Generalized Hidden Parameter MDPs: Transferable Model-Based RL in a Handful of Trials

Christian F. Perez, Felipe Petroski Such, Theofanis Karaletsos
Uber AI Labs
San Francisco, CA 94105
{cfp, felipe.such, theofanis}@uber.com

Abstract

There is broad interest in creating RL agents that can solve many (related) tasks and adapt to new tasks and environments after initial training. Model-based RL leverages learned surrogate models that describe dynamics and rewards of individual tasks, such that planning in a good surrogate can lead to good control of the true system. Rather than solving each task individually from scratch, hierarchical models can exploit the fact that tasks are often related by (unobserved) causal factors of variation in order to achieve efficient generalization, as in learning how the mass of an item affects the force required to lift it can generalize to previously unobserved masses. We propose Generalized Hidden Parameter MDPs (GHP-MDPs) that describe a family of MDPs where both dynamics and reward can change as a function of hidden parameters that vary across tasks. The GHP-MDP augments model-based RL with latent variables that capture these hidden parameters, facilitating transfer across tasks. We also explore a variant of the model that incorporates explicit latent structure mirroring the causal factors of variation across tasks (for instance: agent properties, environmental factors, and goals). We experimentally demonstrate state-of-the-art performance and sample-efficiency on a new challenging MuJoCo task using reward and dynamics latent spaces, while beating a previous state-of-the-art baseline with $> 10\times$ less data. Using test-time inference of the latent variables, our approach generalizes in a single episode to novel combinations of dynamics and reward, and to novel rewards.

1 Introduction

In our pursuit of better learning algorithms, we seek those that can learn quickly across tasks that it encounters during training (called positive *transfer* when there is helpful synergy), and generalize to novel it encounters at test-time. Consider an illustrative problem of an agent with some pattern of broken actuators (agent variation) acting in an environment with changing surface conditions due to weather (dynamics variation), tasked with achieving one of many possible goals (reward variation). We would like a learning algorithm that (1) pools information across observed tasks to learn faster (positive transfer), and generalizes from observed combinations of agent, dynamics, and reward variations to (2) other unseen combinations (*weak* generalization) and (3) novel variations (*strong* generalization) without learning a new policy entirely from scratch (Hu et al. 2018).

To tackle this problem, we introduce Generalized Hidden Parameter MDPs (GHP-MDP) to describe families of MDPs in which dynamics and reward can change as a function of hidden parameters (Section 3.1). This model introduces (multiple) latent variables that capture the factors of variation implicitly represented by tasks at training time. At test time, we infer the MDP by inferring the latent variables that form a latent embedding space of the hidden parameters. This extends and unifies two lines of work: we augment transferable models of MDPs like (Doshi-Velez and Konidaris 2016) with structure on reward and dynamics, and combine it with powerful approaches for learning probabilistic models (Lakshminarayanan, Pritzel, and Blundell 2017) to solve challenging RL tasks.

We propose two variants of latent variable models: one with a shared latent variable to capture all variations in reward and dynamics, and a structured model where latent variables may factorize causally. Our intention is to afford GHP-MDPs the use of prior knowledge and inductive biases that could improve sample efficiency, transfer, and generalization.

In our experiments, agents are trained on a small subset of possible tasks—all related as instances from the same GHP-MDP—and then generalize to novel tasks from the same family via inference. We show that this method improves on the state-of-the-art sample efficiency for complex tasks while matching performance of model-free meta-RL approaches (Section 5) (Finn, Abbeel, and Levine 2017; Rakelly et al. 2019). Notably, our approach also succeeds with a fairly small number of training tasks, requiring only a dozen in these experiments.

2 Model-based RL

We first consider a reinforcement learning (RL) problem described by a Markov decision process (MDP) comprising a state space, action space, transition function, reward function, and initial state distribution: $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho_0\}$ (Bellman 1957). We define a ”task” (or ”environment”) τ to be a MDP from a set of MDPs that share $\mathcal{S}$ and $\mathcal{A}$ but differ in one or more of $\{\mathcal{T}, \mathcal{R}, \rho_0\}$.

In model-based RL, the agent uses a model of the transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ in order to maximize rewards over some task-dependent time horizon H. For a (stochastic) policy π_θ parameterized by θ, the goal is to find an optimal policy π^* that maximizes the expected reward,

$$\pi^*(\mathbf{a}|\mathbf{s}) = \text{argmax}_\theta\, \mathbb{E}_{\mathbf{a}_t \sim \pi_\theta(\cdot|\mathbf{s}_t)} \sum_{t'=t}^{t+H-1} \mathcal{R}(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$
$$\text{s.t. } \mathbf{s}_{t+1} \sim \mathcal{T}(\mathbf{s}_t, \mathbf{a}_t),$$

where $\mathcal{T}$ acts as a probability distribution over next states in a stochastic environment.

While it is common to assume a known reward function $\mathcal{R}$ and even transition function $\mathcal{T}$, one can simultaneously learn an approximate model of both the dynamics and reward,

$$\begin{aligned}\mathcal{T} &\approx \tilde{\mathcal{T}} \doteq p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\\ \mathcal{R} &\approx \tilde{\mathcal{R}} \doteq p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\end{aligned}$$

with parameters θ and ω using data collected from the environment $\mathcal{D} = \{(\mathbf{s}_t^{(n)}, \mathbf{a}_t^{(n)}, \mathbf{s}_{t+1}^{(n)}, r_{t+1}^{(n)})\}_{n=1}^N$. In this work, we do not learn a parametric policy π_θ, but instead use model predictive control to perform planning trajectories sampled from the learned models (see Section 3.5.)

The RL problem is then decomposed into two parts: learning models from (limited) observations, and (approximate) optimal control given those models. By iterating between model learning and control, the agent uses the improved model to improve control and vice versa. This basic approach is effective for a single environment, but is not designed to learn across multiple related environments. This is a key limitation our approach overcomes, described in Section 3.1.

