Generalized Hidden Parameter MDPs for Model-based Meta-Reinforcement Learning

Christian Perez[*1] **Felipe Petroski Such**[*1] **Theofanis Karaletsos**[1]

Abstract

Model-based reinforcement learning relies on powerful dynamics and reward surrogate models which capture a system, such that planning in the surrogate system can lead to control in the true system. We describe parametrized families of MDPs modeled with structured latent spaces with improved ability to transfer knowledge, generalize to new tasks, and tackle combinatorial problems exhibiting partial similarities to different past systems. We explain our method on a CartPole example with multiple latent factors and demonstrate its power on a challenging task on Ant using reward and dynamics latent spaces.

1. Introduction

Consider an illustrative problem of an agent with a certain pattern of broken actuators acting in an environment exhibiting certain weather and aiming to achieve one of many goals. We would like to learn to generalize to combinations of weather, joint states and goals to other, unseen combinations of those without requiring to learn an MDP from scratch. In order to tackle this problem we propose to use structured latent variables to parametrize families of MDPs to perform tasks which can be explained by combinations of latent factors of variation. In particular, we introduce multiple latent variables capturing factors of variation which can act on any function in the MDP.

Previous work on latent variable MDPs focused on small discrete action spaces (Doshi-Velez & Konidaris, 2016; Killian et al., 2017; Yao et al., 2018). A similar idea was presented utilizing Gaussian Processes (Sæmundsson et al., 2018), but operates under significantly less challenging experimental conditions due to the limitations of Gaussian Process models. Another notable recent use of latent variable inference in RL is latent skill embeddings in order to adapt to different goals (Hausman et al., 2018). We extend recent work using single latent variables for dynamics adaptation on hard continuous control tasks (Perez et al., 2018). Whereas all of these approaches use latent variables to model uncertainty in either dynamics or reward, we use structured latent variable models of both dynamics and reward to tackle meta-RL problems where each can vary and introduce a general model class with multiple separate factors of variation, as opposed to the limitation on single factors acting on dynamics only imposed in previous work.

Meta-learning, or learning to learn, is one solution that enables RL agents to learn quickly across different tasks (Schmidhuber, 1987; Ravi & Larochelle, 2017; Al-Shedivat et al., 2017; Rothfuss et al., 2018). Recently, meta-learning has been used to adapt a dynamics model (Clavera et al., 2018) for model-based control, or learn a policy (Rothfuss et al., 2018; Rakelly et al., 2019) that adapts to out-of-distribution tasks and dynamically changing environments, adjusting the model or policy in response to recent experience. However, these methods typically are inefficient in terms of their sample efficiency for training as the meta-learner needs to be trained first, while purely model-based methods like ours are targeting scenarios where low sample complexity is desirable.

2. Model-based RL

We consider a reinforcement learning problem described by a Markov decision process (MDP) comprising a state space, action space, transition function, reward function, and initial state distribution: $\{\mathcal{S}, \mathcal{A}, T, R, \rho_0\}$ (Bellman, 1957).

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

$$\pi^*(a|s) = \text{argmax}_\theta \mathbb{E}_{a_{t'} \sim \pi_\theta(a|s)} \sum_{t'=0}^{H-1} R(s_{t'}, a_{t'})$$
$$\text{s.t. } s_{t'+1} \sim T(s_{t'}, a_{t'}),$$

where T describes a probability distribution over next states in a stochastic environment.

[*]Equal contribution [1]Uber AI Labs, San Francisco, CA.. Correspondence to: Theofanis Karaletsos <theofanis@uber.com>.

Generative Modeling and Model-Based Reasoning for Robotics and AI Workshop, ICML 2019.

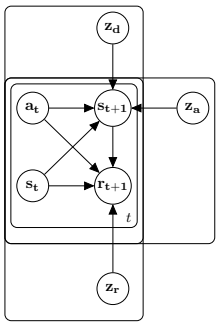

Figure 1. Probabilistic graphical model of MDP with structured latent variables for dynamics, agent variation and reward functions.

