordinate-wise mean and dividing by a coordinate-wise standard deviation. When training the classifier, we take samples from both the dynamics model and the target dynamics model as negative examples, finding that this stabilizes learning somewhat.

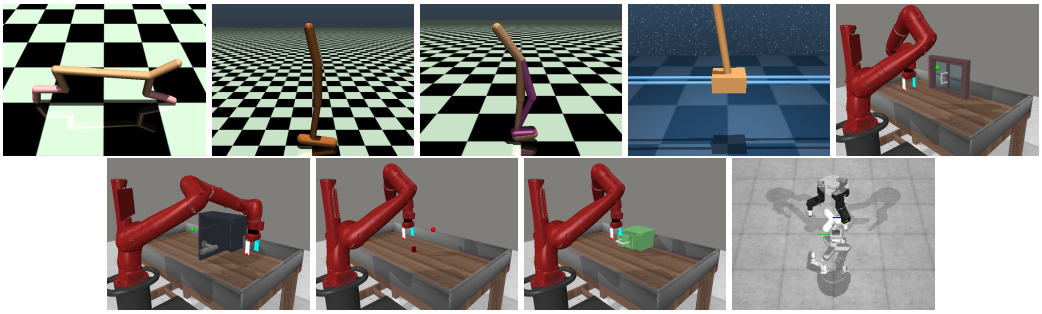

Figure 9: **Environments**: Our experiments look at three locomotion tasks from OpenAI Gym [6], the inverted pendulum task from DM Control [37], four manipulation tasks from Metaworld [42], and four dextrous manipulation tasks from Robel [1]. The Robel tasks use the same dynamics but different reward functions, so we only include an image of one task.

We found that gradient penalties and spectral normalization decreased performance. We found that automatically tuning the classifier input noise also decreased performance. We found that the loss would often plateau around 1e4 batches, but would eventually start decreasing again after 2e4 - 2e5 batches.

Like the MBPO model, we first collect 1e4 transitions of experience from the real environment using a random policy, then train the dynamics model and classifier for 1e5 batches, and only then start updating the policy. Because the Q values are poor at the start of training, we only add the value term to the model loss (resulting in the optimistic dynamics model) after 2e5 batches. To further improve stability, we compute the value term in the model loss by taking the minimum over two target value functions (like TD3 [16]). We update the model, classifier policy, and value functions at the same rate we sample experience from the learned model, which is more frequently than we collect experience from the real environment (see Table 1).

C.2 Environments

Gridworld for Fig. 3. This task is a 10×10 gridworld with obstacles shown in Fig. 3. There are four discrete actions, corresponding to moving to the four adjacent cells. With probability 50%, the agent's action is ignored and a random action is taken instead. The agent starts in the top-left cell. The agent receives a reward of +0.001 at each time step and a reward of +10 when at the goal state. Episodes have 200 steps and we use a discount $\gamma = 0.9$.

Gridworld for Fig. 4 *(Left)*. This task is a 10×10 gridworld with obstacles shown in Fig. 3. There are four discrete actions, corresponding to moving to the four adjacent cells. With probability 90%, the agent's action is ignored and a random action is taken instead. The agent starts in the top-left cell. The rewards depend on the Manhattan distance to the goal: transitions that lead away from the goal have a reward of +0.001, transitions that do not change the distance to the goal have a reward of +1.001, and transitions that decrease the distance to the goal have a reward of +2.001. We use $\gamma = 0.5$ and compute values and returns analytically using value iteration.

Gridworld for Fig. 4 *(Right)*. This task is a 1×3 gridworld with no obstacles. There are four discrete actions corresponding to moving up/down/left/right, but the up/down actions have no effect. The agent starts in the middle state. The only state with stochastic dynamics is the right state (with reward +51): with probability 50%, the agent's action is ignored and the agent is transitioned to the middle state (with reward +1) instead. We use $\gamma = 0.9$ and compute values and returns analytically using value iteration.

With probability 50%, the agent's action is ignored and a random action is taken instead. The agent starts in the top-left cell. The agent receives a reward of +0.001 at each time step and a reward of +10 when at the goal state. Episodes have 200 steps and we use a discount $\gamma = 0.9$.

`HalfCheetah-v2, Hopper-v2, Walker2d-v2.` These tasks are taken directly from the OpenAI benchmark [6] without modification.

dm-cartpole-swingup_sparse. This task is taken directly from the DM Control benchmark [37] without modification.

metaworld-window-open-v2. This task is based on the `window-open-v2` task from the Metaworld benchmark [42]. To increase the difficulty of this task, we set `random_init=True` and use a sparse reward function. The sparse reward function is +0 when the window is within 3 units of the (default) open position and -1 otherwise.

metaworld-door-open-v2. This task is based on the `door-open-v2` task from the Metaworld benchmark [42]. To increase the difficulty of this task, we set `random_init=True` and use a sparse reward function. The sparse reward function is +1 when the window is within 3 units of the (default) open position and +0 otherwise.

metaworld-reach-v2. This task is based on the `reach-v2` task from the Metaworld benchmark [42]. To increase the difficulty of this task, we set `random_init=True` and use a sparse reward function. The sparse reward function is +0 when the window is within 1 units of the goal and -1 otherwise.

metaworld-drawer-open-v2. This task is based on the `drawer-open-v2` task from the Metaworld benchmark [42]. To increase the difficulty of this task, we remove the reward shaping term (`reward_for_caging`) and just optimize the reward for opening the drawer (`reward_for_opening`).

DClawPoseRandom-v0, DClawTurnRandom-v0, DClawScrewFixed-v0, DClawScrewRandom-v0. These tasks are taken directly from the ROBEL benchmark [1] without modification.