Another shortcoming of this approach is the tendency for expressive models (e.g., neural networks) to overfit to observed samples and produce overconfident and erroneous predictions (also called "model bias" (Deisenroth and Rasmussen 2010)). The result is a sub-optimal policy, worse sample efficiency, or both. This problem is exacerbated for transfer learning scenarios, in which an agent that overfits to training tasks fails to generalize to novel tasks at test time. Bayesian learning for neural networks can properly account for model uncertainty given limited data, but can be difficult to scale to high-dimensional states $\mathcal{S}$ and actions $\mathcal{A}$ (Deisenroth 2011) and is burdened by the hardness of representation for the posterior over weights. As a tractable alternative, we extend *Deep Ensembles* of neural networks (Lakshminarayanan, Pritzel, and Blundell 2017), which perform well on isolated tasks (Chua et al. 2018), to learn transferable agents without changing the underlying model.

2.1 Learning probabilistic models

In order to perform model-based control, an agent requires knowledge of the dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and reward $p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. When these underlying mechanisms are unknown, one can resort to learning parameterized models $p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and $p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. Because environments can be stochastic, we use a generative model of dynamics and reward. Because these are continuous quantities, each can be modeled with a Gaussian likelihood. The dynamics, for example, can be parameterized by mean $\boldsymbol{\mu}_\theta$ and diagonal covariance $\boldsymbol{\Sigma}_\theta$ produced by a neural network with parameters θ (and similarly for the reward model using parameters ω),

$$\begin{aligned}p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) &= \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{s}_t, \mathbf{a}_t), \boldsymbol{\Sigma}_\theta(\mathbf{s}_t, \mathbf{a}_t))\\ p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) &= \mathcal{N}(\boldsymbol{\mu}_\omega(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}),\\ &\qquad \boldsymbol{\Sigma}_\omega(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}))\,.\end{aligned} \tag{1}$$

From these building blocks, we construct a joint probability distribution over trajectories and jointly optimize model parameters $\{\theta, \omega\}$ given data $\mathcal{D}$. (See Section 3.4 for the learning objective involving $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$.)

Also, as commonly done elsewhere, the neural network prediction target is actually the change in the states $\Delta_s = \mathbf{s}_{t+1} - \mathbf{s}_t$ given the state and action: $p_\theta(\Delta_s|\mathbf{s}_t, \mathbf{a}_t; \theta)$.

2.2 Ensembles of networks

In order to be robust to model misspecification and handle the small data setting, one can model uncertainty about parameters θ and ω and marginalize over their posterior after observing dataset $\mathcal{D}$ to obtain the predictive distributions

$$\begin{aligned}p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathcal{D}) &= \int p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) p(\theta|\mathcal{D}) d\theta\\ p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathcal{D}) &= \int p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) p(\omega|\mathcal{D}) d\omega\,.\end{aligned} \tag{2}$$

Learning these models can be posed as inference of model parameters given observed data, e.g. using a Bayesian Neural Network (BNN) which entails inferring the posteriors $p(\theta|\mathcal{D})$ and $p(\omega|\mathcal{D})$. As is usually the case in inference for such models, computing the exact posterior is intractable. A practical way to approximate the predictive distribution of the network is by capturing uncertainty through frequentist ensembles of models, in which each ensemble member is trained on a shuffle of the training data (Lakshminarayanan, Pritzel, and Blundell 2017). For an ensemble with M members and the collection of all network parameters $\Theta \doteq \{\theta_1, ..., \theta_M\}$, we define a model of the next state predictive distribution as a mixture model as follows:

$$\begin{aligned}p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t; \Theta) &= \frac{1}{M} \sum_{\theta \in \Theta} p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\\ &\approx p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\,.\end{aligned} \tag{3}$$

The reward model follows similarly,

$$\begin{aligned}p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}; \Omega) &= \frac{1}{M} \sum_{\omega \in \Omega} p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\\ &\approx p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\,,\end{aligned} \tag{4}$$

also including its dependence on $\mathbf{s}_{t+1}$, whose values are observed from training data, but at test-time are the result of predictions from they dynamics model of (3).

3 Modeling with hidden parameters

Our goal is to learn a model of the system we would like to control and then plan on that model in order to achieve high reward on the actual system. For sufficiently complex systems and finite training data, we expect that the model can only approximate the real system. Furthermore, the real system may differ in significant ways from the system our models were trained on, as when a robot actuator force degrades

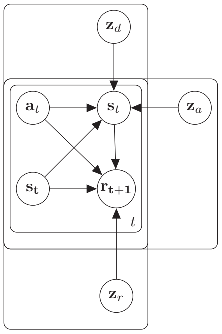

Figure 1: Probabilistic graphical model of MDP with structured latent variables for environment, agent, and reward function variations.

over time, unless the conditions were deliberately included in training. However, it is unreasonable to train a model across all possible conditions an agent may encounter. Instead, we propose a model that learns to account for the causal factors of variation observed across tasks at training time, and then infer at test time the model that best describe the system. We hypothesize that explicitly incorporating these factors will facilitate generalization to novel variations at test time.

A Hidden Parameter MDP (HiP-MDP), first formalized in (Doshi-Velez and Konidaris 2016), describes a family of MDPs in which the transition dynamics are parameterized by hidden parameters $\eta \in \mathbb{R}^n$, here expressed as $\mathcal{T}_\eta$ or $\mathcal{T}(\cdot\,; \eta)$. In dynamical systems, for example, parameters can be physical quantities like gravity, friction of a surface, or the strength of a robot actuator. Their effects are ”felt” but not directly observed; η is not part of the observation space. Prior work using meta-learning for adaptive dynamics problems proposes agents that learn across *distinct* tasks τ_i where the transition dynamics varies according to some problem-specific distribution $\mathcal{T}_i \sim p(\mathcal{T})$. For example, by changing the terrain, agent ability, or observations for tasks during training, agents can learn to adjust to novel yet similar conditions at test time (Clavera et al. 2018; Fu, Levine, and Abbeel 2016). In contrast, we treat each task as an instance of the same HiP-MDP with different hidden parameters $\eta \sim p(\eta)$ affecting the transition function $\mathcal{T}_\eta$. Previous work showed that for simple, low-dimensional systems, agents can infer an effective representation of the hidden parameters, and generalize the dynamics to novel parameter settings (Killian et al. 2017; Sæmundsson, Hofmann, and Deisenroth 2018). Yet neither the HiP-MDP nor adaptive dynamics methods account for different task rewards.