While it is common to assume known reward functions and even dynamics, one can simultaneously learn approximate models of the dynamics $p_{\theta_d}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \approx T$ and reward $p_{\theta_r}(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \approx R$ with parameters θ_d and θ_r, respectively (see Appendix B). By using model predictive control, for instance via trajectory sampling and CEM (see Appendix D), an approximately optimal policy (conditioned on the model of dynamics and reward) can be executed to collect more data to improve the models and, indirectly, the policy (Perez et al., 2018; Chua et al., 2018; Clavera et al., 2018). In this way, models are improved iteratively in-between rounds of data collection on the true task. This basic approach is effective for control of a single environment, but is not designed to learn and control multiple related tasks. We describe how to overcome this shortcoming using latent variable models in the following sections.

3. Generalized Hidden Parameter MDPs

We explore partially observable MDPs (POMDPs) that can be described as a family of MDPs where environment dynamics T and reward R are parameterized by hidden parameters $\eta \in \mathbb{R}^n$, expressed as T_η and R_η. In typical meta-RL settings, an agent learns across tasks where only the reward function R varies, for example, as a function of some unobserved goal position, direction, or velocity (Finn et al., 2017). In our formulation, all of the tasks form a POMDP in which the unobserved goal function can be parameterized by a hidden parameter, R_η. In adaptive dynamics settings, an agents learns across tasks where the transition function T varies, for example, by changing terrain, or agent actions, or observations (Clavera et al., 2018; Fu et al., 2016). Again, we model each variation as an instance of the same POMDP with hidden parameters affecting the transition function, T_η. In this way, we can solve tasks where either or both of these vary. We denote POMDPs which are fully described by hidden parameters as *Generalized Hidden Parameter MDPs* (**GHP-MDP**).

Note that T_η and R_η do not have to use all the dimensions in η and could form disjoint subsets $\eta = \{\eta_1, \eta_2\}$ for T_{η_1} and R_{η_2}. In this case, there are distinct hidden parameters that affect the transition function and reward function separately.

The GHP-MDP motivates the use of latent variables to model the hidden parameters of the dynamics and reward function. In Sec. 3.1, we describe the simplest probabilistic model of a GHP-MDP that uses a single continuous latent variable $\mathbf{z} \in \mathcal{R}^D$ to model hidden parameters of both the dynamics and reward. Because a single z jointly models all unobserved parameters, we call this the *joint latent variable (JLV)* model in later sections. In Sec. 3.2, we describe a probabilistic model that utilizes multiple latent variables $\{\mathbf{z}_d, \mathbf{z}_a, \mathbf{z}_r\} \in \mathcal{R}^D$ (shown in Fig. 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 (SLV)* model. For details on learning and inference, see Appendix B & C.

3.1. Joint Latent Variables: Meta-RL without meta-learning

For any GHP-MDP, we can model the dynamics and reward hidden parameters jointly with a single latent variable $\mathbf{z} \in \mathcal{R}^D$. The transition and reward model can either by known or learned. In general, for continuous state spaces we assume:

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

with diagonal covariance. See Appendix B for details on how the functions $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are learned. The model for episode return $\mathbf{R} = \sum \mathbf{r}_t$ for a trajectory decomposed into partial rewards $\mathbf{r}_t$ is

$$p(\mathbf{R}|\mathbf{s}_{0:H+1}, \mathbf{a}_{0:H}, \mathbf{z}) = \prod_t p_{\theta_r}(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}), \tag{2}$$

and the resulting joint model mapped over trajectories $\{\mathbf{s}_{0:H+1}, \mathbf{a}_{0:H}\}$ is

$$\begin{aligned} &p(\mathbf{s}_{0:H+1}, \mathbf{a}_{0:H}, \mathbf{R}, \mathbf{z}) = \\ &p(\mathbf{z})p(\mathbf{s}_0) \prod_{t=0}^{H-1} p(\mathbf{r}_{t+1}|\mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z})p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z})p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}) \end{aligned} \tag{3}$$

over an episode of length H.

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.

Instead of learning a parametric policy, we use the model for planning via model predictive control, setting $p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}) = \pi^*(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})$. We elaborate on our approach for π^* using MPC in detail in Appendix D.

3.2. Structured latent variables: efficient generalization through structure

In the previous section we assume a global variable $\mathbf{z}$ which is fed into both dynamics and reward model to generalize hidden parameter MDPs. Here, we generalize this setting further and assume that we can have multiple *plated variables* which constitute the latent space of the GHP-MDP, as explained in Sec. 3.