Consider a multi-task setting, in which an agent learns across tasks τ_i where only the reward function $\mathcal{R}_i$ varies, for example, performing tasks that require navigation to a goal position, or movement in a certain direction or target velocity (Finn, Abbeel, and Levine 2017). In our formulation, all of these tasks come from a parameterized MDP in which the reward function $\mathcal{R}_\eta$ or $\mathcal{R}(\cdot; \eta)$ depends on hidden parameters η that determine the goal/reward structure.

3.1 Generalized Hidden Parameter MDPs

We denote a set of tasks/MDPs with transition dynamics $\mathcal{T}_\eta$ and rewards $\mathcal{R}_\eta$ that are fully described by hidden parameters η as a *Generalized Hidden Parameter MDP* (*GHP-MDP*). A GHP-MDP includes settings in which tasks can exhibit multiple factors of variation. For example, consider a robotic arm with both an unknown goal position g and delivery payload m. This problem can be modeled as drawing tasks from a distribution $\eta_g, \eta_m \sim p(\eta)$ with effects on both the transition $\mathcal{T}_{\eta_m}$ and reward function $\mathcal{R}_{\eta_g}$. Additional factors of variation can be modeled with additional parameters, for example, by changing the size of the payload η_l. Note that we generalize η to describe more than just physical constants. All of these hidden parameters are treated as latent variables $\{\mathbf{z}_i \in \mathbb{R}^{d_i} : i = 1, \ldots, c\}$, and we express the GHP-MDP as a latent variable model.

We pose jointly learning the two surrogate models of (1) and latent embeddings $\mathbf{z}_i$ via the maximization of a variational lower bound over data collected from a small set of training tasks (see Section 3.4 for the inference objective.) At test-time, only the parameters ϕ for the approximate posterior $p_\phi(\mathbf{z}_i|\mathcal{D})$ of the latent variables are learned via inference. Note that the latent variables $\mathbf{z}_i$ are an *embedding* of the true parameters η, and in general, are not equal to η, nor even have the same dimensions (i.e., $d_i \neq n$).

In Section 3.2, we describe the simplest probabilistic model of a GHP-MDP that uses a single continuous latent variable $\mathbf{z} \in \mathbb{R}^D$ to model hidden parameters of both the dynamics and reward. Because a single $\mathbf{z}$ jointly models all unobserved parameters, we call this the *joint latent variable (joint LV)* model. In Section 3.3, we extend the model to multiple latent variables $\{\mathbf{z}_d, \mathbf{z}_a, \mathbf{z}_r\} \in \mathbb{R}^D$ (shown graphically in Figure 1), one for each aspect of the task that is known to vary in the training environments. In other words, we encode our prior knowledge about the (plated) structure of the tasks into the structure of the model; hence, we refer to this as the *structured latent variable (structured LV)* model. In this paper, we assume that latent variables are specified *a prior* to be either shared or distinct. We leave learning how to disentangle these factors to future work.

3.2 Joint latent variable model

For any GHP-MDP, we can model the dynamics and reward hidden parameters jointly with a single latent variable $\mathbf{z} \in \mathbb{R}^D$. The model for episode return $\mathbf{R} = \sum \mathbf{r}_{t+1}$ for a trajectory decomposed into partial rewards $\mathbf{r}_{t+1}$ is

$$p(\mathbf{R}|\mathbf{s}_{0:T}, \mathbf{a}_{0:T-1}, \mathbf{z}) = \prod_{t=0}^{T-1} p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z})\,, \quad (5)$$

where T is the episode length. The resulting joint model mapped over trajectories $p(\mathbf{s}_{0:T}, \mathbf{a}_{0:T-1}, \mathbf{R}, \mathbf{z})$ is

$$\begin{aligned} p(\mathbf{z})p(\mathbf{s}_0) \prod_{t=0}^{T-1} \big[& p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}) \\ & p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}) p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}) \big] \end{aligned} \quad (6)$$

The key feature of this model is the same latent variable $\mathbf{z}$ conditions both the dynamics and the reward distributions.

The priors for auxiliary latent variable are set to simple normal distributions, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and initial state distribution $p(\mathbf{s}_0)$ to the environment simulator. Again we note that $p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}) = \pi^*(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})$ via planning (Section 3.5).

Common meta-RL tasks that vary only reward/task, e.g. by varying velocities or goal directions (Finn, Abbeel, and Levine 2017), can be tackled with a simplified joint LV model that conditions only the reward model. Combined with inference at test-time, this approach is an alternative to meta-learning via MAML or RNNs (Duan et al. 2017). Our experiments (Section 5) demonstrate the full joint LV model (conditioning both dynamics and reward models) can solve one such reward-variation task.

3.3 Structured latent variable model

In the previous section, we introduced a global latent variable $\mathbf{z}$ that is fed into both the dynamics and reward model. Here, we extend this idea and introduce multiple *plated variables* which constitute the structured latent space of the GHP-MDP. Separate latent spaces for dynamics and reward are intuitive because agents may pursue different goals across environments with different dynamics.

Consider a structured model with two latent variables $\mathbf{z}_d \in \mathbb{R}^{D_1}$ and $\mathbf{z}_r \in \mathbb{R}^{D_2}$ to separately model hidden parameters in the dynamics $\mathcal{T}(\cdot\,; \mathbf{z}_d)$ and reward $\mathcal{R}(\cdot\,; \mathbf{z}_r)$. The joint model $p(\mathbf{s}_{0:T+1}, \mathbf{a}_{0:T}, \mathbf{R}, \mathbf{z}_d, \mathbf{z}_r)$, including the action distribution implied by control, is

$$p(\mathbf{z}_d)p(\mathbf{z}_r)p(\mathbf{s}_0) \prod_{t=0}^{T-1} \big[p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}_r) \quad (7)$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_d)p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}_r, \mathbf{z}_d)\big]\,.$$

This structure facilitates solving tasks where both of these aspects can vary independently. Typically, only one or the other is varied in meta-RL tasks for continuous control (Clavera et al. 2018; Finn, Abbeel, and Levine 2017; Rakelly et al. 2019), and so we introduce new tasks in Section 5 to test the weak and strong generalization ability of this modeling choice.

More generally we may have c arbitrary plated contexts, such as agent, dynamics, reward variation, etc. Then for the set of latent variables $\{\mathbf{z}_1, \ldots, \mathbf{z}_c\}$, each explains a different factor of variation in the system, implying $p(\mathbf{z}) = \prod p(\mathbf{z}_c)$. This allows the model to have separate degrees of freedom in latent space for distinct effects. Note that the use of plated variables implies that tasks will have known factors of variation (but unknown values and effects) at training time only. In practice, this is case when training on a simulator.

By factorizing the latent space to mirror the causal structure of the task, the structured LV model can also more efficiently express the full combinatorial space of variations. For example, with $c = 3$ factors of variation and 10 variations for each η_i for $i \in \{1, 2, 3\}$, the latent space must generalize to $10 \times 10 \times 10 = 10^3$ combinations. Learning a global latent space would require data from some non-trivial fraction of this total. In contrast, a structured space can generalize from $10 + 10 + 10 = 30$. We examine this generalization ability experimentally in Section 5.3.

3.4 Training and inference

Each step/episode of training consists of two phases: collect an episode of trajectories $\mathcal{D}_k$ for each task via planning (Algorithm 1), and infer model parameters and latent variables using *all* collected data via SGD. The goal of the inference (learning) step is to maximize the marginal likelihood of observed transitions with respect to θ and ϕ. For the *joint latent variable model*, the intractable distribution $p(\mathbf{z}|\mathcal{D})$ is approximated with $q_\phi(\mathbf{z})$ parameterized by a diagonal Gaussian where $\phi = \{\mu, \Sigma\}$. We then maximize the evidence lower bound (**ELBO**) to the marginal log-likelihood:

$$\log p(\mathcal{D}) = \sum_{t=0}^{T} \big[\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\big]$$
$$\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})} \Bigg[\sum_{t=0}^{T} (\log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}) +$$
$$\log p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}))\Bigg] - \mathrm{KL}\big(q_\phi(\mathbf{z})||p(\mathbf{z})\big). \quad (8)$$

For simplicity, we choose the prior $p(\mathbf{z})$ and variational distribution $q_\phi(\mathbf{z})$ to be Gaussian with diagonal covariance. We can use this criterion during the training phase to jointly update network parameters Θ and variational parameters ϕ capturing beliefs about latent variables.

In practice, we use stochastic variational inference (Ranganath, Gerrish, and Blei 2013; Kingma and Welling 2014) and subsample in order to perform inference and learning via gradient descent, yielding the loss function:

$$\mathcal{L}(\theta, \omega, \phi) =$$
$$-\frac{1}{M}\sum_{m=1}^{M}\sum_{t=0}^{T}\big[\log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}^{(m)})$$
$$+ \log p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}^{(m)})\big] \quad (9)$$
$$+\mathrm{KL}\big(q_\phi(\mathbf{z})||p(\mathbf{z})\big)$$

with $\mathbf{z}^{(m)} \sim q_\phi(\mathbf{z})$ and number of samples $M = 2$. Recall that both models are ensembles and each network in the ensemble is optimized independently, but the variational distribution is shared according the relationship between tasks. During training, we minimize $\mathcal{L}(\theta, \omega, \phi)$, and at test time, reset q_ϕ to the prior and minimize with respect to ϕ only.

For structured latent variable models with plated contexts, (9) can be extended to multiple latent variables. For a graphical model (Fig. 1) with three factors of variation—environment, agent, and reward—and variational parameters $\Phi \doteq \{\phi_d, \phi_a, \phi_r\}$ for each, the loss function becomes

$$\mathcal{L}(\theta, \omega, \phi) =$$
$$-\frac{1}{M}\sum_{m=1}^{M}\sum_{t=0}^{T}\big[\log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_d^{(m)}, \mathbf{z}_a^{(m)})$$
$$+ \log p_\omega(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}_r^{(m)})\big] \quad (10)$$
$$+ \mathrm{KL}(q_{\phi_d}(\mathbf{z}_d)||p(\mathbf{z}_d))$$
$$+ \mathrm{KL}(q_{\phi_a}(\mathbf{z}_a)||p(\mathbf{z}_a))$$
$$+ \mathrm{KL}(q_{\phi_r}(\mathbf{z}_r)||p(\mathbf{z}_r))\,.$$

Algorithm 1 Learning and control with MPC and Latent Variable Models

```
1:  Initialize data $\mathcal{D}$ with random policy
2:  for Episode m = 1 to M do
3:      Sample an environment indexed by $k$
4:      If learning, train a dynamics model $p_\theta$ and reward model $p_\omega$ with $\mathcal{D}$ using (9) or (10)
5:      Initialize starting state $\mathbf{s}_0$ and episode history $\mathcal{D}_k = \varnothing$
6:      for Time t = 0 to TaskHorizon do                                                    ▷ MPC loop
7:          for Iteration i = 0 to MaxIter do                                               ▷ CEM loop
8:              Sample actions $\mathbf{a}_{t:t+h} \sim \mathrm{CEM}(\cdot)$
9:              Sample latent $\mathbf{z}^{(p)} \sim q_\phi(\mathbf{z})$ for each state particle $s_p$
10:             Propagate next state $\mathbf{s}_{t+1}^{(p)} \sim p_\theta(\cdot|\mathbf{s}_t^{(p)}, \mathbf{a}_t, \mathbf{z}^{(p)})$ using TS-$\infty$    ▷ See (Chua et al. 2018)
11:             Sample reward $\mathbf{r}_{t+1}^{(p)} \sim p_\omega(\cdot|\mathbf{s}_t^{(p)}, \mathbf{a}_t, \mathbf{s}_{t+1}^{(p)}, \mathbf{z}^{(p)})$ for each particle trajectory
12:             Evaluate expected return $G_t = \sum_{\tau=t}^{t+h} {}^1/_P \sum_{p=1}^{P} \mathbf{r}_{\tau+1}^{(p)}$
13:             if $\mathrm{any}(\mathrm{early\_termination}(\mathbf{s}_t^{(p)}))$ then
14:                 $G_t \leftarrow \mathrm{done\_penalty}$                              ▷ Hyperparameter for early termination
15:             end if
16:             Update $\mathrm{CEM}(\cdot)$ distribution with highest reward trajectories
17:         end for
18:         Execute first action $\mathbf{a}_t$ determined by final $\mathrm{CEM}(\cdot)$ distribution
19:         Record outcome $\mathcal{D}_t \leftarrow \{(\mathbf{s}_t, \mathbf{a}_t)), (\mathbf{s}_{t+1}, \mathbf{r}_{t+1})\}$
20:         Record outcome $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \mathcal{D}_t$
21:         If test-time, update approximate posterior $q_\phi(\mathbf{z}|\mathcal{D}_t)$ using (9)
22:     end for
23:     Update data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_k$
24: end for
```