It is intuitive to establish separate latent spaces for dynamics and reward because we may expect agents to pursue different goals across different dynamics and each of these aspects can vary independently (see corresponding experiments in Sec. 4.1 and Sec. 4.2). We implement this model with a factorized latent space into two latent variables $\mathbf{z}_d$ and $\mathbf{z}_r$ to separately model hidden variability in the dynamics $T_{\mathbf{z}_d}$ and reward $R_{\mathbf{z}_r}$, respectively. Typically, only one or the other is varied in meta-RL tasks for continuous control (Clavera et al., 2018; Finn et al., 2017; Rakelly et al., 2019).

In this case, $\mathbf{z} = \{\mathbf{z}_r, \mathbf{z}_d\}$ with $\mathbf{z}_r \in \mathcal{R}^{D_1}$ and $\mathbf{z}_d \in \mathcal{R}^{D_2}$, such that the joint model is

$$
\begin{aligned}
p(\mathbf{s}_{0:T+1}, \mathbf{a}_{0:T}, \mathbf{R}, \mathbf{z}_d, \mathbf{z}_r) = \\
p(\mathbf{z}_d)p(\mathbf{z}_r)p(\mathbf{s}_0)\prod_{t=0}^{T}\Big[p(\mathbf{r}_{t+1}|\mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}_r) \\
p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_d) \\
p(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}_r, \mathbf{z}_d)\Big].
\end{aligned} \tag{4}
$$

More generally we may have $\mathbf{c}$ plated contexts, such as variations in the agent, environment, reward, etc. (see Fig. 1 and Sec. 4.1 for an example of $c = 3$.) Then for the set of latent variables $\{\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_c\}$, 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. By combining different states of these individual factors of variation into a rich joint state the model is also afforded with the ability to express combinatorial spaces of variation between these factors. An example of separating sub-structure in the dynamics part of a system can be seen in Sec. 4.1, where multiple latent factors model individual effects that can be modeled separately.

4. Experiments

We first test our method in a toy example (`CartPole`). For simplicity, we assume a known, differentiable transition and reward function with known but unobserved "hidden" parameters. This setting is analogous to test time conditions in later experiments where a dynamics and reward model has been learned in a prior training phase.

We later compare the joint and structured LV model against a state-of-the-art model-free adaptive method (PEARL) in the MuJoCo (Todorov et al., 2012) Ant environment that has variations in both dynamics and reward structure. For this experiment, we learn the dynamics and reward functions as well as hidden variables per task.

4.1. Toy example of GHP-MDP

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 (see Appendix E.)

The experiment consists of only two tasks with different hidden parameters (Fig. 2 left and right column.) 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 et al., 2018). The transition function $T_\eta(\mathbf{s}_t, \mathbf{a}_t; \eta_a, \eta_l, \eta_x)$ takes as parameters: η_a that scales the action/control signal (the force), the length of the pole η_l, and the position of the pole tip η_x (colors correspond to parameters in Fig. 2). We

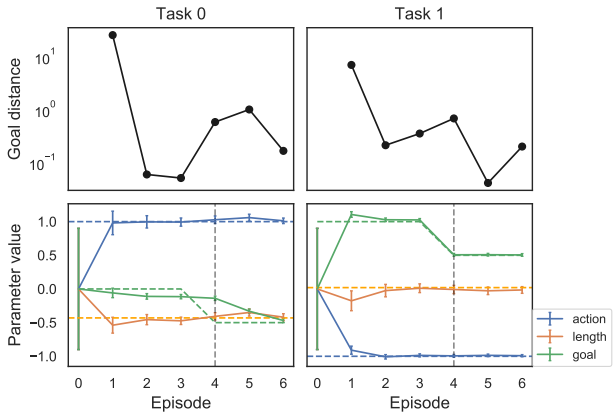

Figure 2. Inference of latent variables. **Top:** Distance to the goal tip position for two CartPole tasks with different hidden parameters (pole length $\in \{0.5, 0.7\}$.) **Bottom:** Mean of posterior of latent variables (solid) and the true values (dashed).

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 softplus mapping $\eta = \log(1 + \exp(\mathbf{z}))$.

To test the system, the goal position is suddenly changed in

episode 4. A corresponding change in only z_r is observed in Fig. 2. 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. As Fig. 2 shows, our GHP-MDPs can handle this simplified setting easily through inference on latent variables.