3.5 Control with Latent Variable Models

Given a learned dynamics model, agents can plan into the future by recursively predicting future states $\mathbf{s}_{t+1}, ..., \mathbf{s}_{t+h}$ induced by proposed action sequences $\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+h}$ such that $\mathbf{s}_{t+1} \sim \widetilde{\mathcal{T}}(\mathbf{s}_t, \mathbf{a}_t)$. If actions are conditioned on the previous state to describe a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$, then planning becomes learning a policy π^* to maximize expected reward over the predicted state-action sequence. A limitation of this approach is that modeling errors are compounded at each time step, resulting in sub-optimal policies when the learning procedure overfits to the imperfect dynamics model. Alternatively, we use *model predictive control (MPC)* to find the action trajectory $\mathbf{a}_{t:t+H}$ that optimizes $\sum_t^{t+H-1} \mathbb{E}_{q_\phi(\mathbf{z})} \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a}_t)}[p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z})]$ at run-time (Camacho and Bordons 2013), using $\mathbf{s}_{t+1}$ predicted from the learned model (Algorithm 1, line 10). At each time step, the MPC controller plans into the future, finding a good trajectory over the planning horizon H but applying only the first action from the plan, and re-plans again at the next step. Because of this, MPC is better able to tolerate model bias and unexpected perturbations.

Algorithm 1 includes a control procedure that uses the cross-entropy method (CEM) as the optimizer for an MPC controller (De Boer et al. 2005). On each iteration, CEM samples 512 proposed action sequences $\mathbf{a}_{t:t+H-1}$ from H independent multivariate normal distributions $\mathcal{N}(\mathbf{a}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, one for each time step in the planning horizon (line 8), and calculates the expected reward for each sequence. The top 10% performing of these are used to update the proposal distribution mean and covariance. However, evaluating the expected reward exactly is intractable, so we use a particle-based approach called trajectory sampling (TS) from (Chua et al. 2018) to propagate the approximate next state distributions. We adapt the TS+CEM algorithm to incorporate beliefs about the MDP given data observed so far: Each state particle $\mathbf{s}_t^{(p)}$ uses a sample of each latent variable $\mathbf{z}^{(p)} \sim q_\phi(\mathbf{z})$ so that planning can account for their effect on the dynamics and reward models.

At test time, we skip line 4 to keep the neural networks fixed. The algorithm iterates between acting in the environment at step t and inferring $p(\mathbf{z}|\mathcal{D}_t)$ in order to align the dynamics and reward models with the current system as new information is collected. In order to plan when episodes can terminate early due to constraints set by the environment (e.g., when MuJoCo Ant or Walker2d falls over), we set cumulative rewards for particle trajectories that violate those constraints to a fixed constant. This hyperparameter is set to 0 during training to allow exploration, and -100 at test time for more conservative planning.

4 Related Work

Transfer learning and learning transferable agents has a long history in reinforcement learning; see (Taylor and Stone 2009; Lazaric 2012) for a survey. Recent prior work on latent variable models of MDPs focused on models of dynamics with small discrete action spaces (Doshi-Velez and Konidaris

2016; Killian et al. 2017; Yao et al. 2018). We extend Hidden Parameter MDPs introduced in (Doshi-Velez and Konidaris 2016) to the more general case including reward modeling and accounting for potentially multiple factors of variation. Latent dynamics models for hard continuous control tasks were used in (Perez, Such, and Karaletsos 2018), and using Gaussian Processes in (Sæmundsson, Hofmann, and Deisenroth 2018), albeit under significantly less challenging experimental conditions. Another notable use is latent skill embeddings in robotics for adaptation to different goals (Hausman et al. 2018). In contrast, *DeepMDP* models a single task/MDP entirely in a latent space (Gelada et al. 2019), and can be merged to form a Deep-GHP-MDP. Similarly, (Hafner et al. 2019) demonstrates latent space planning from pixel observations, and has some multi-task ability when the tasks appear different. (Zhang, Satija, and Pineau 2018) also learns and plans in a latent space with learned dynamics and reward models, and explores transfer using prior learned encoders on slightly perturbed tasks. The problem of learning and generalization across combinations of reward and dynamics in discrete action spaces on visual domains was also tackled in (Hu et al. 2018) using factorized policies and complementary embeddings of reward and dynamics.

Meta-learning, or learning to learn, is one solution that enables RL agents to learn quickly across different or non-stationary tasks (Schmidhuber 1987; Ravi and Larochelle 2017; Al-Shedivat et al. 2017; Finn, Abbeel, and Levine 2017). Recently, meta-learning has been used to adapt a dynamics model (Clavera et al. 2018) for model-based control to changing environments (but does not model reward or solve multiple tasks), or to learn a policy (Rothfuss et al. 2018; Rakelly et al. 2019) that adapts by adjusting the model or policy in response to recent experience. Extensions can continuously learn new tasks online (Nagabandi, Finn, and Levine 2019). However, model-free meta-RL methods can require millions of samples and dozens of training tasks. One can also simply adapt a neural network online at test-time via SGD without MAML for one/few-shot learning (Fu, Levine, and Abbeel 2016). Learning across tasks with common dynamics has been approached with meta-reinforcement learning (Finn, Abbeel, and Levine 2017) or using successor features (Barreto et al. 2017).

5 Experiments

The GHP-MDP conceptually unifies various RL settings, including multi-task RL, meta-RL, transfer learning, and test-time adaptation. In these experiments, we wish to 1) demonstrate inference on a GHP-MDP with multiple factors of variation, 2) compare the sample efficiency and performance of our model-based implementation of the GHP-MDP on hard continuous control tasks with these flavors, and 3) explore its ability to generalize to novel tasks at test time.

5.1 Didactic example with multiple factors

To demonstrate inference over a structured latent variable model for a GHP-MDP, we construct a toy example where the goal is to infer the hidden parameter values at test time in order to facilitate accurate planning given the parameterized dynamics function.

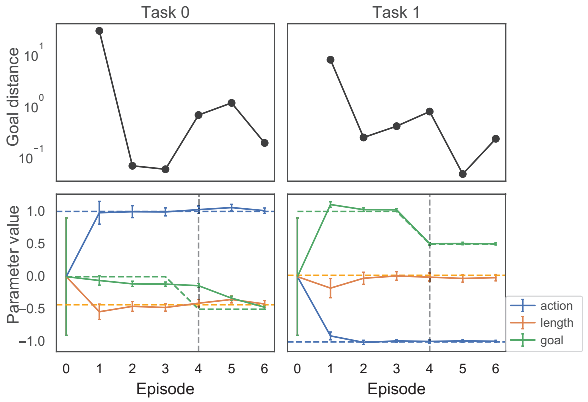

Figure 2: Inference of latent variables when reward hidden parameters are changing but dynamics are not. Left and right columns are two different tasks. **Top:** Distance to the goal tip position for two CartPole tasks with different hidden parameters (pole length $\in \{0.5, 0.7\}$.) The first episode is a random policy. **Bottom:** Mean and standard deviation of the posterior of latent variables (solid), and the true values (dashed). Episode 0 shows the priors on latents before inference.

For this experiment, we use a modified version of `CartPoleSwingUp` with a reward function proportional to the distance between the tip of the pole and the desired position $(x_{\text{goal}} - x_{\text{tip}})^2$ (Sæmundsson, Hofmann, and Deisenroth 2018). The transition function $T(\mathbf{s}_t, \mathbf{a}_t; \eta_a, \eta_l, \eta_x)$ takes as parameters: η_a that scales the action/control signal (blue), the length of the pole η_l (orange), and the position of the pole tip η_x (green; Figure 2). We model the tasks by replacing the hidden parameters η in the transition model T_η with corresponding latent variables $\mathbf{z}_a$, $\mathbf{z}_l$, $\mathbf{z}_x$. In order to use generic priors for missing information, e.g. $p(\mathbf{z}) = \mathcal{N}(0, 1)$, we also model unknown positive parameter values η with latent variables $\mathbf{z}$ via the softplus: $\eta = \log(1 + \exp(\mathbf{z}))$.

The experiment consists of only two tasks with different hidden parameters (Figure 2 left and right columns). Latent variables are inferred using mean field variational inference with Gaussian priors and variational distributions, and perform control using random search. Because there is no model learning, we can immediately infer latent variables given data. After collecting data for 200 steps using a random policy (episode 1), inference yields accurate estimates (Figure 2; bottom) resulting in good control (Figure 2; top).

To test the system, the goal position is suddenly changed in episode 4 and see a corresponding change in only the corresponding latent variable when continuing inference. Thus, we demonstrate the ability of a simplified structured latent variable model to properly disentangle variations in an environment at test time through the usage of inference alone and even adapt on the fly to targeted changes by inferring the right component.

5.2 GHP-MDP for continuous control

We evaluate both the joint and structured LV model with a total of 8 latent dimensions using experiments in the MuJoCo

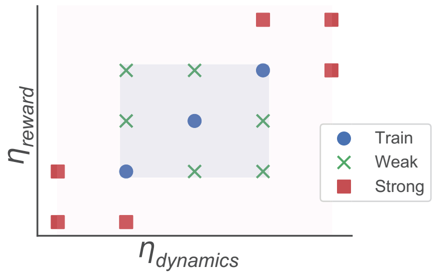

Figure 3: Hidden parameters for training tasks represented by blue dots. Weak generalization requires an agent to work on new combinations (green crosses). Strong generalization requires extrapolation to new hidden parameters (red squares).

Ant environment, a challenging benchmark for model-based RL (Todorov, Erez, and Tassa 2012).

Because the GHP-MDP models both dynamics and reward, we introduce novel tasks with up to two factors of variation. In `DirectionAnt`, the agent receives reward proportional to the portion of its velocity along the goal direction. This is an example of a multi-task RL setting, often used in meta-reinforcement learning benchmarks, in which tasks τ_i have common dynamics $\mathcal{T}$ but a unique reward function $\mathcal{R}_i$. In `CrippledLegDirectionAnt`, agents must learn 1) which of four legs is crippled and does not respond to actions, and 2) which of eight directions to travel. In both experiments, directions are the eight cardinal plus intercardinal directions. The method for dividing tasks into training, test, and holdout sets is described below.

We experimentally evaluate the joint model from Section 3.2 and structured LV model from Section 3.3. The joint model has a global $\mathbf{z}$ that conditions both dynamics and reward models. The structured model has separate latent variables for dynamics $\tilde{T}(\cdot\,; \mathbf{z}_d)$ and reward model $\tilde{\mathcal{R}}(\cdot\,; \mathbf{z}_r)$. Latent variables are 4-D per factor of variation $\mathbf{z}_i \in \mathbb{R}^4$ for $i \in \{1, \ldots, K\}$; the joint model has the same total dimensionality in one variable $\mathbf{z} \in \mathbb{R}^{4K}$ for K factors of variation. The architecture for all experiments is an ensemble of 5 neural networks with 3 hidden layers of 256 units for the dynamics model, and 1 hidden layer of 32 units for the reward model. We report results averaged across 5 seeds using 95% bootstrapped confidence intervals. yielding new combinations with an unseen factors. Experimental results are divided into three categories to probe questions/hypotheses about different flavors of generalization:

- *Transfer* that occurs when learning across tasks is faster than learning each task individually. We compare the learning curves of agents trained across tasks to those of specialists trained per task in Section 5.3.
- *Weak generalization* that requires performing well on a task that was not seen during training but has closely related dynamics and/or reward. Meta-RL commonly assumes tasks at meta-test time are drawn from the same distribution as meta-training, and so falls under this umbrella. In Section 5.3, we test on novel *combinations* of crippled leg and goal direction in `CrippledLegDirectionAnt` that were not observed during training.
- *Strong generalization* that requires performing well on a task with dynamics and/or reward that is outside what was seen during training. This setting falls under transfer learning or online adaptation, in which an agent leverages previous training to learn more quickly on a new out-of-distribution task/environment. In Section 5.3, we test on a novel goal direction not observed during training in `DirectionAnt` and `CrippledLegDirectionAnt`.