4.2. Generalization on Ant tasks

In `CrippledLegDirectionAnt`, our models must learn 1) which of 4 legs is crippled, and 2) which of 8 directions to travel. In total, there 32 possible tasks. The joint LV models dynamics and reward using a single latent variable. The structured LV models them separately.

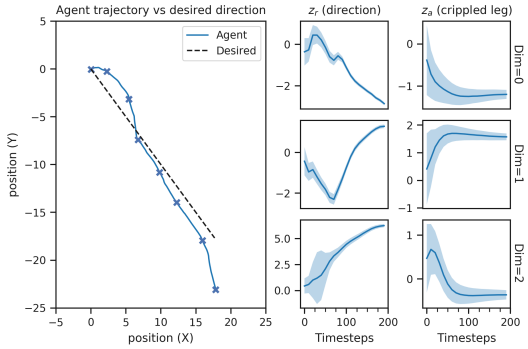

Figure 3. **Left:** XY position of Ant agent in first episode of inferring the crippled leg and a novel reward direction. A $\times$ marks every 25 steps. **Right:** Posterior ($\mu \pm \sigma$) of 3D latent variables for direction and crippled leg via Online Inference every 10 steps.

To test generalization to novel combinations of dynamics and rewards, we evaluate the reward on combinations that were not observed at test time. Training covers all crippled legs but only seven of eight reward directions for a total of 12 tasks. At test time, we evaluate on the other 20 novel tasks. Fig. S1 in Appendix A shows our models generalize to novel combinations not seen during training time. Our models also generalize to an unseen direction (Fig. 4) using $10\times$ fewer samples than PEARL (Fig. S2). Confirming the value of factorized latent spaces, the structured LV model outperforms the joint LV and both strong baselines (described in Appendix A).

In addition to episode reward, we examine the agent's movement and latents for the structured LV model on test environments in Fig. 3 for one task and Fig. 5 for all. How well the agent moves in the desired direction indicates how well the model can infer an appropriate reward latent. While inferring both the crippled leg and direction, the agent manages to move in the correct direction eventually most of the time. One direction was never seen during training, but both latent variable models generalize to this goal and its combinations with all crippled leg states.

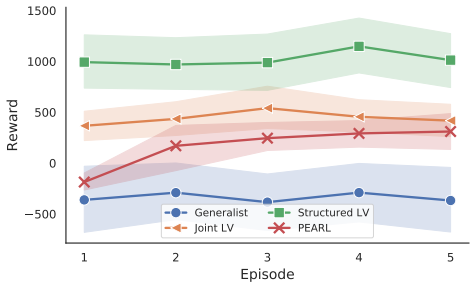

Figure 4. Strong generalization on a previously unseen directions across crippled legs. 95% CIs over 5 seeds.

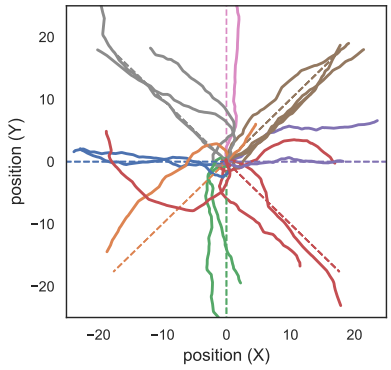

Figure 5. Agent paths from first episode on 20 *novel combinations* of directions (grouped by colors) and crippled legs. The red **South-East** direction was not observed during training but can still be inferred at test time.

5. Discussion

In this work we have introduced the Generalized Hidden Parameter MDP, a model which is able to capture latent structure in dynamics and reward functions. We demonstrate the usefulness of such spaces in a didactic toy example and challenging continuous control task with a known dynamics function with multiple hidden physical parameters and goals. In both cases our approach can solve tasks in few samples by performing inference on its structured latent space. We show that it can capture combinatorial spaces of variation elegantly by performing inference on each latent factor while acting to maximize reward. In future work, we want to study the extent in which we can combine more factors of variation and how to do so efficiently. In this paper, we assume that latent variables are either shared or distinct. We leave learning how to disentangle these factors of variation from data to future work.

References

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

Bellman, R. A Markovian Decision Process. *Indiana Univ. Math. J.*, 1957. ISSN 01650114. doi: 10.1007/BF02935461.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

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

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

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

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

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

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

Fu, J., Levine, S., and Abbeel, P. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *IEEE International Conference on Intelligent Robots and Systems*, 2016. ISBN 9781509037629. doi: 10.1109/IROS.2016.7759592.

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

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

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

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *NIPS*, arXiv:1612.01474, 2017. ISSN 10495258. URL http://arxiv.org/abs/1612.01474.

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

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

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

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

Ravi, S. and Larochelle, H. Optimization As a Model for Few-Shot Learning. In *International Conference on Learning Representations 2017*, 2017. ISBN 0953-5438. doi: 10.1016/j.electacta.2007.11.046.

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

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

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

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

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

A. Experiment details

Episodes of Ant are ≤ 500 steps and terminate early unless the torso height satisfies $0.3 \leq z_{\text{torso}} \leq 1.0$. Models are trained for 20 epochs every 250 steps using Adam with learning rate 5×10^{-4}.

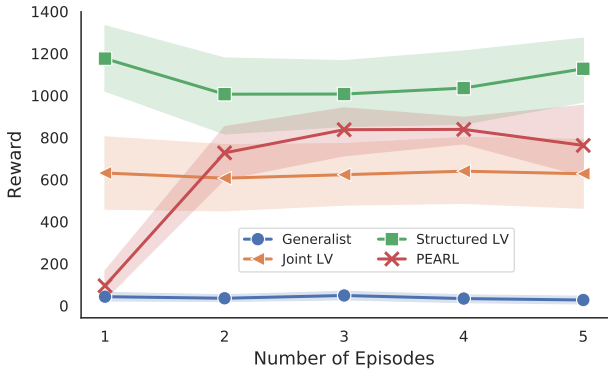

Figure S1. Mean reward after several episodes of test time inference on novel combinations of crippled legs and directions.

We compare the structured and joint latent variable models to two baselines:

- **Generalist**: our model-based agent baseline (Chua et al., 2018) that learns dynamics and rewards on all training environments but lacks latent variables. This is our negative control; we expect it cannot solve these tasks because it lacks the ability to model hidden parameters.
- **PEARL**: a sample-efficient model-free algorithm that also uses latent "context" variables (Rakelly et al., 2019) given $10\times$ the training data as our method.

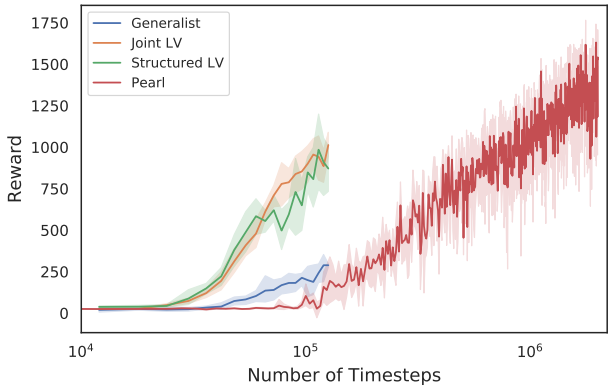

Figure S2. Reward during training.

Dynamics ensembles (N=5) have 256 units in 3 hidden layers in ensembles of size 5. Reward ensembles (N=5) have 200 units in 1 hidden layer. All networks use Swish activation (Ramachandran et al., 2017). Other hyperparameters follow those used in (Clavera et al., 2018).

B. Model learning

We use a generative model of transition dynamics $T_\eta(\mathbf{s}_t, \mathbf{a}_t)$ and reward $R_\eta(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. Because these are continuous quantities, they can be modeled with a Gaussian likelihood parameterized by mean μ_θ and covariance Σ_θ via a neural

network f with parameters θ_d for T and θ_r for R. Here as elsewhere, instead of predicting $\mathbf{s}_{t+1}$, a neural network predicts the change in the states $\Delta_s = \mathbf{s}_{t+1} - \mathbf{s}_t$ given the state and action $f_{\theta_d} \doteq p(\Delta_s|\mathbf{s}_t, \mathbf{a}_t; \theta_d)$.