Within the GHP-MDP formalism, the last two types of generalization can be visualized as in- or out-of-distribution in the space of hidden parameters, as shown in Figure 3. Another point of interest is how well a structured LV model will generalize compared to the joint LV model. Intuitively, separating the latent variables along causal relationships ought to improve (strong) generalization when samples are scarce.

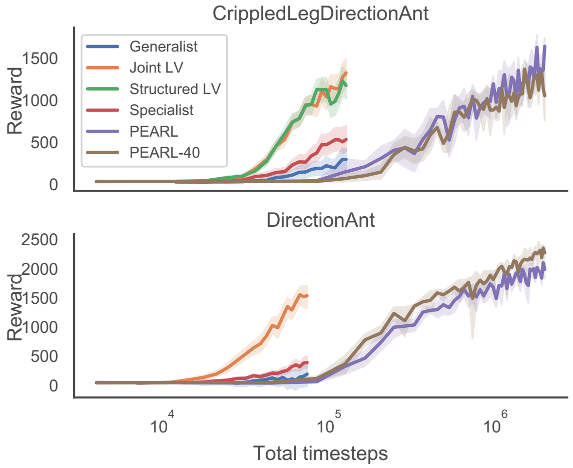

Figure 4: Learning curves (semi-log scale) for (bottom) `DirectionAnt` and (top) `CrippledLegDirectionAnt`. ”Joint LV” in the legend refers to the only LV model in the bottom panel.

We compare our method to three baselines. The **Generalist** is a ensemble-model-based baseline (Chua et al. 2018) that learns dynamics and rewards but lacks latent variables. This baseline is a negative control; its failure confirms that the environments are challenging enough to require additional modeling complexity introduced by the GHP-MDP. The **Specialist** is the same model as the Generalist but trains on each task *individually*, providing a benchmark for per-task sample efficiency. **PEARL** is a sample-efficient off-policy meta-RL algorithm that also uses latent ”context” variables and amortized inference (Rakelly et al. 2019), and reports state-of-the-art sample efficiency on continuous control meta-RL tasks like `DirectionAnt`. We note that PEARL can be posed as the model-free analogue of the joint LV model, using an inference network (like a VAE) instead of SVI. (We know of no analogue to the structured LV model.) We allow PEARL $\approx 2M$ samples, up to $\approx 15\times$ the training data as our method. To compare generalization ability when learning

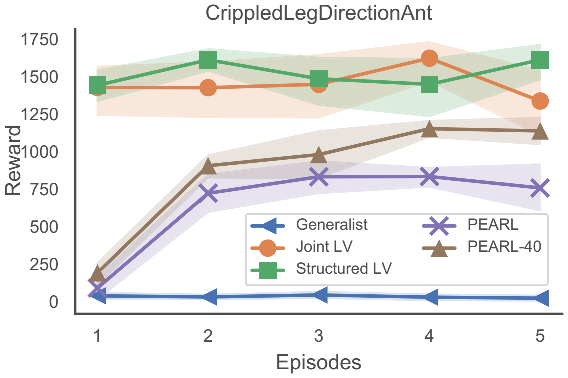

Figure 5: Episode reward on novel combinations.

from few tasks, PEARL learns from the same number of tasks as our model. Because reward varies across tasks in our experiment, we do not compare against adaptive model-based methods like (Clavera et al. 2018) that do not model reward. Because meta-RL methods like PEARL are usually trained with many more tasks, an additional baseline **PEARL-40** trains on 40 out of 100 possible tasks. We compare PEARL and PEARL-40 to evaluate the effect of additional tasks, and compare our methods with PEARL trained on the same tasks to evaluate our method's performance.

5.3 Learning, transfer, and generalization

Of the 28 total tasks (7 directions and 4 crippled legs) in `CrippledLegDirectionAnt`, 12 are sampled for training such that each direction and crippled leg is seen at least once. A sample of 5 remaining combinations is used to evaluate weak generalization in the next section. For `DirectionAnt`, 7 training directions are seen during training.

First, we compare the training performance of the LV models to the Specialist baseline to measure positive transfer. The average performance across tasks is plotted against the total number of timesteps taken across all tasks in Figure 4. In both experiments, the LV models learn significantly faster across tasks than the Specialist, indicating that the latent variables facilitate information sharing into the global neural network models. This is confirmed by the poor performance of the Generalist that also sees all the tasks but is unable to pool information effectively, yielding agents that perform worse than the Specialist. In addition to modeling the reward, our controller also handles early termination of an episode (see Sec. 3.5), whereas other model-based methods fail (Wang et al. 2019). Accordingly, even the Specialist is state-of-the-art under these conditions, outperforming similar models proposed int he literature on our tasks. Second, we compare to PEARL benchmarks and observe that the LV models are $> 10\times$ more sample efficient than the most efficient off-policy meta-RL algorithm that we are aware of. Finally, the difference between PEARL and PEARL-40 is small, suggesting that training is unaffected by fewer tasks, but as we will see (Sec. 5.3), still negatively impacts generalization.

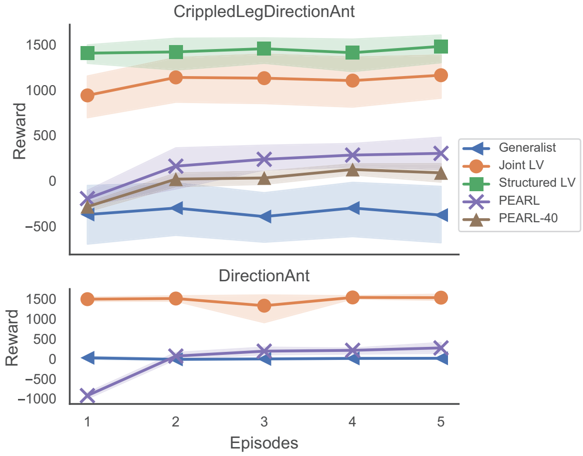

Figure 6: Episode reward on a novel direction.

Weak generalization: To test for weak generalization, 5 tasks are sampled from novel combinations of crippled leg and direction in `CrippledLegDirectionAnt` (excluding the holdout direction) and evaluated for 5 episodes at test-time. For LV models, the same objective function (either (9) or (10)) is minimized except that only variational parameters are updated. Every 10 steps on the test task, 100 iterations of SVI are performed on observed data at $5\times$ the learning rate.