Algorithm 1 also outlines the learning procedure. An episode of trajectories and rewards is collected given the current policy and concatenated with all data collected so far $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{r}_1), ..., (\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{r}_{t+1})\}$. Using $\mathcal{D}$, the dynamics and reward model paramet are then optimized with respect to Equation 8 via Adam optimizer for a fixed number of epochs (line 1).

In order to be robust to model misspecification and handle the small data setting, one can model uncertainty about parameters θ and marginalize to obtain

$$\begin{aligned} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) &= \int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t; \theta_d) p(\theta_d) d\theta_d \\ p(\mathbf{r}_t|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) &= \int p(\mathbf{r}_t|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}; \theta_r) p(\theta_r) d\theta_r \,. \end{aligned} \tag{5}$$

Commonly, this is done using Bayesian Neural Networks (BNNs). However, BNNs can be computationally expensive to infer and viable tractable approximations oftentimes underperform. A practical way to approximate the posterior $p(\theta|\mathcal{D})$ (with $\theta \in \{\theta_r, \theta_d\}$) is by approximating model uncertainty through frequentist ensembles of predictors, each trained on a different bootstrap or shuffle of the training data (Lakshminarayanan et al., 2017). For an ensemble with $|E|$ many members and the collection of all parameters from each network $\Theta \doteq \{\theta_0, ..., \theta_{|E|}\}$, we can define a model of the next state predictive distribution:

$$\begin{aligned} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t; \Theta) &= \frac{1}{|E|} \sum_{i=0}^{|E|} p_{\theta_i}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \\ &\approx p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \,. \end{aligned} \tag{6}$$

The reward model follows similarly. Crucially, we can use ensembles as a high-performance replacement for Bayesian Neural Networks of Equation 5.

In this work, each member of the ensemble is trained on distinct shuffles of the same data, and are reshuffled at every epoch as suggested in (Lakshminarayanan et al., 2017). In contrast, (Chua et al., 2018) use bootstrap samples.

C. Variational inference

Formally, for the *joint latent variable model*, the intractable distribution $p(\mathbf{z}|\mathcal{D})$ is approximated with $q_\phi(\mathbf{z})$ parameterized by a multivariate normal where $\phi = \{\mu_q, \Sigma_q\}$. The learning objective is to maximize the marginal likelihood of observed transitions with respect to θ and ϕ. We can maximize the evidence lower bound (**ELBO**) to the marginal log-likelihood:

$$\begin{aligned} \log p(\mathcal{D}) &= \sum_{t=0}^{T} \big[\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \big] \\ &\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})} \left[\sum_{t=0}^{T} \big[\log p_{\theta_d}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}) + \log p_{\theta_r}(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}) \big] \right] - \mathrm{KL}\big(q_\phi(\mathbf{z})||p(\mathbf{z})\big). \end{aligned} \tag{7}$$

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 et al., 2013; Kingma & Welling, 2014) and subsample in order to perform inference and learning via gradient descent, yielding the training objective:

$$\mathcal{L}(\theta_d, \theta_r, \phi) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T} [\log p_{\theta_d}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}^m) + \log p_{\theta_r}(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}^m)] - \mathrm{KL}\big(q_\phi(\mathbf{z})||p(\mathbf{z})\big) \tag{8}$$

with $\mathbf{z}^m \sim q_\phi(\mathbf{z})$ and number of samples $M = 2$. Our objective is to maximize $\mathcal{L}(\theta_d, \theta_r, \phi)$ with respect to $\{\theta_d, \theta_r, \phi\}$ for training and ϕ at test time.

For structured latent variable models with plated contexts, Equation 8 can be extended to multiple latent variables. For the graphical model of Fig. 1 with three factors of variation—environment, agent, and reward—the objective function becomes

$$\begin{aligned}\mathcal{L}(\theta_d, \theta_r, \Phi) = &\frac{1}{M}\sum_{m=1}^{M}\sum_{t=0}^{T}\left[\log p_{\theta_d}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_d^m, \mathbf{z}_a^m) + \log p_{\theta_r}(\mathbf{r}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{z}_r^m)\right] \\ &- \mathrm{KL}\big(q_{\phi_d}(\mathbf{z}_d)||p(\mathbf{z}_d)\big) - \mathrm{KL}\big(q_{\phi_a}(\mathbf{z}_a)||p(\mathbf{z}_a)\big) - \mathrm{KL}\big(q_{\phi_r}(\mathbf{z}_r)||p(\mathbf{z}_r)\big)\,,\end{aligned} \tag{9}$$

where each of $\{\mathbf{z}_d^m, \mathbf{z}_a^m, \mathbf{z}_r^m\}$ are sampled from their respective approximate posteriors $q(\cdot)$ and $\Phi = \{\phi_d, \phi_a, \phi_r\}$.