To address the gap in sample efficiency between model-based and model-free methods, we evaluate all models after the maximum amount of training so that the LV models are compared to PEARL with much more training data. This shifts the focus to the realized reward for a fairer comparison. Despite far less training data and similar training performance, Figure 5 reveals that both LV models outperform the meta-RL baseline regardless of the training regimen. The LV models also infer quickly, performing well on the first episode, whereas PEARL is designed to perform well after one or two episodes. (Note that we did not attempt to tune PEARL's inference to perform well in the first episode, and ours was tuned to maximize average reward in the first 3 episodes.) There is, however, no difference between the joint and structured LV models. We hypothesize the latter will scale better with even more factors of variation.

Strong generalization: In this experiment, trained models infer the eight holdout directions at test-time to evaluate generalization to a novel task. In `CrippledLegDirectionAnt`, the agent must also infer which leg is crippled, further raising the difficulty (Recall that each leg was crippled at least once during training).

Figure 6 shows the LV models significantly outperform the baselines, supporting the claim that the GHP-MDP effectively models variations across these tasks. The structured LV model is the best performing, consistent with our hypothesis that causally factorized latent variables can improve generalization under some circumstances. Surprisingly, PEARL trained with fewer tasks fares slightly better than PEARL-40, indicating that a model trained with more tasks, which aids weak generalization to similar tasks (Figure 5, slightly biases the policy against out-of-distribution tasks that require extrapolation.

6 Discussion

In this work we have introduced the GHP-MDP, which can capture hidden structure in the dynamics and reward functions of related MDPs. We demonstrate this modeling approach on continuous control tasks with dynamics and reward variations that surpass strong baselines in performance and sample efficiency. In future work, it would be interesting to study the extent to which one can model more factors of variation and disentangle them automatically, obviating the specification of structure upfront.

References

Al-Shedivat, M.; Bansal, T.; Burda, Y.; Sutskever, I.; Mordatch, I.; and Abbeel, P. 2017. Continuous adaptation via meta-learning in nonstationary and competitive environments. *CoRR* abs/1710.03641.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H. P.; and Silver, D. 2017. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, 4055–4065.

Bellman, R. 1957. A Markovian Decision Process. *Indiana Univ. Math. J.*

Camacho, E. F., and Bordons, C. 2013. *Model Predictive Control*. Springer Science & Business Media.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. *NeurIPS* arXiv:1805.12114v1.

Clavera, I.; Nagabandi, A.; Fearing, R. S.; Abbeel, P.; Levine, S.; and Finn, C. 2018. Learning to Adapt: Meta-Learning for Model-Based Control. *CoRR* abs/1803.11347.

De Boer, P.-T.; Kroese, D. P.; Mannor, S.; and Rubinstein, R. Y. 2005. A tutorial on the cross-entropy method. *Annals of operations research* 134(1):19–67.

Deisenroth, M. P., and Rasmussen, C. E. 2010. Reducing model bias in reinforcement learning.

Deisenroth, M. P. 2011. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*.

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

Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P. L.; Sutskever, I.; and Abbeel, P. 2017. Rl^2: Fast reinforcement learning via slow reinforcement learning. *ICLR* arXiv:1611.02779.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *Proceedings of the 34th International Conference on Machine Learning* abs/1703.03400.

Fu, J.; Levine, S.; and Abbeel, P. 2016. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *IEEE International Conference on Intelligent Robots and Systems*.

Gelada, C.; Kumar, S.; Buckman, J.; Nachum, O.; and Bellemare, M. G. 2019. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, 2170–2179.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2555–2565.

Hausman, K.; Springenberg, J. T.; Wang, Z.; Heess, N.; and Riedmiller, M. 2018. Learning an embedding space for transferable robot skills. *ICRL*.

Hu, H.; Chen, L.; Gong, B.; and Sha, F. 2018. Synthesized policies for transfer and adaptation across tasks and environments. *Advances in Neural Information Processing Systems 31* 1168–1177.

Killian, T. W.; Daulton, S.; Konidaris, G.; and Doshi-Velez, F. 2017. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in Neural Information Processing Systems*, 6250–6261.

Kingma, D. P., and Welling, M. 2014. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *NIPS* arXiv:1612.01474.

Lazaric, A. 2012. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*. Springer. 143–173.

Nagabandi, A.; Finn, C.; and Levine, S. 2019. Deep online learning via meta-learning: Continual adaptation for model-based RL. In *International Conference on Learning Representations*.

Perez, C. F.; Such, F. P.; and Karaletsos, T. 2018. Efficient transfer learning and online adaptation with latent variable models for continuous control. *Continual Learning Workshop, NeurIPS 2018* abs/1812.03399.

Rakelly, K.; Zhou, A.; Quillen, D.; Finn, C.; and Levine, S. 2019. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *CoRR* abs/1903.08254.

Ranganath, R.; Gerrish, S.; and Blei, D. M. 2013. Black box variational inference. *arXiv preprint arXiv:1401.0118*.

Ravi, S., and Larochelle, H. 2017. Optimization As a Model for Few-Shot Learning. In *International Conference on Learning Representations 2017*.

Rothfuss, J.; Clavera, I.; Schulman, J.; Asfour, T.; and Abbeel, P. 2018. Model-Based Reinforcement Learning via Meta-Policy Optimization. *CoRL* arXiv:1809.05214v1.

Sæmundsson, S.; Hofmann, K.; and Deisenroth, M. P. 2018. Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*.

Schmidhuber, J. 1987. Evolutionary Principles in Self-Referential Learning. *On learning how to learn: The meta-meta-... hook.)* ....

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(Jul):1633–1685.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033.

Wang, T.; Bao, X.; Clavera, I.; Hoang, J.; Wen, Y.; Langlois, E.; Zhang, S.; Zhang, G.; Abbeel, P.; and Ba, J. 2019. Benchmarking model-based reinforcement learning.

Yao, J.; Killian, T.; Konidaris, G.; and Doshi-Velez, F. 2018. Direct policy transfer via hidden parameter markov decision processes. *LLARLA Workshop, FAIM 2018*.

Zhang, A.; Satija, H.; and Pineau, J. 2018. Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689*.