D. 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 $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. 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 $\sum_t^{t+h} \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim p(\mathbf{s_t}, \mathbf{a}_t)}[R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{z})]$ at run-time (Camacho & Bordons, 2013). 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 conditions.

Algorithm 1 Learning and control with Model Predictive Control

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_{θ_d} and reward model p_{θ_r} with $\mathcal{D}$ using Equation 7
5: Initialize starting state $\mathbf{s}_0$ and episode history $\mathcal{D}_k = \varnothing$
6: **for** Time $t = 0$ to TaskHorizon {MPC loop} **do**
7: **for** Iteration $i = 0$ to MaxIter {CEM loop} **do**
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 particle state particle s_p
10: Propagate next state $\mathbf{s}_{t+1,p} \sim p_{\theta_d}(\cdot|\mathbf{s}_{t,p}, \mathbf{a}_t, \mathbf{z})$ using TS-∞ {See (Chua et al., 2018)}
11: Evaluate expected reward $\sum_{\tau=t}^{t+h} {}^1\!/_P \sum_{p=1}^{P} \hat{\mathbf{r}}_{\tau+1}$ where $\hat{\mathbf{r}}_{\tau+1} \sim p_{\theta_r}(\cdot|\mathbf{s}_{\tau,p}, \mathbf{a}_t, \mathbf{s}_{\tau+1,p}, \mathbf{z}^p)$
12: Update $\mathrm{CEM}(\cdot)$ distribution
13: **end for**
14: Execute first action $\mathbf{a}_t$ determined by final $\mathrm{CEM}(\cdot)$ distribution
15: Record outcome $\mathcal{D}_t \leftarrow \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{r}_{t+1})\}$
16: Record outcome $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \mathcal{D}_t$
17: Update approximate posterior $q_\phi(\mathbf{z}|\mathcal{D}_t)$ using Equation 8
18: **end for**
19: Update data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_k$
20: **end for**

Algorithm 1 describes a learning 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 500 proposed action sequences $\mathbf{a}_{t:t+h}$ 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 1), 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 CEM algorithm to incorporate the auxiliary latent variable in our model into the planner, effectively mapping to a policy $\pi^*(\cdot|\mathbf{z}_k)$.

For each particle, the latent variable $\mathbf{z}_k \sim q_\phi(\mathbf{z}_k|\mathcal{D}_t)$ is sampled from approximate posterior so that planning can also account for the auxiliary latent variable capturing beliefs about unknown conditions.

On a new environment, we skip training line 1 to keep the dynamics model p_{θ_d} fixed. Our task then is to iterate between acting at step t and inferring $p(\mathbf{z}|\mathcal{D}_t)$ in order to align the expected dynamics with the current system the agent is acting in as it changes.

E. CartPole Pyro model

The function `model()` is the Pyro (Bingham et al., 2018) model for a structured latent variable model of a GHP-MDP. The variable `dyn_rew_model` is an instance of `DynamicsCartPoleModel` that uses the true dynamics and reward of the environment (with fixed observation noise) conditioned on the hidden parameters. The `CartPoleEnv` from OpenAI `gym` (Brockman et al., 2016) was modified to make a `_step()` on the environment differentiable, add scaling of actions, and to add rewards for reach a goal tip position.

```
1
2 def model(s_t, a_t, s_tp1, rew_tp1, env_ids, dyn_rew_model, dim_latents=[1,1,1]):
3     "Multiple latent variables model."
4     dyn_rew_model = pyro.module('dyn_rew_model', dyn_rew_model)
5
6     num_latents = len(dim_latents)
7     indices_by_latent = env_ids.unbind(-1)
8     assert len(indices_by_latent) == num_latents
9
10    zs = []
11    for i, latent_indices in enumerate(indices_by_latent):
12        ids, idx = torch.unique(latent_indices, return_inverse=True, sorted=True)
13        dim_latent = dim_latents[i]
14
15        mu = s_t.new_zeros((len(ids), dim_latent))
16        sigma = s_t.new_ones((len(ids), dim_latent))
17        z_latent = pyro.sample(f'z_{i}', Normal(mu, sigma))
18        z_latent = z_latent[idx]
19        zs.append(z_latent)
20
21    with pyro.plate(f'mdp'):
22        # z_e, z_a, z_r = zs
23        next_state, reward = dyn_rew_model(s_t, a_t, *zs)
24        pyro.sample('s_tp1',
25                    next_state.to_event(-1),
26                    obs=s_tp1)
27        pyro.sample('rew',
28                    reward.to_event(-1),
29                    obs=rew_tp1)
30
31
32 class DynamicsCartPoleModel(torch.nn.Module):
33     constraint_pos = {'length', 'm', 'gravity'}
34
35     def __init__(self,
36                  state_space: int,
37                  action_space: int,
38                  dim_latents: list,
39                  layers: List[int]=[20],
40                  ensemble_size: int=1):
41         super().__init__()
42         self.dim_latents = dim_latents
43         self.dynamics = CartPoleEnv()
44
45     def forward(self, s_t, a_t, *zs):
46         for z, attr in zip(zs, ['action_scale', 'length', 'goal_position']):
47             if attr in self.constraint_pos:
48                 z = softplus(z)
```

```
            setattr(self.dynamics, attr, z)
        next_state = Normal(self.dynamics._step(s_t, a_t), 1e-2)
        reward = Normal(self.dynamics.get_reward(s_t, a_t), 1e-2)
        return next_state, reward


class CartPoleEnv(gym.Env):
    def __init__(self, m=1, length=0.5, gravity=9.8, action_scale=1, goal_position=0):
        self.gravity = gravity
        self.masscart = m
        self.masspole = 0.1
        self.action_scale = action_scale
        self.total_mass = (self.masspole + self.masscart)
        self.length = length # actually half the pole's length
        # self.polemass_length = (self.masspole * self.length)
        self.goal_position = torch.as_tensor(np.atleast_1d(goal_position))

    def _step(self, obs, action):
        action = torch.as_tensor(action)
        x, x_dot, theta, theta_dot = obs.unsqueeze(-2).unbind(-1)
        force = self.force_mag * torch.clamp(action * self.action_scale, -1, 1)
        costheta = torch.cos(theta)
        sintheta = torch.sin(theta)
        temp = (force + self.polemass_length * theta_dot * theta_dot * sintheta) / self.total_mass
        thetaacc = (self.gravity * sintheta - costheta*temp) / (self.length * (4.0/3.0 - self.masspole *
 costheta * costheta / self.total_mass))
        xacc  = temp - self.polemass_length * thetaacc * costheta / self.total_mass
        if self.kinematics_integrator == 'euler':
            x  = x + self.tau * x_dot
            x_dot = x_dot + self.tau * xacc
            theta = theta + self.tau * theta_dot
            theta_dot = theta_dot + self.tau * thetaacc
        else: # semi-implicit euler
            x_dot = x_dot + self.tau * xacc
            x  = x + self.tau * x_dot
            theta_dot = theta_dot + self.tau * thetaacc
            theta = theta + self.tau * theta_dot
        return torch.cat((x,x_dot,theta,theta_dot), dim=-1)


    def get_reward(self, obs, u):
        x, x_dot, theta, theta_dot = torch.as_tensor(obs).unsqueeze(-2).unbind(-1)
        theta = angle_normalize(theta)
        costs = 0.1*theta_dot**2 + .001*(u**2)
        if self.goal_position is not None:
            length = self.length
            x_pole = x + length * torch.sin(theta)
            y_pole = length * (1 - torch.cos(theta))
            costs += 5 * (y_pole**2 + (x_pole - self.goal_position)**2)
        reward = -costs.squeeze(-1)
        return reward

    def get_goal_distance(self, obs):
        x, x_dot, theta, theta_dot = torch.as_tensor(obs).unsqueeze(-2).unbind(-1)
        x_pole = x + self.length * torch.sin(theta)
        y_pole = self.length * (1 - torch.cos(theta))
        d2 = (y_pole**2 + (x_pole - self.goal_position)**2)
        return torch.sqrt(d2) / self.length
```